History, Analysis, and Implementation of Traveling Salesman Problem (TSP) and Related Problems

By Anne Maredia

Spring 2010

In Partial Fulfillment of

Math 4395-Senior Project

Department of Computer and Mathematical Sciences

University of Houston-Downtown

Faculty Advisor:

Dr. Timothy Redl _____

Committee Member:

Dr. Katarina Jegdic _____

Committee Member:

Dr. Ryan Pepper _____

Department Chairman:

Dr. Dennis Rodriguez _____

Table of Contents

Table of Figures

Table of Tables

Abstract

Suppose a salesperson needs to travel from a city to all the other cities exactly once to sell his products and return back to the city he started from. He wants to do this while traveling the minimum total distance. How can he do this? This project helps to answer that question using the Traveling Salesman Problem (TSP).

TSP is an application of graph theory. In terms of graph theory, given a list of cities and their pair-wise distance, the task is to find a shortest possible tour (Hamiltonian Cycle) that visits every city exactly once. Even though TSP is easy to understand, it is very difficult to solve. Researchers have proven that Traveling Salesman Problem is NP-complete.

This project involves a study of the history and growth of TSP, an analysis of solution methods of TSP, a discussion of other applications of TSP, an implementation of TSP using MATLAB, and an introduction to related problems such as Chinese Postman Problem. One of the main objectives of this project is to use MATLAB to implement some solution methods of TSP.

Acknowledgements

Special thanks to my advisor Dr. Timothy Redl for assisting me throughout the project. Without his help I would have not learned so much about Traveling Salesman Problem. I would like to thank Yusuke Motazawa for inspiring me to continue work in this area. I would also like to acknowledge both of my committee members, Dr. Katarina Jegdic and Dr. Ryan Pepper for their time and support. I would also like to thank my professor, Dr. Xie Shishen, and my classmates in senior project class for giving me inputs and opinions to help me improve my project and presentation.

Introduction

Graph Theory is the study of graphs. Graphs are mathematical structures used to model pair-wise relations between objects. Graph theory can be used to solve problems in branches of Mathematics, Computer Science, and other scientific areas. Graphs are one of the most useful mathematical objects. You can model an enormous number of real world systems and phenomena using graphs.

There are many different problems in graph theory that have attracted much attention. One such problem is the Traveling Salesman Problem (TSP), which refers to a salesman who wants to find a shortest possible tour that visits every city exactly once and returns back to the city from which he started. In terms of graph theory, given a list of nodes (cities) and their pair-wise distance, the task is to find the shortest possible tour that visits every node exactly once. The distance between every pair of cities is symmetric, because the direction of traversal of the given arc does not matter. Since the start node and the end nodes are the same the tour creates a Hamiltonian cycle, which is a cycle in the graph that visits each vertex exactly once.

Figure 1 shows the example of a Hamiltonian cycle on 10 nodes.

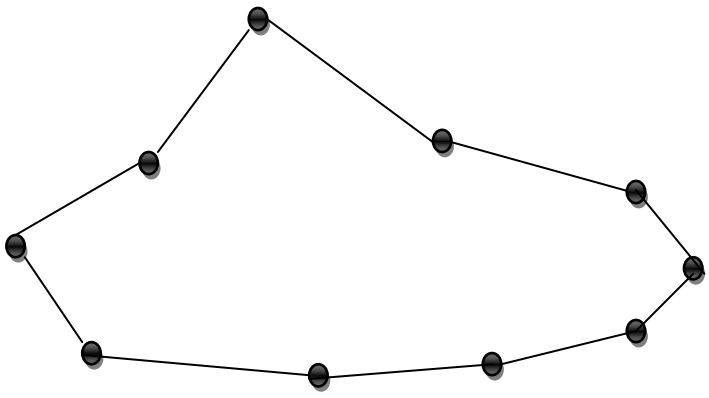

Figure 1: Hamiltonian cycle

History

Mathematical problems related to the Traveling Salesman Problem were studied in the 1800's by the Irish mathematician Sir William Rowan Hamilton and by the British mathematician Thomas Penyngton Kirkman. Hamilton created an Icosian game in 1857 that requires a player to complete a tour using only specified connectors through 20 points. [6]

TSP's were first studied in the 1930's by mathematician and economist Karl Menger in Vienna and Harvard. It was later investigated by Hassler Whitney and Merrill Flood at Princeton. [6]

Later, in 1940's the TSP was studied by statisticians Mahalanobis, Jessen, Gosh, and Marks in relation to agricultural application, and mathematician Merill Flood popularized it among his colleagues at the RAND Corporation. [6]

Solution methods of TSP began to appear in papers in the mid-1950s; these papers used a variety of minor variations of the term "Traveling Salesman Problem." Dantzig, Fulkerson, and Johnson referred to the "traveling-salesman problem" in 1954. Heller also used "travelling salesman's problem" in 1954. Morton and Land preferred "the travelling salesman problem" in 1955 and they originally called it the "laundry van problem." In the following decades, the problem has been studied by many researchers from mathematics, computer science, chemistry, physics, and other sciences. [6]

Although TSP is easy to understand, it is very difficult to solve. Richard M. Karp showed in 1972 that the Hamiltonian cycle problem was NP-complete (non-deterministic polynomial-time hard), which implies the NP-hardness of TSP. NP-hard, is a class of problems that are informally the hardest problems in NP which means no polynomial-time algorithm is known for

solving TSP. This supplied a scientific explanation for the apparent computational difficulty of finding optimal tours. [6]

The solution methods for TSP have become sophisticated, and solvable instances have become larger and larger. Below we present a brief history of TSP milestones.

Dantzig, Fulkerson, and Johnson published a description of a method for solving the TSP and illustrated the power of this method by solving an instance with 49 cities in 1954. They created this instance by picking one city from each of the 48 states plus Washington, D.C. in the U.S.A. Rather than solving this problem they solved the 42-city problem by excluding Baltimore, Wilmington, Philadelphia, Newark, New York, Hartford, and Providence. It turned out that an optimal tour through the 42 cities used the edge joining Washington, D.C. to Boston; since the shortest route between these two cities passes through the seven removed cities, and this solution of the 42-city problem yields a solution of the 49-city problem. [6]

Held and Karp solved a TSP using 64 cities in 1971. Later in 1975, Camerini, Fratta, and Maffioli solved a TSP with 67 cities. In 1977, Grotschel solved a TSP using 120 cities in and around Germany. [6]

In 1973 Lin and Kernighan obtained the data set from R. Habermann, and used 318 points that arose in a drilling application, where the drill was a pulsed laser. This instance was first solved by H. Crowder and M. W. Padberg in 1980. In 1987, Padberg and Rinaldi solved a TSP consisting of 532 AT&T switch locations in the USA. A TSP with 666 cities was first solved by Holland and Groetschel, appearing in Olaf Holland's 1987 PhD Thesis. The data set consists of interesting cities from around the world and the inter-city distances are specified by a function that approximates the great circle distances on the globe. Later in the same year,

Padberg and Rinaldi found the optimal tour through a layout of 2,392 points obtained from Tektronics Incorporated. [6]

In 1994, Applegate, Bixby, Chvatal, and Cook solved TSP containing 7,397 cities. Later in 1998, they solved it using 13,509 cities in United States. [6]

In 2001, Applegate, Bixby, Chvátal, and Cook found the optimal tour of 15,112 cities in Germany. Later in 2004, TSP of visiting all 24,978 cities in Sweden was solved; a tour of length of approximately 72,500 kilometers was found and it was proven that no shorter tour exists. This is currently the largest solved TSP. [6]

Table 1 summarizes the milestones of solving Traveling Salesman Problem.

Year	Research Team	Size of Instance
1954	G. Dantzig, R. Fulkerson, and S. Johnson	49 cities
1971	M. Held and R.M. Karp	64 cities
1975	P.M. Camerini, L. Fratta, and F. Maffioli	67 cities
1977	M. Grötschel	120 cities
1980	H. Crowder and M.W. Padberg	318 cities
1987	M. Padberg and G. Rinaldi	532 cities
1987	M. Grötschel and O. Holland	666 cities
1987	M. Padberg and G. Rinaldi	2,392 cities
1994	D. Applegate, R. Bixby, V. Chvátal, and W. Cook	7,397 cities
1998	D. Applegate, R. Bixby, V. Chvátal, and W. Cook	13,509 cities
2001	D. Applegate, R. Bixby, V. Chvátal, and W. Cook	15,112 cities
2004	D. Applegate, R. Bixby, V. Chvátal, W. Cook, and K. Helsgaun	24,978 cities

Table 1: Milestones of TSP [6]

Motivation

After taking a Decision Math course at University of Houston-Downtown, which involved the study of some network problems, I became more interested in learning about other applications and topics related to graph theory. Dr. Redl also encouraged me to look at the senior project of his fellow student Yusuke Motozawa, and advised me to continue his research on Traveling Salesman Problem (TSP), which I thought was very interesting.

Purpose

The purpose of the project is to introduce computation, modify and write code using MATLAB, implement and compare various algorithms, and heuristic solutions to solve TSP, and look at related problems.

The description of the report is as follows:

Section 1: This is an introductory section that basically discusses about the formulation of TSP and the structure of graphs. This section also focuses on the comparison of the number of tour and the number of arcs as the number of nodes increases.

Section 2: This section focus on different solution methods and heuristics used to solve TSP. It also discusses some examples and comparison of the different solution methods with heuristics.

Section 3: This section describes the implementation of Brute-force and Branch and Bound in MATLAB. It also focuses on the comparison of the computational results in MATLAB.

Section 4: This section focuses about the related problem to TSP. It also shows the similarities and differences between the Chinese Postman Problem and Traveling Salesman Problem.

1. Introduction of the Structure of TSP

1.1 Arcs, Edges, and Tours

Solving a TSP amounts to finding a minimal cost Hamiltonian cycle. Recall, even though TSP is easy to understand it is very difficult to solve. Suppose we have a complete graph with nodes representing cities, and arcs representing distances between the given cities. It can be verified that ——— . For example:

Nodes : cities = 4

Arcs : ——— = ——— = 6

Figure 2 illustrates such a network with 4 cities and 6 arcs.

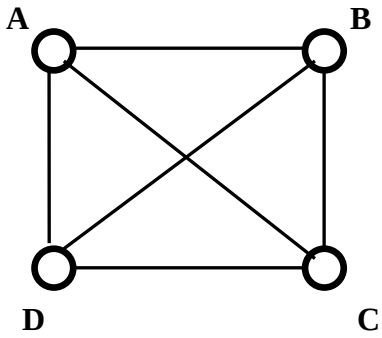

Figure 2: Network with 4 cities and 6 arcs

A complete graph with nodes has ——— distinct tours or Hamiltonian cycle. For example:

Nodes : cities = 4

Number of tours : ——— = ——— = 3

As the number of nodes increases, the number of tours also increases. For example: when and = 10, using the formula for the number of tours we get . We can see that just by adding one more node the number of tours increased by a factor of 3.

Table 2 shows how rapidly the number of increases as increases.

Nodes	Arcs ———	Number of Tours ———
6	15	60
7	21	360
8	28	2,520
9	36	20,160
10	45	181,440
11	55	1, 814, 400
12	66	19, 958, 400
13	78	239, 500, 800
14	91	3, 113, 510, 400
15	105	43, 589, 145, 600
16	120	653, 837, 184, 000
17	136	$1.04613949 \times 10^{13}$
18	153	$1.77843714 \times 10^{14}$
19	171	$3.20118685 \times 10^{15}$
20	190	$6.08225502 \times 10^{16}$

Table 2: The number of tours increases as *n* increases.

To see the results more clearly we wrote a MATLAB program that plots number of *n* nodes versus the number of *a* arcs, and number of *n* nodes versus number of *t* tours, when *n* is from 1to 20.

Figure 3 further illustrates the number of *n* nodes vs. *a* arc, and the number of *n* nodes vs. *t* tours.

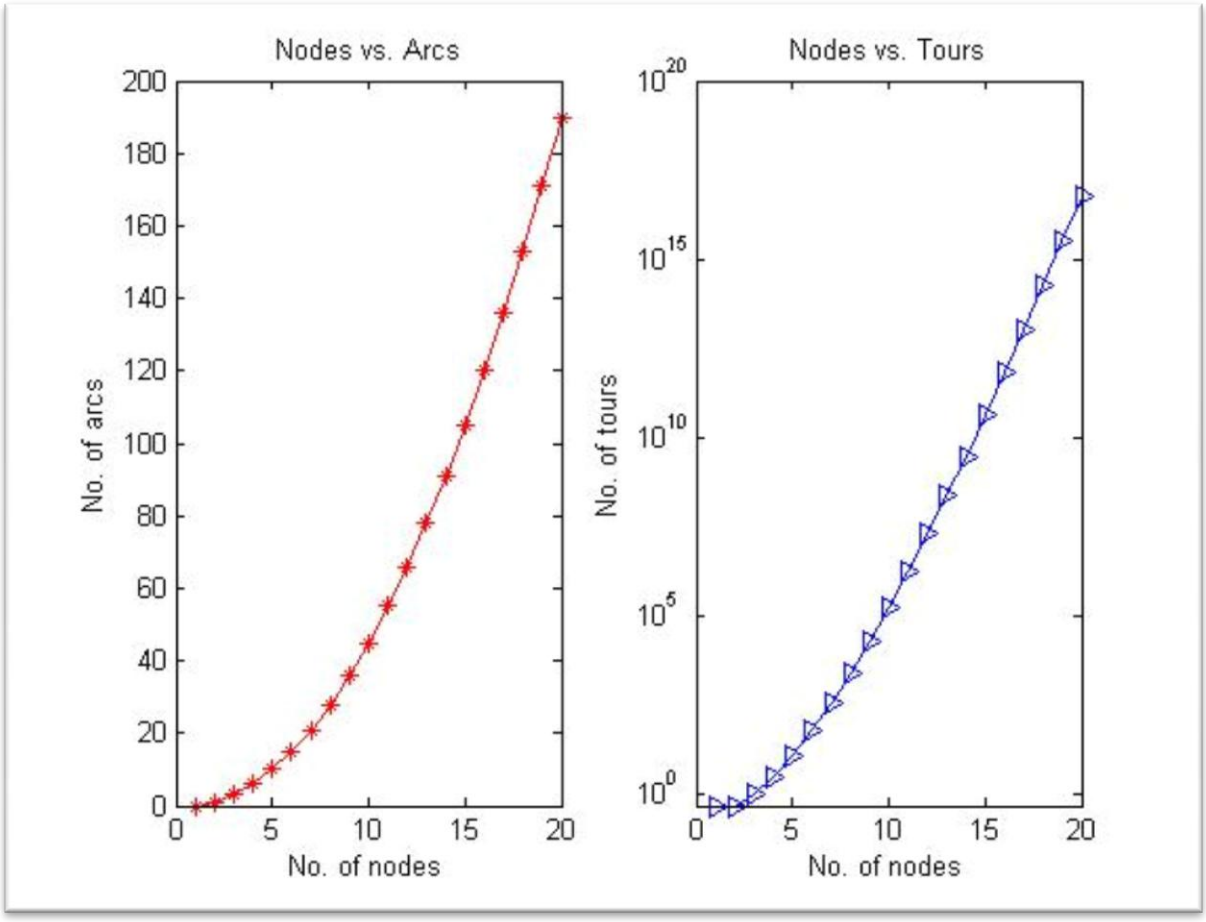

Figure 3: Graph of Nodes vs. arcs and Nodes vs. Tours.

We can see in the figure above that when $n = 3$, we get $a = 3$ and $t = 1$; therefore, as *n* nodes gets closer to 20 the number of *t* tours gets closer to 10^20, and *a* arcs gets closer to 200. This concludes that as the number of *n* increases, the number of *t* tours and the number of *a* arcs increases rapidly.

2. Solution methods of TSP

Introduction

Suppose a salesperson needs to travel from a city to all the other cities exactly once to sell his products and return back to the city he started from. He wants to do this while covering the minimum total distance. How can he do that? This is where solving the TSP comes in. Some solution methods of TSP include:

- Brute – force method.
- Approximations
    - Nearest Neighbor (Greedy)
    - Greedy Approach
- Branch and bound method.

2.1 **Brute-force method.**

When one thinks of solving TSP, the first method that might come to mind is a brute-force method. The brute-force method is to simply generate all possible tours and compute their distances. The shortest tour is thus the optimal tour. To solve TSP using Brute-force method we can use the following steps:

Step 1: calculate the total number of tours (= ——— where represents the number of nodes (cities).

Step 2: draw and list all the possible tours.

Step 3: calculate the distance of each tour.

Step 4: choose the shortest tour; this is the optimal solution.

Example 1: Suppose and as seen in Figure 3. Calculate TSP using Brute- force method.

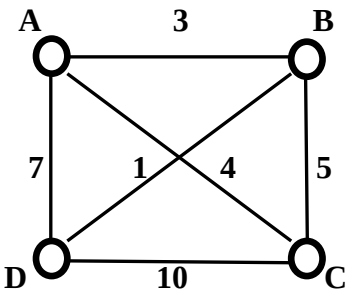

Figure 4: A 4 city TSP

Step 1: Number of tours = ——— = ——— 3.

Step 2 and 3:

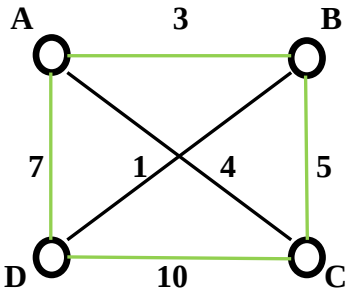

Figure 5: Tour 1 of a 4 city TSP

Tour 1: A→B→C→D→A with total distance of 3+5+10+7 = 25.

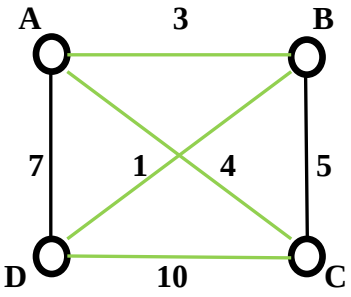

Figure 6: Tour 2 of a 4 city TSP

Tour 2: A→B→D→C→A with total distance of 3+1+10+4 = 18.

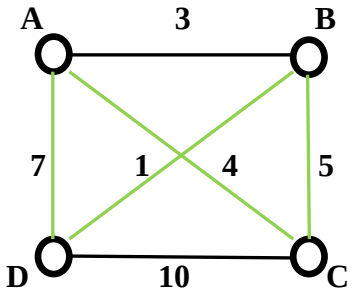

Figure 7: Tour 3 of a 4 city TSP

Tour 3: A→C→B→D→A with total distance of 4+5+1+7 = 17.

Step 4: From analyzing all of the three tours we see that it will be best to take Tour 3: A→C→B→D→A with the minimum total distance of 17.

Example 2: Calculate TSP of 5 cities in Figure 7 using Brute- force method.

nodes(cities) = 5.

arcs (distances) = 10.

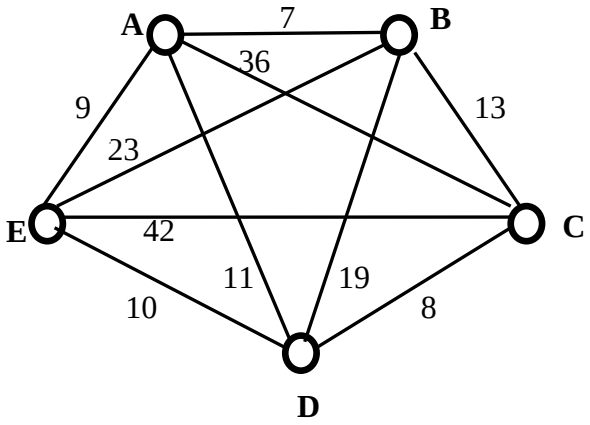

Figure 8: A 5 city TSP

Step 1: Number of tours = ——— ——— = 12.

Step 2 and 3:

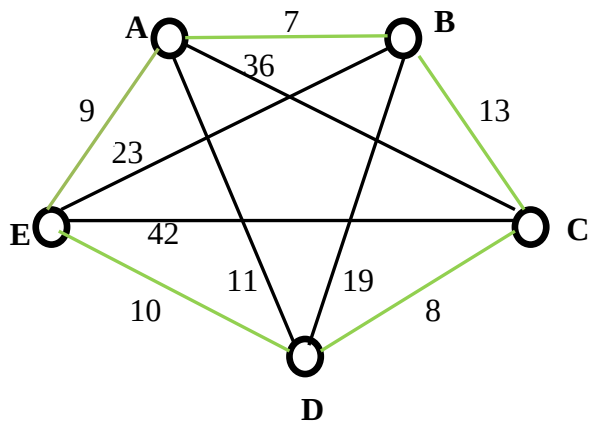

Figure 9: Tour 1 of a 5 city TSP

Tour 1: A→B→C→D→E→A

Distance of tour 1: 7+13+8+10+9 = 47

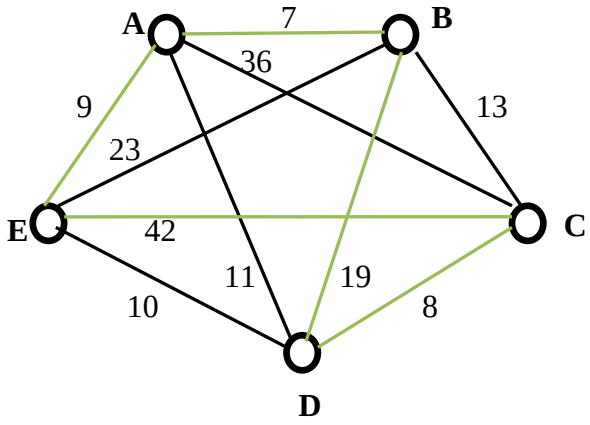

Figure 10: Tour 2 of a 5 city TSP

Tour 2: A→B→D→C→E→A

Distance of tour 2: 7+19+8+42+9 = 85

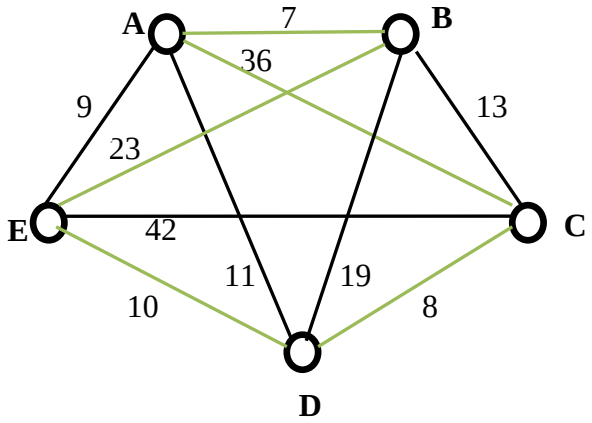

Figure 11: Tour 3 of a 5 city TSP

Tour 3: A→B→E→D→C→A

Distance of tour 3: 7+23+10+8+36 = 84

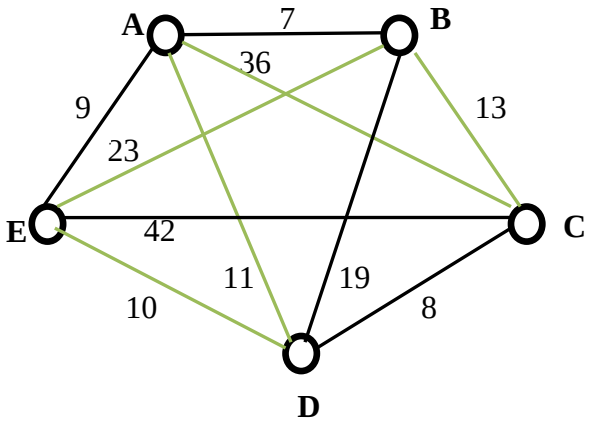

Figure 12: Tour 4 of a 5 city TSP

Tour 4: A→C→B→E→D→A

Distance of tour 4: 36+13+23+10+11 = 93

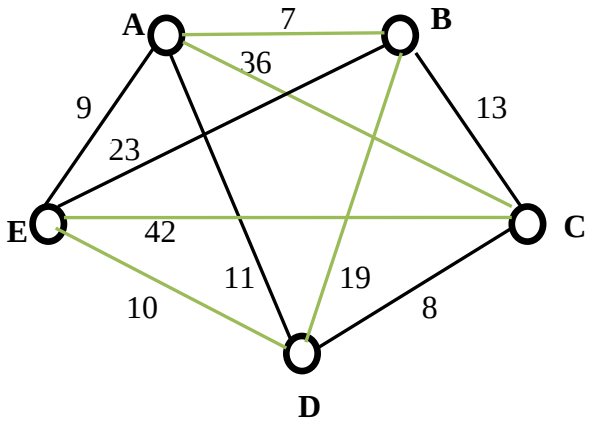

Figure 13: Tour 5 of a 5 city TSP

Tour 5: A→C→E→D→B→A

Distance of tour 5: 36+42+10+19+7 = 114

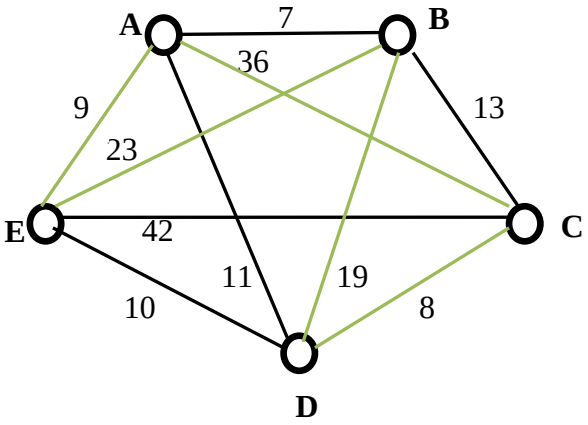

Figure 14: Tour 6 of a 5 city TSP

Tour 6: A→C→D→B→E→A

Distance of tour 6: 36+8+19+23+9 = 95

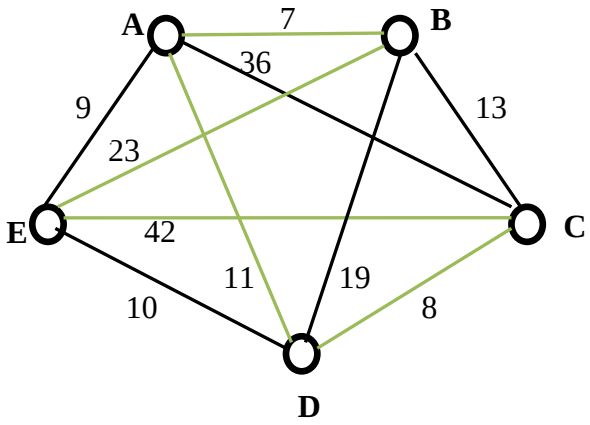

Figure 15: Tour 7 of a 5 city TSP

Tour 7: A→B→E→C→D→A

Distance of tour 7: 7+23+42+8+11 = 91

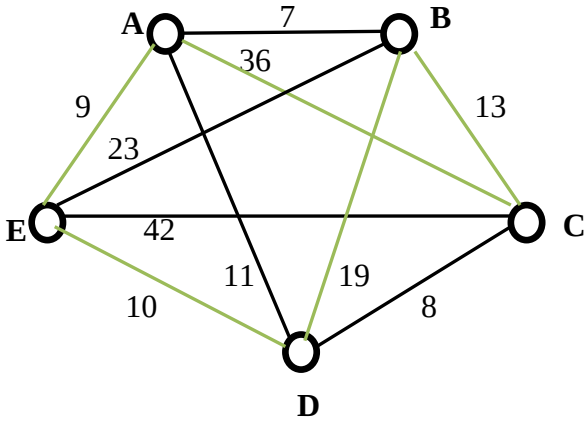

Figure 16: Tour 8 of a 5 city TSP

Tour 8: A→E→D→B→C→A

Distance of tour 7: 9+10+19+13+36 = 87

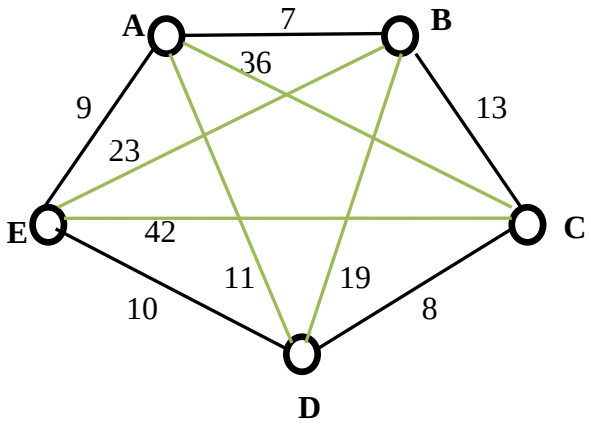

Figure 17: Tour 9 of a 5 city TSP

Tour 9: A→D→B→E→C→A

Distance of tour 9: 11+19+23+42+36 = 131

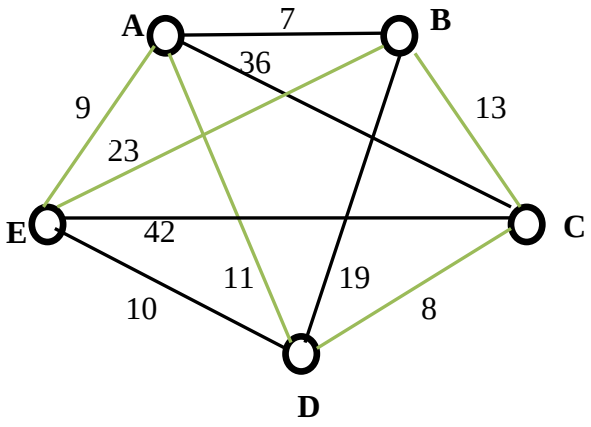

Figure 18: Tour 10 of a 5 city TSP

Tour 10: A→D→C→B→E→A

Distance of tour 10: 11+8+13+23+9 = 64

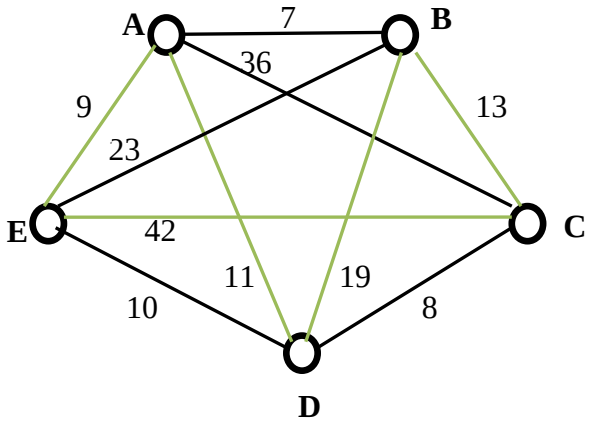

Figure 19: Tour 11 of a 5 city TSP

Tour 11: A→E→C→B→D→A

Distance of tour 11: 9+42+13+19+11 = 94

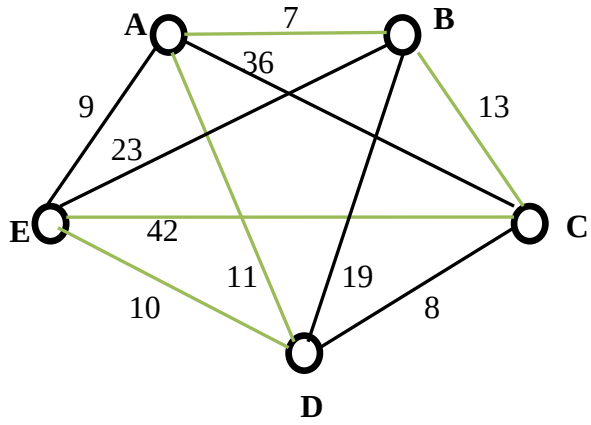

Figure 20: Tour 12 of a 5 city TSP

Tour 12: A→D→E→C→B→A

Distance of tour 12: 11+10+42+13+7 = 83

Step 4: From analyzing all of the twelve tours we see that it will be best to take Tour 1: A→B→C→D→E→A with the minimum total distance of 47.

Brute-force method always works if given enough time and care. Because of its nature, it is convenient for relatively small number of nodes. Recall, from Table 2, we know that as the number of nodes increases the number of tours increases factorially; therefore, using Brute-force method to solve TSP with a large number of nodes can be frustrating and can even take years or centuries to solve.

Table 3 shows the estimated time it could take to solve various TSP using Brute-force method.

Nodes	Number of tours	Estimated time
15	43,589,145,600	50 days
16	653,837,184,000	2 years
20	6.082255×10^{16}	193,000 years

Table 3: Estimated time to solve TSP using Brute-force method. [5]

Brute-force method is very inefficient as gets larger. We will also implement brute-force method in MATLAB using example 1 and 2 of this subsection which will be shown in section 3. We can also think of solving TSP using a Nearest Neighbor heuristic which is explained in next subsection. Heuristics are methods which cannot guarantee to produce optimal solutions, but which, we hope, produces fairly good solutions. [7]

2.2 **Nearest Neighbor (Greedy)**

The Nearest Neighbor heuristic, is a simple approach for solving the Traveling Salesman Problem. To solve TSP with a Nearest Neighbor heuristic we look at all the arcs coming out of the city (node) that have not been visited and choose the next closest city, then return to the starting city when all the other cities are visited. To solve TSP using Nearest Neighbor Heuristic we can use the following steps:

Step 1: Pick any starting node.

Step 2: Look at all the arcs coming out of the starting node that have not been visited and choose the next closest node.

Step 3: Repeat the process until all the nodes have been visited at least once.

Step 4: Check and see if all nodes are visited. If so return to the starting point which gives us a tour.

Step 5: Draw and write down the tour, and calculate the distance of the tour.

Example 1: Suppose and as seen in Figure 8. Calculate TSP using Nearest Neighbor heuristic.

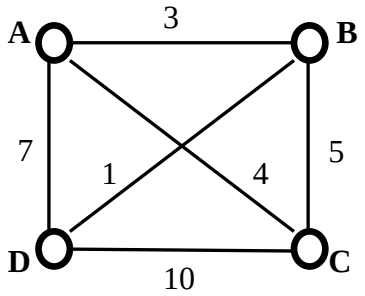

Figure 21: Network of a 4 city TSP

Step 1: Suppose we start at city A.

Step 2: there are three arcs coming out of city A, first one is arc A→B with distance of 3, second one is arc A→C with distance of 4 , and the third one is arc A→D with distance of 7. The next closest city is the one which has the minimum total distance, which is arc A→B with a distance of 3; therefore, the next closest city is B.

Step 3: Now from B, we have three arcs coming out of it. We can not go back to A because we have already visited that city. So we have two options we can either go to city C or city D. The arc coming out of B which has the minimum distance is the next closest city; therefore B→D is the next closest city with the minimum distance of 1.

Now from D we can not visit cities A and B since we have already visited those cities. Our only option is to visit city D→C with distance of 10. Even though D→C is the arc with the largest distance, it is our only option to choose that arc; therefore D→C is the next closest city with the distance of 10.

Step 4: since we have visited all the cities, now we can finally return from city C to city A which has the distance of 4.

Step 5:

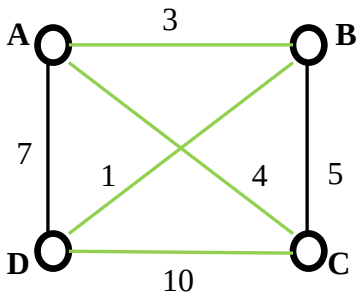

Figure 22: Tour of a 4 city TSP using Nearest Neighbor Heuristic

Tour: A→B→D→C→A

Distance of the tour: 3+1+10+4 = 18.

By comparing the above example to example 1 in sub section 2.1 we see that the optimal solution for Brute-force method is 17 and optimal solution using Nearest Neighbor heuristic is 18. Therefore it follows that Nearest Neighbor heuristic works but it does not necessarily give you the best solution as Brute-force method.

Example 2: Suppose and as seen in Figure 9. Calculate TSP using Nearest Neighbor heuristic.

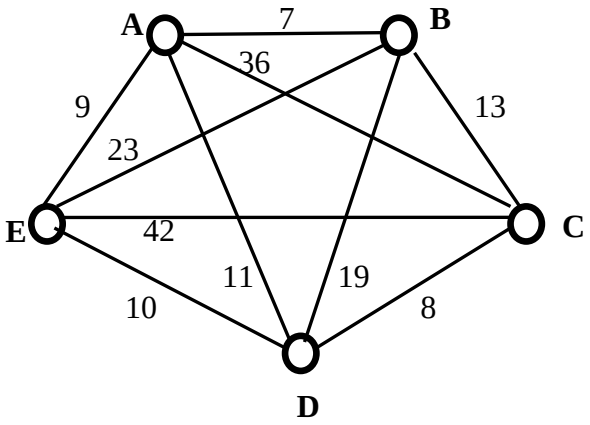

Figure 23: Network of a 5 city TSP

Step 1: Suppose we start with city A.

Step 2: there are four arcs coming out of city A, first one is arc A→B with distance of 7, second one is arc A→C with distance of 36 , and the third one is arc A→D with distance of 11, and the fourth one is arc A→E with distance of 9. The next closest city is B with the distance of 7.

Step 3: from city B, we have four arcs coming out. We can not go back to A since we have already visited that city. We can either go to city C, city D, or city E. We pick city C since the next closest city from city B is C with the distance of 13.Now from C we can not go back to city A and city B, since we have already visited those cities. Therefore,

the closest city from city C is D with the distance of 8, and from D the next closest city is E with the distance of 10.

Step 4: since we have visited all the cities, now we can finally return from city E to city A which has the distance of 9.

Step 5:

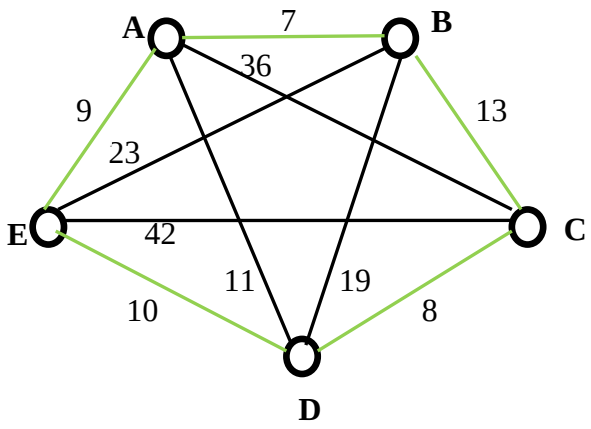

Figure 24: Tour of a 4 city TSP using Nearest Neighbor Heuristic

Tour: A→B→C→D→E→A

Distance of tour: 7+13+8+10+9=47

By comparing the above example to example 2 in sub section 2.1 we see that the optimal solution for Brute-force method is same as the optimal solution of Nearest Neighbor heuristic which is 47. For this example Nearest Neighbor actually worked, but it will not always be the case. Therefore, by solving example 1 and 2 in this sub section, it follows that Nearest Neighbor Heuristic works but it does not always give you the best solution as Brute-force method.

2.3 Greedy Approach

The Greedy Approach can be the first method which can be used to solve TSP. To solve TSP using Greedy Approach, we look at all the arcs coming out of the city (node) and choose the n cheapest arcs. If those n cheapest arcs forms a Hamiltonian cycle than we have an optimal solution. To solve TSP using Greedy Approach we can use following steps:

Step1: Look at all the arcs with minimum distance.

Step 2: Choose the *n* cheapest arcs

Step 3: List the distance of arcs starting from the minimum distance to maximum distance.

Step 4: Draw and check if it forms a Hamiltonian cycle.

Step 5: If step 4 forms a Hamiltonian cycle than we have an optimal solution; write down the tour of the optimal solution and calculate their distance.

Example 1: Suppose and as seen in Figure 24. Calculate TSP using Greedy Approach.

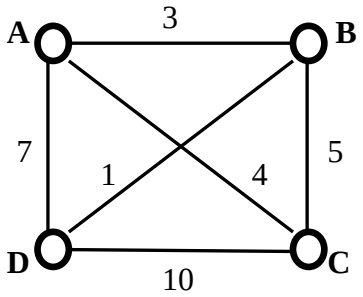

Figure 25: A 4 city TSP network

Step1: the arcs with minimum distances are arc DB, arc BA, arc AC, arc CB.

Step 2: we choose *n* cheapest arcs and they are DB, BA, AC, and CB.

Step 3: arc DB = 1, the next smallest distance is BA = 3, arc AC = 4 is the next smallest distance, and the fourth smallest distance is arc CB = 5. We stop at arc CB because we already picked *n* cheapest arcs.

Step 4 and 5:

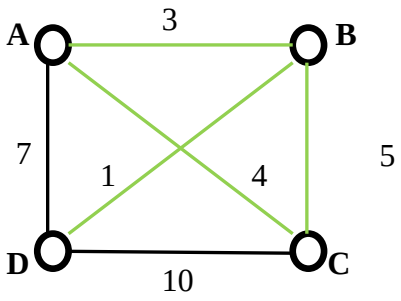

Figure 26: A greedy approach with 4 cities

We do not have Hamiltonian cycle; therefore, we do not have a tour.

Example 2: Suppose and as seen in Figure 26. Calculate TSP using greedy approach.

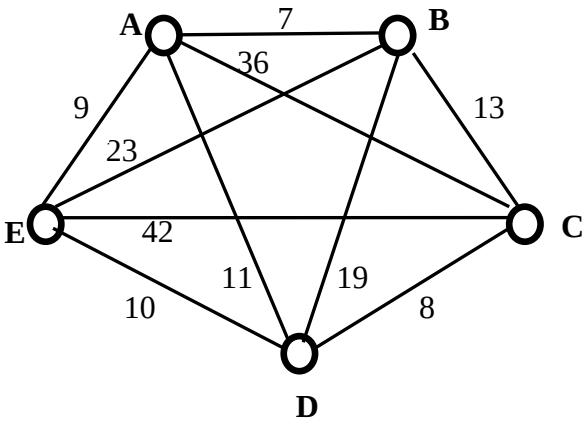

Figure 27: A 5 city TSP network

Step 1: the arcs with minimum distances are arc AB, arc CD, arc AE, arc ED, arc DA, and arc BC.

Step 2: we choose *n* cheapest arcs and they are AB, CD, AE, ED, and DA.

Step 3: arc AB = 7, the second smallest distance is CD = 8, arc AE = 9 is the third smallest distance, the fourth smallest distance is arc ED = 10, and the fifth smallest distance is arc DA = 11. We stop at arc DA because we already picked *n* cheapest arcs.

Step 4 and 5:

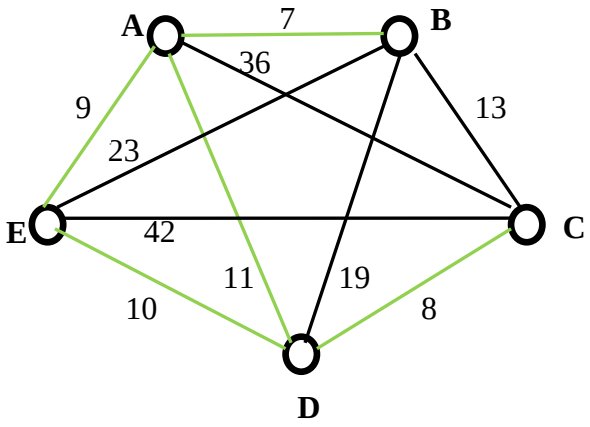

Figure 28: A greedy approach with 5 cities

We do not have Hamiltonian cycle; therefore, we do not have a tour.

We conclude that this method is a good approach, but it does not give us the optimal solution. Therefore the greedy approach is not a good approximation.

2.4 Branch and Bound

The Branch and Bound method implicitly enumerates all the feasible solutions, using calculations where the integer constraints of the problems are relaxed. In other words the branch and bound strategy divides a problem to be solved into a number of sub-problems. It is a system for solving a sequence of sub-problems each of which may have multiple possible solutions and where the solution chosen for one sub-problem may affect the possible solutions of later sub-problems. To avoid the complete calculation of all partial trees, we first try to find a practical solution and note its value as an upper bound for the optimum. As the distance exceeds the distance of the upper bound the calculations are done. If a new cheaper solution was found, its value is used as the new upper bound. This method is convenient for 40 to 60 nodes (cities). To solve TSP sing Branch and Bound we can use following steps:

Step 1: Choose a start node.

Step 2: Set bound to a very large value, let's say infinity.

Step 3: Choose the cheapest arc between the current and unvisited node and add the distance to the current distance and repeat while the current distance is less than the bound.

Step 4: If current distance is less than bound, then we are done

Step 5: Add up the distance and bound will be equal to the current distance.

Step 6: Repeat step 5 until all the arcs have been covered.

3. Implementations of TSP in MATLAB

3.1 Implementation of Brute-force.

We implemented brute-force in MATLAB using example 1 of subsection 2.1, where

and We wrote a function bf4 (A) which takes in the matrix A and gives us the tour length of each tour and all three tours. It also gives us an optimal tour length which is 17 and it also tells us that tour 3 is an optimal tour which is 1, 3, 2, 4, 1. We also get the time it takes to solve this example, which is 0.0116 seconds. Below we have the MATLAB solution for

```
>> A = [0 3 4 7; 3 0 5 1; 4 5 0 10; 7 1 10 0]

A =

    0    3    4    7
    3    0    5    1
    4    5    0   10
    7    1   10    0

>> bf4(A)

tour_length = 25

tour1 =1, 2, 3, 4, 1

tou_length = 25    18

tour2 = 1, 2, 4, 3, 1

tour_length = 25    18    17

tour3 = 1, 3, 2, 4, 1

tour = 1, 2, 3, 4, 1
       1, 2, 4, 3, 1
       1, 3, 2, 4, 1
```

```
tour = '1, 2, 3, 4, 1'
       '1, 2, 4, 3, 1'
       '1, 3, 2, 4, 1'

Opt_Tour_Length = 17

Opt_Tour = 3

Shortest_Tour = ‘1, 3, 2, 4, 1’

elapsed_time = 0.0116
```

We also implemented brute-force in MATLAB using example 2 of subsection 2.1, where and We wrote a function bf4 (B) which takes in the matrix B and gives us the tour length of each 12 tours and also all 12 tours. It gives us an optimal tour length, which is 47, and it tells us that tour 1 is an optimal tour, which is 1, 2, 3, 4, 5, 1. We also get the time it takes to solve this example, which is 0.0051 seconds. Below we have the MATLAB solution for

```
>> B= [0 7 36 11 9; 7 0 13 19 23; 36 13 0 8 42; 11 19 8 0 10; 9 23 42 10 0]

B =

     0     7    36    11     9
     7     0    13    19    23
    36    13     0     8    42
    11    19     8     0    10
     9    23    42    10     0

>> bf5(B)

Tour_length = 47

tour1 = 1, 2, 3, 4, 5, 1

tour_length = 47    85

tour2 = 1, 2, 4, 3, 5, 1
```

tour_length = 47 85 84

tour3 = 1, 2, 5, 4, 3, 1

tour_length = 47 85 84 93

tour4 = 1, 3, 2, 5, 4, 1

tour_length = 47 85 84 93 114

tour5 = 1, 3, 5, 4, 2, 1

tour_length = 47 85 84 93 114 95

tour6 = 1, 3, 4, 2, 5, 1

tour_length = 47 85 84 93 114 95 91

tour7 = 1, 2, 5, 3, 4, 1

tour_length = 47 85 84 93 114 95 91 87

tour8 = 1, 5, 4, 2, 3, 1

tour_length = 47 85 84 93 114 95 91 87 131

tour9 = 1, 4, 2, 5, 3, 1

tour_length = 47 85 84 93 114 95 91 87 131 64

tour10 = 1, 4, 3, 2, 5,1

tour_length = 47 85 84 93 114 95 91 87 131 64 94

tour11 = 1, 5, 3, 2, 4, 1

tour_length = 47 85 84 93 114 95 91 87 131 64 94 83

tour12 = 1, 4, 5, 3, 2, 1

tour = 1, 2, 3, 4, 5, 1
1, 2, 4, 3, 5, 1
1, 2, 5, 4, 3, 1
1, 3, 2, 5, 4, 1
1, 3, 5, 4, 2, 1
1, 3, 4, 2, 5, 1

```
        1, 2, 5, 3, 4, 1
        1, 5, 4, 2, 3, 1
        1, 4, 2, 5, 3, 1
        1, 4, 3, 2, 5, 1
        1, 5, 3, 2, 4, 1
        1, 4, 5, 3, 2, 1

tour =  '1, 2, 3, 4, 5, 1'
        '1, 2, 4, 3, 5, 1'
        '1, 2, 5, 4, 3, 1'
        '1, 3, 2, 5, 4, 1'
        '1, 3, 5, 4, 2, 1'
        '1, 3, 4, 2, 5, 1'
        '1, 2, 5, 3, 4, 1'
        '1, 5, 4, 2, 3, 1'
        '1, 4, 2, 5, 3, 1'
        '1, 4, 3, 2, 5, 1'
        '1, 5, 3, 2, 4, 1'
        '1, 4, 5, 3, 2, 1'

Opt_Tour_Length = 47

Opt_Tour = 1

Shortest_Tour =   '1, 2, 3, 4, 5, 1'

elapsed_time = 0.0051
```

Another MATLAB implementation of Brute-force. This code was originally written by Dr. Timothy Redl. We ran this code for *n = 3, …, 11*. We only have the output for *n=11*, since that is the largest number it worked for. This code gives us the elapsed time it takes to solve in MATLAB. We also get the best tour for the first 11 cities out of 34 cities. The tour starts from Twin Falls which is the 3rd city and it also gives us the optimal solution with 2610 miles and it also outputs the map with the optimal tour in red. Below we have the MATLAB solution for *n = 11*.

```
>> bestTour=bruteTour(11)

elapsed_time =

  118.9540

 The best tour is

    'Twin Falls'
    'Boise'
    'Butte'
    'Lewiston'
    'Portland'
    'Redding'
    'Reno'
    'Gustine'
    'Lone Pine'
    'Marble Canyon'
    'Mexican Hat'

 with the distance (in miles) of

  2.6102e+003

 bestTour =

     3     2     4     1     5     7     6     9     8    10    11
```

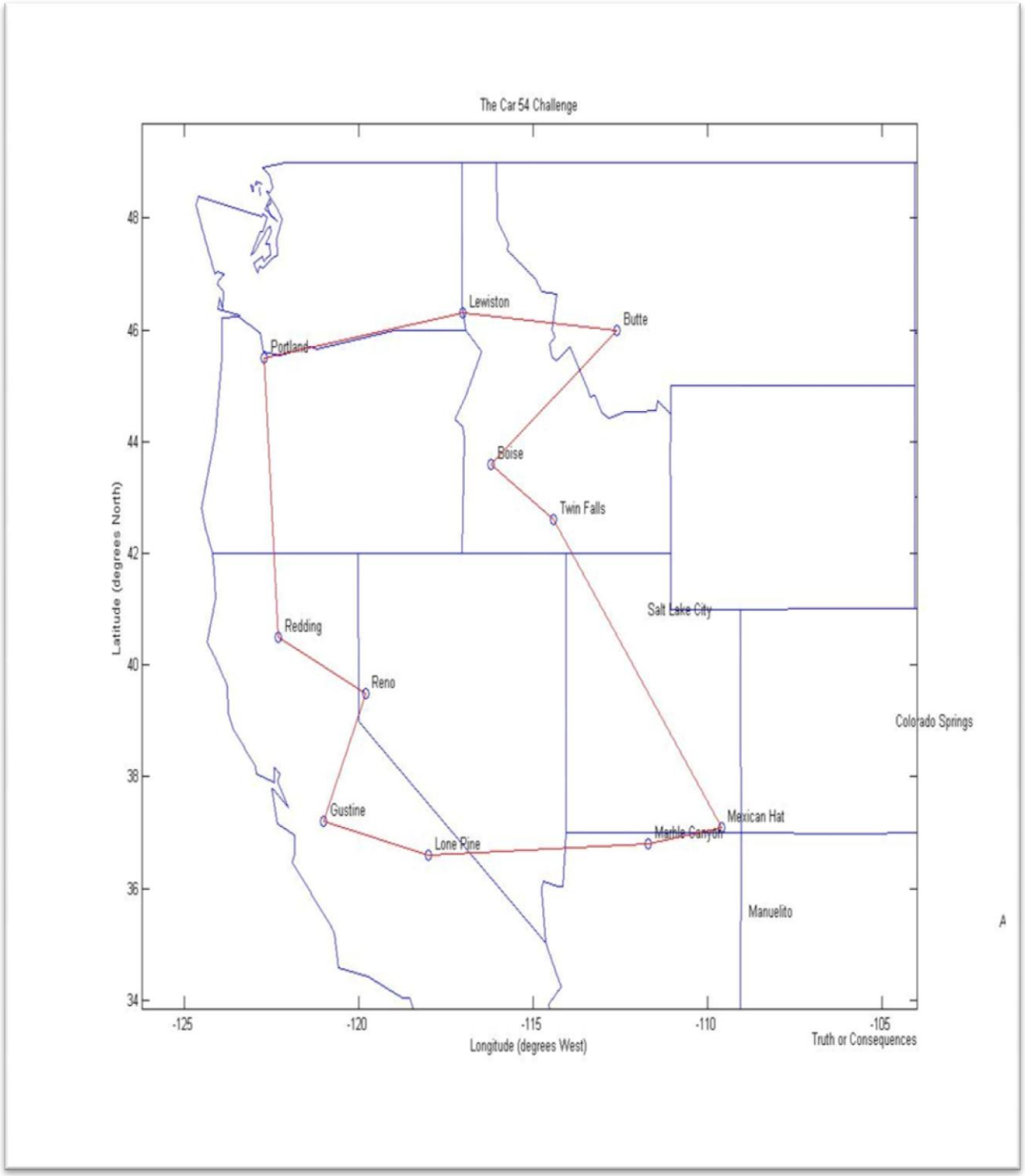

Figure 29: Optimal tour of 11 cities in MATLAB using Brute-force.

3.2 Implementation of Branch and Bound

MATLAB implementation of Branch and Bound. This code was originally written by Dr. Timothy Redl. We ran this code for *n = 3, …, 12*. We only have the output for *n=12*, since that is the largest number it worked for. This code gives us the elapsed time it takes to solve in MATLAB. We also get the best tour for the first 12 cities out of 34 cities. The tour starts from Lewiston which is the 1st city and then goes to the 4th city which is Butte and so on and finally ends at the fifth city which is Portland. We also get an optimal tour which is from 1→ 4→ 2→3 →12→11→10→8→9→6→7→5 with 2610 miles and it also outputs the map with the optimal tour in red. Below we have the MATLAB solution for *n = 12*.

```
>>tsp(12,'bnb')
Elapsed time is 42.873576 seconds.
The best tour is

    'Lewiston'
    'Butte'
    'Boise'
    'Twin Falls'
    'Salt Lake City'
    'Mexican Hat'
    'Marble Canyon'
    'Lone Pine'
    'Gustine'
    'Reno'
    'Redding'
    'Portland'

 with the distance (in miles) of

  2.6159e+003

ans =

     1     4     2     3    12    11    10     8     9     6     7     5
```

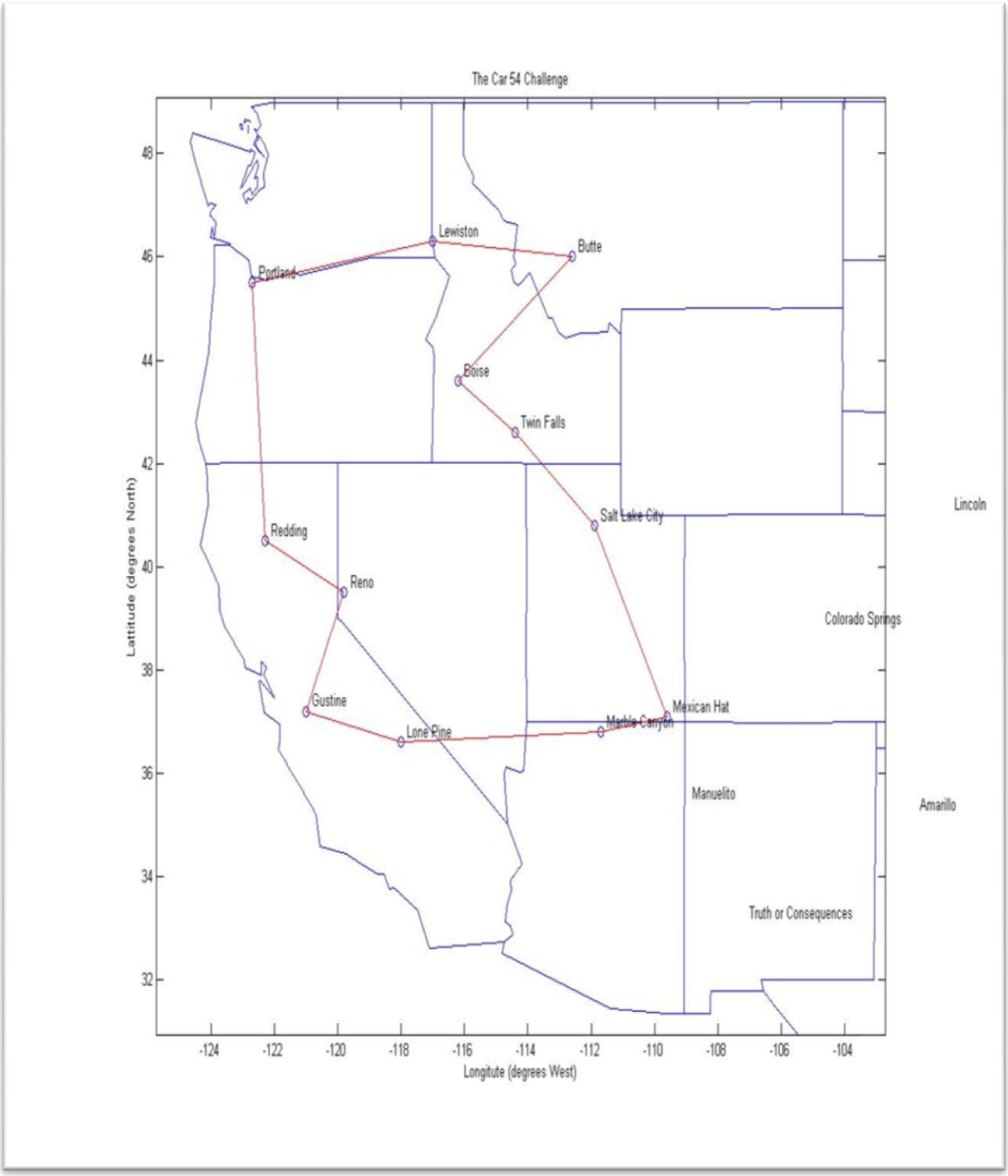

Figure 30: Optimal tour of 12 cities in MATLAB using Branch and Bound.

Since we ran the Dr. Redl's code for Brute-force and Branch and Bound for *n=12*. We can compare the timings it takes to solve it in MATLAB to see which method produces the results rapidly.

Table 4 shows us the comparison of the timing it takes to solve it in MATLAB.

Number of Nodes(*n*)	**Brute-force elapsed time in seconds**	**Branch and Bound elapsed time in seconds**
3	0.01	0.0004
4	0.01	0.0008
5	0.01	0.002
6	0.01	0.0077
7	0.0240	0.0338
8	0.1692	0.1603
9	1.3453	0.7107
10	11.9430	3.3361
11	118.9540	11.5930
12	…….	42.8735

Table 4: Computational results comparing Brute-force and Branch and Bound

By comparing the timings of Brute-force and Branch and Bound we can see that Branch and Bound method solves fast in MATLAB. When we ran the MATLAB code using Brute-force method for *n =12* it took time to give results, while as for Branch and Bound it took only 43 second to produce results.

4. Chinese Postman Problem (CPP)

Chinese Postman Problem is a related problem to Traveling Salesman Problem. Suppose a postman needs to deliver letters to a certain neighborhood. He wants to find a tour through the neighborhood covering the least possible total distance and returning to his starting point. One of the requirements is, the postman must travel through each road in his tour at least once, but should avoid covering too many roads.

In terms of graph theory, given a connected weighted graph, the task is to find the shortest tour that uses each edge exactly once. This problem, related to the TSP, was originally discussed by the Chinese mathematician Mei-Ku Kwan in 1962. [13] The Chinese Postman Problem is basically finding an Euler cycle. It is also known as the *Route Inspection Problem*. The CPP has many practical applications:

- Bus routing
- Trash collection
- Road sweeping
- Snow-plowing
- Transmission line inspection.

There are certain similarities and differences between Chinese Postman Problem and Traveling Salesman Problem, and that is the reason we call CPP a related problem to TSP.

Table 5 shows us the similarities and differences between TSP and CPP.

	TSP	CPP
Similarities	Start and end at a same node.	Start and end at a same node.
	Find a shortest tour.	Find a shortest tour.
Differences	Cover each node *exactly* once.	Cover each arc *at least* once.
	It is NP (non-deterministic polynomial-time hard) no polynomial-time algorithm is known for solving TSP.	It is in P which is set of all problems that can be solved in polynomial time.

Table 5: Comparison of TSP and CPP.

Conclusion/Future Work

Future work for my project includes the following:

- Model TSP with Linear and Integer Programming
- Further study Branch and Bound as a solution method to solving LP's and IP's.
- Implement Branch and Bound for TSP in MATLAB.
- Investigate other solution methods of TSP and CPP.
- Analyze various applications of TSP and CPP.

Working on this project was a great experience. Throughout this project I have learned and applied different solution methods and Heuristics to solve TSP. Also by implementing and running code for Brute-force and Branch and Bound, clarified which method works fast in MATLAB. Analyzing Chinese Postman Problem also provided a better understanding of how it is similar and different from Traveling Salesman Problem.

When I first started this project I was really worried about how I will get through it, since I had knowledge about it. I have learned so much academically just by doing research on TSP that it makes me feel proud. This project has provided me which a great research experience and knowledgeable information that will help me in my future studies.

Appendix A

MATLAB Code 1

```
%Written by Anne Maredia
%Description: This program will plot nodes vs. tours and nodes vs. arcs when n the number of
   nodes is from 1 to 20.
%
%
%
X=[];
Y=[];
format short e
for n=1:20
   arcs=(n*(n-1))/2;
   tours=(factorial(n-1))/2;
   X(1,n)=arcs
   Y(1,n)=tours
end
subplot(1,2,1)
plot(X,'r*-')
xlabel 'No. of nodes'
ylabel 'No. of arcs'
title 'Nodes vs. Arcs'
subplot(1,2,2)
semilogy(Y,'->')
xlabel 'No. of nodes'
ylabel 'No. of tours'
axis([0 20 0 10^20])
title 'Nodes vs. Tours'
```

Appendix B

MATLAB Code 2

```
%Written by Anne Maredia
%Description: A MATLAB function bf4(A), takes in the matrix A and outputs all three tours with the distance and elapsed time in seconds and also tells us which tour is the best tour. This program works for any 4 by 4 matrix.
%
%
%
%A=[0 3 4 7; 3 0 5 1; 4 5 0 10; 7 1 10 0];
function bf4(A)
tic
tour_length(1)=A(1,2)+A(2,3)+A(3,4)+A(4,1)
tour1='1,2,3,4,1'

tour_length(2)=A(1,2)+A(2,4)+A(4,3)+A(3,1)
tour2='1,2,4,3,1'

tour_length(3)=A(1,3)+A(3,2)+A(2,4)+A(4,1)
tour3='1,3,2,4,1'

tour=[tour1;tour2;tour3]
tour=cellstr(tour)
Opt_Tour_Length=min(tour_length)
Opt_Tour=(find(tour_length == Opt_Tour_Length))
Shortest_Tour=tour(Opt_Tour)
elapsed_time = toc
```

Appendix C

MATLAB Code 3

```
%Written by Anne Maredia
%Description: A MATLAB function bf5(B), takes in the matrix B and outputs all twelve tours with the distance and elapsed time in seconds and also tells us which tour is the best tour. This program works for any 5 by 5 matrix.
%
%
%
%B=[0 7 36 11 9; 7 0 13 19 23; 36 13 0 8 42; 11 19 8 0 10; 9 23 42 10 0];
function bf4(B)
tic
tour_length(1)=B(1,2)+B(2,3)+B(3,4)+B(4,5)+B(5,1)
tour1='1,2,3,4,5,1'

tour_length(2)=B(1,2)+B(2,4)+B(4,3)+B(3,5)+B(5,1)
tour2='1,2,4,3,5,1'

tour_length(3)=B(1,2)+B(2,5)+B(5,4)+B(4,3)+B(3,1)
tour3='1,2,5,4,3,1'

tour_length(4)=B(1,3)+B(3,2)+B(2,5)+B(5,4)+B(4,1)
tour4='1,3,2,5,4,1'

tour_length(5)=B(1,3)+B(3,5)+B(5,4)+B(4,2)+B(2,1)
tour5='1,3,5,4,2,1'

tour_length(6)=B(1,3)+B(3,4)+B(4,2)+B(2,5)+B(5,1)
tour6='1,3,4,2,5,1'

tour_length(7)=B(1,2)+B(2,5)+B(5,3)+B(3,4)+B(4,1)
tour7='1,2,5,3,4,1'

tour_length(8)=B(1,5)+B(5,4)+B(4,2)+B(2,3)+B(3,1)
tour8='1,5,4,2,3,1'

tour_length(9)=B(1,4)+B(4,2)+B(2,5)+B(5,3)+B(3,1)
tour9='1,4,2,5,3,1'
```

```
tour_length(10)=B(1,4)+B(4,3)+B(3,2)+B(2,5)+B(5,1)
tour10='1,4,3,2,5,1'

tour_length(11)=B(1,5)+B(5,3)+B(3,2)+B(2,4)+B(4,1)
tour11='1,5,3,2,4,1'

tour_length(12)=B(1,4)+B(4,5)+B(5,3)+B(3,2)+B(2,1)
tour12='1,4,5,3,2,1'

tour=[tour1;tour2;tour3;tour4;tour5;tour6;tour7;tour8;tour9;tour10;tour11;tour12]
tour=cellstr(tour)
Opt_Tour_Length=min(tour_length)
Opt_Tour=(find(tour_length == Opt_Tour_Length))
Shortest_Tour=tour(Opt_Tour)
elapsed_time = toc
```

References

[1]Adamchik, Victor. "Graph Theory". 2 April 2010 <http://www.cs.cmu.edu/~adamchik/21-127/lectures/graphs_4_print.pdf>.

[2]Bertsekas, Dimitri. Network Optimization: Continuous and Discrete Models. Belmont: Dimitri P. Bertsekas, 1998.

[3]Boffey, T. Graph Theory in Operations Research. 1st ed. London and Basingstoke: The Macmillan Press Ltd, 1982. 148-185. 25 November 2009.

[4] Bronson, Richard, and Naadimuthu Govindasami. Operations Research. New York: McGraw-Hill, 1982

[5]Calgor, H. "The Brute Force Algorithm". 24 January 2010 <http://www.ctl.ua.edu/math103/hamilton/HCalgor.htm#THE BRUTE FORCE ALGORITHM>.

[6] Cook, William. "History of the TSP." *The Traveling Salesman Problem*. Oct 2009. Georgia Tech, 22 Jan 2010. <http://www.tsp.gatech.edu/index.html>.

[7]Cook, William J., William H. Cunningham, William R. Pulleyblank, and Alexander Schrijver. Combinatorial Optimization. New York [etc.: John Wiley & Sons, 1998.

[8]Davis, Timothy, and Kermit Sigmon. MATLAB Primer 7th edition. Boca Raton: Chapman & Hall/CRC, 2005.

[9]Garey, Michael, and David Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness . New Jersey: Bell Laboratories Murray Hill, 1979.

[10]Motozawa, Yusuke. Analysis of Linear, Integer, and Binary Programming and their Applications. University of Houston-Downtown Senior Project Fall 2009.

[11]Redl Timothy, MATLAB code

[12]Robling, Guido. "Branch and Bound". 27 April 2010 <http://www.animal.ahrgr.de/en/ Animation 9.html>.

[13]Wilson, Robin. Introduction to graph theory. 4th ed. England: Addison-Wesley, 1996. 166. 19 January 2010.