American University of Armenia

Zaven and Sonia Akian College of Science and Engineering

Optimization Project

Travelling Salesman Problem

Team: Meri Mirijanyan, Larisa Malkhasyan, Elen Tumasyan

Spring, 2018

Abstract

This paper aims to discuss algorithms about Travelling Salesman Problem. Basic information about the algorithms would be explained with simple examples and then those algorithms would be applied on the Armenian cities. We used mainly 7-10 cities of Armenia. The main focus of the paper would be the Recursive and Dynamic Programming algorithms.

Keywords: Travelling Salesman Problem, Hamiltonian circuit, TSP, Dynamic Programming, Brute Force, Recursive Approach

Contents

Chapter 1

Introduction

Travelling salesman problem is a realistic one, which each of us can face during our life. The idea behind the problem is the following: a salesman has to visit every single city in the region just once and additionally he needs to end the trip in a city where he started it. Thus, the problem salesman faces is to find the most efficient route. Naturally, there are many ways to visit all those cities, but he needs to choose the one, which has the minimum cost (distance).

1.1 The History

Now let us look back in the history and make a guess that the ancestor of this problem is "The seven bridges of Konigsberg". Moreover, this mathematical problem was basis for Euler‘s invention of field called "the geometry of position"which we now call graph theory. In 1800s Irish mathematician Sir William Rowan Hamilton and British mathematician Thomas Penyngton Kirkman introduced a problem similar to TSP. The picture which you can see is a photo of Hamilton‘s Icosian Game which requires players to complete tours through 20 points using only the specified connections.

Moreover, Hamilton tried to solve a problem, which is now called Hamilton path. Before continuing, let us first learn that Hamilton path is alike Euler path and for understanding, fully one may take a look at both definitions.

- A Eulerian path through a graph is any path where every single edge is visited exactly once.
- A Eulerian circuit is a Eulerian path that is also a cycle.
- A Hamilton path through a graph is one that passes through each vertex exactly once.
- A Hamilton circuit is just a Hamilton path but it is additionally a cycle.

In other words, travelling salesman problem is a graph problem where the cities are vertices and the roads between them are the edges. Thus, the problem itself is to find a Hamilton circuit in the graph taking into consideration the weighted edges to find the shortest circuit. One may ask why Hamilton path works for TSP? The reason is that job of the salesman is travelling to each city exactly once, and end in a place where it started which is a graph problem. The general form of TSP has been first studied by mathematicians starting in the 1930s by Karl Menger in Vienna and Harvard. Later the problem was promoted by Hassler Whitney and Merrill Flood at Princton. TSP is an NP-complete problem (from non-polynomial) for which no known efficient (for example polynomial time) algorithm exists.

1.2 Applications

Traveling salesman problem is well known for its variety of uses in different real life problems. The traveling salesman problem can be extended or modified in several ways. The solution of TSP has several applications, such as planning, scheduling, logistics and packing. Here are some examples of real life applications of TSP.

1.2.1 Drilling of printed circuit boards

One of the direct applications of the TSP is in the drilling problem of printed circuit boards (PCBs). To connect a conductor on one layer with a conductor on another layer holes have to be drilled through the board. As the holes might be of different sizes the head of the machine has to move to a tool box and change the drilling equipment to drill two holes of different diameters consecutively. To avoid wasting time on constantly changing the head of the drill after doing one hole is it more convenient to drill all the holes that have the same size and then change the head and do that for the remaining holes. Therefore, this drilling problem can be viewed as a series of TSPs, one for each hole diameter, where the 'cities' are the set of all holes that can be drilled with one and the same drill. The 'distance' between two cities is given by the time it takes to move the drilling head from one position to the other. The aim is to minimize the travel time for the machine head.

1.2.2 Railways

TSP can be used in organizing railway routs and finding the optimized and fastest ways that would guarantee the effectiveness of railways. Railway can analyze the best route available for a long distance train with many stops. They can come up with the best route which has more travelers and takes minimal time and distance between first and last destination.

1.2.3 Genome

Researchers at the National Institute of Health have used TSP to construct radiation hybrid maps as part of their ongoing work in genome sequencing. The TSP provides a way to integrate local maps into a single radiation hybrid map for a genome. In this case, the cities are the local maps and the cost of travel is a measure of the likelihood that one local map immediately follows another. Moreover, this

application of the TSP has been adopted by a group in France developing a map of the mouse genome.

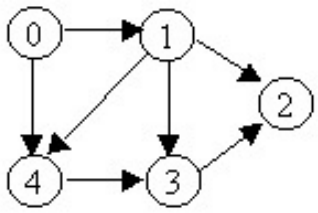
Chapter 2

Brute Force And Recursive Approach

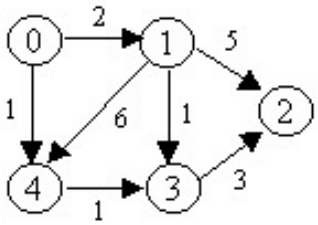
2.1 Theoretical Background: Adjacency matrix

Adjacency matrix is a square matrix used to represent a finite graph. The elements of the matrix indicate whether pairs of vertices are adjacent on not in the graph. Moreover, two vertices are said to be adjacent if there is an edge connecting them.

Thus, for simplicity we will assume that our graph is an adjacency matrix. What is more, in case the edge does not exist we can simply say that the value is plus infinity.



$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$



$$A = \begin{bmatrix} \infty & 2 & \infty & \infty & 1 \\ \infty & \infty & 5 & 1 & 6 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 3 & \infty & \infty \\ \infty & \infty & \infty & 1 & \infty \end{bmatrix}$$

2.2 Brute Force: Method of solving

When one thinks of solving TSP, the first method that might come to mind is a brute-force method. The algorithm is simple. The brute-force method is to simply generate all possible tours and compute their distances. In other words, we want

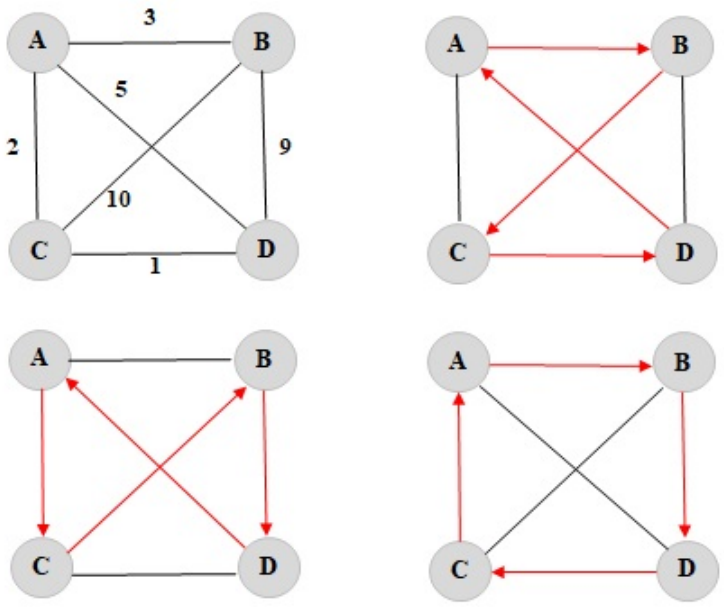
to find every single permutation or possible path that our salesman could take. The shortest tour is thus the optimal tour. Thus, basically to solve TSP using Brute-force method we can use the following steps:

1. Calculate the total number of tours.
2. Draw and list all the possible tours.
3. Calculate the distance of each tour.
4. Choose the shortest tour, this is the optimal solution.

The time complexity of Brute Force algorithm is O(n!)[1].

2.3 Discussion of examples

Let us consider some easy example to fully understand. Imagine the salesman has to visit 4 cities. Let us call them as A, B, C and D. He has to start from the city A, go through the all other cities by crossing them only once and come back to the A city. We weight every path from one city to the other and then find the minimal weighted path that satisfies our problem. As you can see in the picture:



- A->C->B->D->A = 26

[1] For more information on Big O notation see: Big O notation

- A->D->B->C->A = 26 (reverse traversal so it is the same)
- A->B->C->D->A = 15
- A->D->C->B->A = 15 (reverse traversal so it is the same)
- A->B->D->C->A = 19
- A->C->D->B->A = 19 (reverse traversal so it is the same)

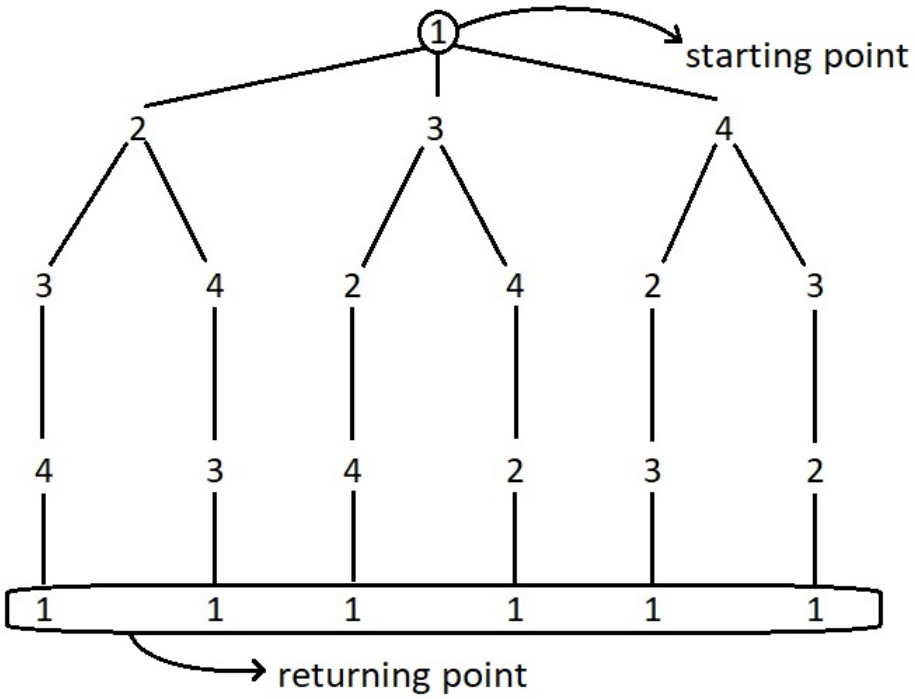
From this we see that the best distance is A>B>C>D>A or A>D>C>B>A = 15. Moreover, it turns out that there are exactly n! different permutations of the paths. Since we only care about permutations that start with the fixed city and those paths that start with different cities we do not consider, to solve an n-city TSP instance with brute force requires that we look at exactly (n-1)! different permutations.

Now that we know the main idea behind Brute Force approach let us consider an example where we will examine it in more detail. This example solves for the cases where the distances A->B=x, B->A=y and x!=y may not be necessarily equal. Representing the distances as matrix entries means that matrix is not necessarily symmetric. Note that our project's problem discusses symmetric matrices(distances) as the path between cities is the same, thus the following example includes our project's problem as well. Also, if there is no path from some p->q, then we can put a very big number instead of it so it will be not considered as an optimal solution anyway.

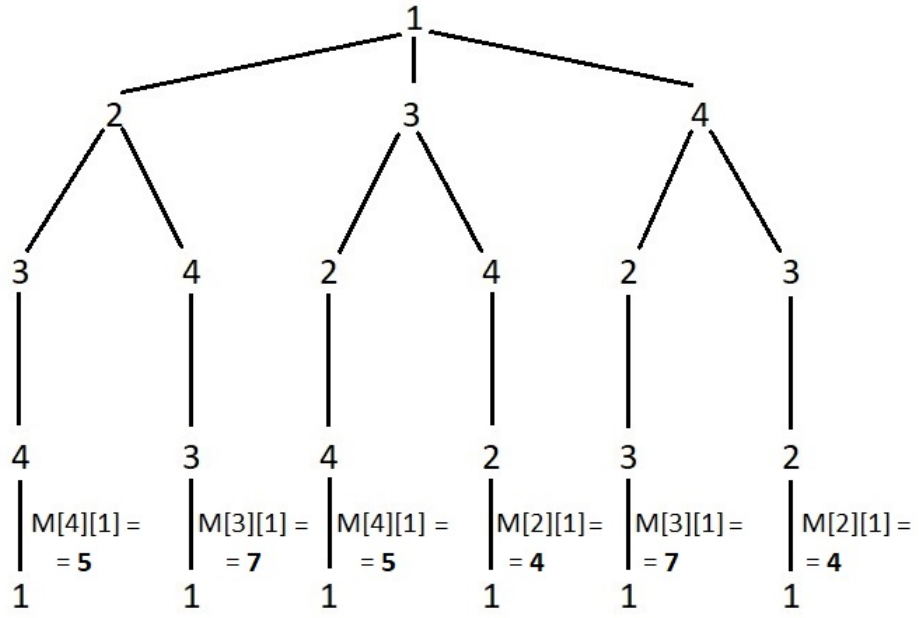
Consider the following example given in form of matrix = M: cost to go from 1 to 3 for instance is equal to M[1][3].

0	9	15	10
4	0	8	11
7	12	0	10
5	6	7	0

Assume we start from 1 and we should traverse all the vertices once and return to vertex 1. The following graph shows all the possible traversals.

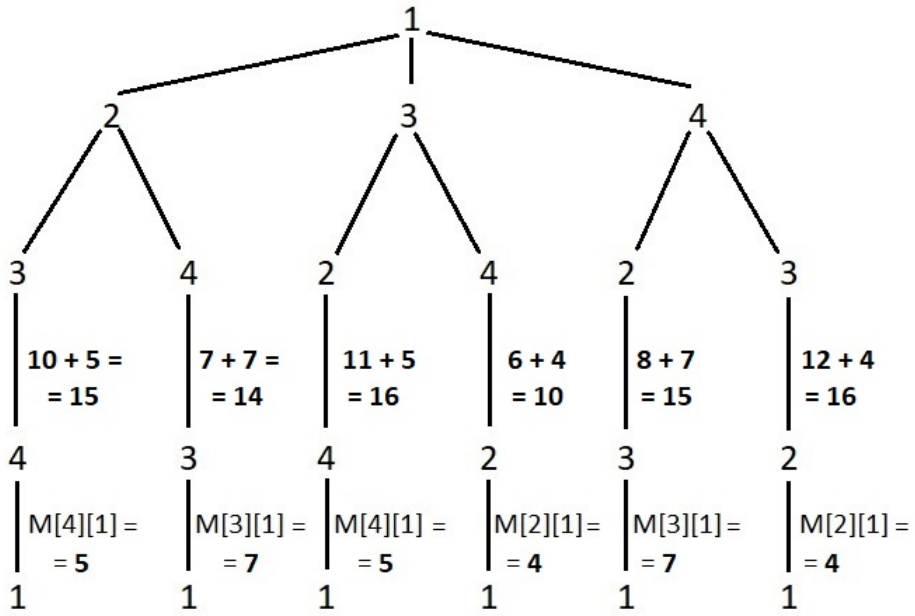


Suppose while attempting to find an optimal solution, we think weather to go from 1 to 2, 1 to 3 or 1 to 4. For that we would need to know the costs from the bottom level (costs from 2 to 3, 2 to 4, 3 to 2, ...), so let us start from the most bottom. Suppose the traveller went to 1->2->3->4 and now he is on the 4th vertex. It remains for him to go from 4->1 i.e. M[4][1] to finish his way. In the same way let us compute all the last traversals for all the possible paths.

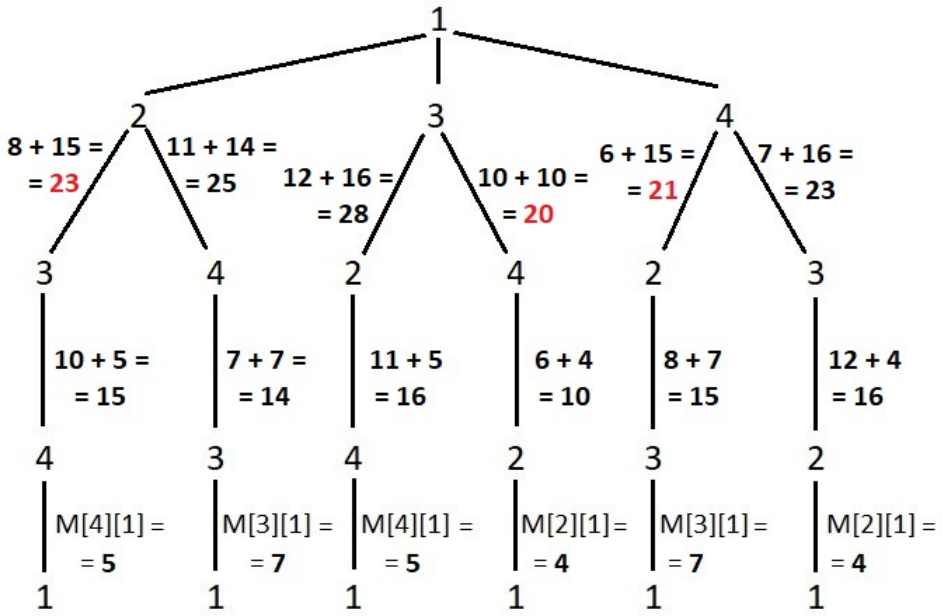


Now we go one level up. Suppose the traveller went from 1->2->3 and now we can calculate the cost of going from 3->4->1 which will be M[3][4]+M[4][1]. In the

same way let us compute all the remaining possible traversals.

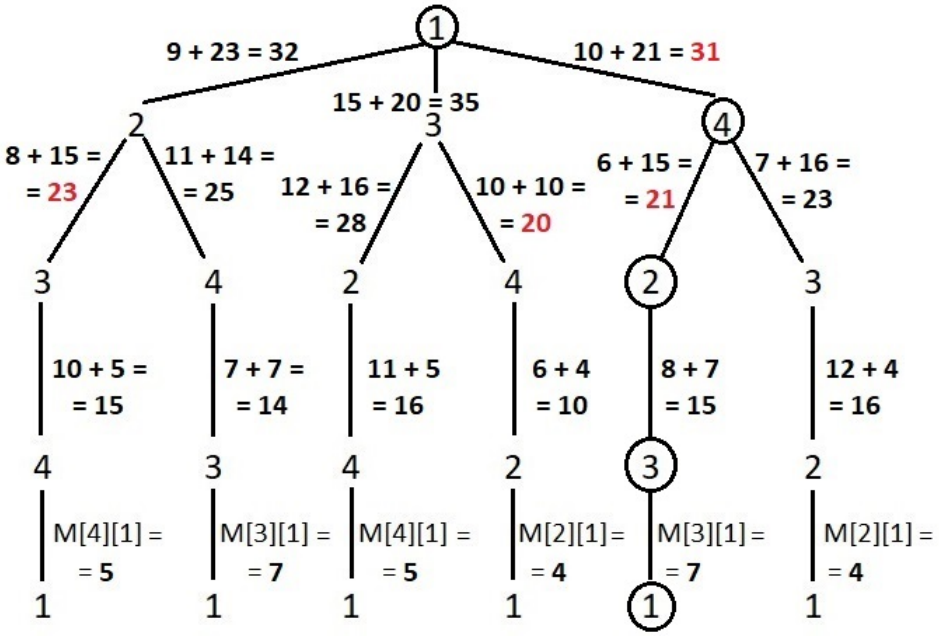


Now again we go one level up. Suppose traveller went from 1->2, now we want to compute the costs of 2->3->4->1 and 2->4->3->1, for sake of taking the minimum of them, which will be M[2][3] + M[3][4] + M[4][1] and M[2][4] + M[4][3] + M[3][1] correspondingly. Let us form for all possible paths.



We can see the minimum ones are written in red. This means that when the traveller reaches to those vertices that we have already looked at, he must choose the path with minimum cost. Now we go one level up and compute the remaining possible traversing costs. If the traveller goes from 1->2 then he must surely go

from 2->3->4->1 as it is the least cost one(23 < 25). Let us continue examining our graph.



As we can see going from 1->4->2->3->1 with cost 31 is the optimal solution because all other possible paths will give a traversal with bigger cost. Discussion of this example will help us understand the Recursive Approach easily as we continue into our paper.

2.4 Recursive Approach

In the recursive approach we will present our problem as a solution that we obtained by solving the smaller version of that problem previously. The formula we examine looks as follows.

$$F(i, S) = min_{j \in S}\{C_{ij} + F(j, S/\{j\})\}$$

where

- i - is the point where we currently are. From the beginning i is the starting vertex itself.
- S - is the set of vertices that we still have to traverse.
- Cij - is the cost of travelling from i vertex to j vertex.

Each time while trying to make a choice we go from i to j and then we solve the same problem with the same set, starting from j, just without having j vertex in

our set S. The practice of taking the minimum of all calculated paths is intuitive and gives us the optimal solution. Let us continue to the Dynamic Programming approach where we will encounter this form of function as well in a bottom up manner.

Chapter 3

Dynamic Programming

3.1 The Main Idea

Sometimes problems seem us to be hard or even unsolvable as we try to find magically a solution to the whole problem. However, in most cases especially in computer science problems, we need to divide the problem into several small pieces and after find solution for each one. Finally, giving solution to each part will lead us finding solution to the initial problem. The method described is called dynamic programming. The method of dynamic programming, which is sometimes referred to as dynamic optimization is an approach to solving complex problems by breaking down into smaller parts and storing the results to these sub-problems so that they only need to be computed once. Moreover, it is essential to understand that solving travelling salesman problem in this way will make our solution more efficient. In our case the given problem is a complete graph with weighted edges(as an adjacency matrix) and we need to find Hamiltonian cycle of minimum cost.

The main idea will be to compute the optimal solution for all the subpaths of length N while using information from the already known optimal partial tours of length N-1.

3.2 Example of how DP method works

Let us start from the easiest case just to get an idea how DP works. Imagine you are asked to compute 4+4+4. What will you do ? Answer is simple just add them up and get 12. Next you are asked to compute 4+4+4+4. Would we add them again in the same way as in the previous case? The answer is no as we already know that 4+4+4 is 12, as we solved it previously and remember our

answer. Thus, we get 12+4, which is 16. This simple example, where we used our mathematics knowledge helped us to get the idea of DP algorithm. Thus, when it meets the same sub-problem again that is we have an overlapping sub-problem it remembers that already calculated answer exists and there is no need to do an additional work. The method how DP algorithm understands that problem is already solved is called memoization.

3.3 The Minimization Process

Consider the following TSP:

minimize z = $\sum_{j=1}^{n} c_{ij}x_{ij}$ i=1, ..., n, j=1, ..., n

Subject to the constraints:

- $\sum_{i=1}^{n} x_{ij}$ = 1 j=1, ..., n
- $\sum_{j=1}^{n} x_{ij}$ = 1 i=1, ..., n

where

x_{ij} is 1 if the salesman travels from city i to city j, 0 otherwise.

c_{ij} is the distance of going from city i to city j.

z = min, the total cost of the matrix.

3.4 DP Step By Step

Step 1: Consider the given travelling salesman problem in which he wants to find that route which has the shortest distance for the set of cities { 1, 2, 3, 4 }. We start from city 1.

Step 2: Consider set of 0 elements, such that

$$g(2, \emptyset) = c_{12}$$
$$g(3, \emptyset) = c_{13}$$
$$g(4, \emptyset) = c_{14}$$

Here we assumed that we start our route from vertex 1 and we want to find what is minimum cost to reach 2 from start vertex 1 going via empty set. (same situation with vertex 4 and 3).

Step 3: After completion of Step 2, consider sets of 1 elements, such that

Set {2}, that is we approach to 3 or 4 from starting vertex 1 through the set {2} :

$$g(3,\{2\}) = c_{23} + g(2,\emptyset) = c_{23} + c_{12}$$
$$g(4,\{2\}) = c_{24} + g(2,\emptyset) = c_{24} + c_{12}$$

Set {3}, that is we approach to 2 or 4 from starting vertex 1 through the set {3} :

$$g(2,\{3\}) = c_{32} + g(3,\emptyset) = c_{32} + c_{13}$$
$$g(4,\{3\}) = c_{34} + g(3,\emptyset) = c_{34} + c_{13}$$

Set {4}, that is we approach to 2 or 3 from starting vertex 1 through the set {4} :

$$g(2,\{4\}) = c_{42} + g(4,\emptyset) = c_{42} + c_{14}$$
$$g(3,\{4\}) = c_{43} + g(4,\emptyset) = c_{43} + c_{14}$$

Step 4: After completion of step 3, consider sets of 2 elements(in the same way as previously), such that

Set {2,3}: $g(4,\{2,3\}) = min\{c_{24} + g(2,\{3\}), c_{34} + g(3,\{2\})\}$

Set {2,4}: $g(3,\{2,4\}) = min\{c_{23} + g(2,\{4\}), c_{43} + g(4,\{2\})\}$

Set {3,4}: $g(2,\{3,4\}) = min\{c_{32} + g(3,\{4\}), c_{42} + g(4,\{3\})\}$

Step 5: After completion of step 4.

$$z = g(1,\{2,3,4\}) = min\{c_{21} + g(2,\{3,4\}), c_{31} + g(3,\{2,4\}), c_{41} + g(4,\{2,3\})$$

And the optimal TSP z is found.

The analog of this algorithm would work for any number of cities.

3.5 Bitmask DP

There is a version of DP algorithm where we use bits 0 and 1 as a key to our solution. Suppose we have 4 cities to visit, then at first we have 0000, when we visit a city, the bit vector becomes either 0001 or 0010 or 0100 or 1000, meaning that 1 indicates we have visited that particular city. Suppose we are on some 1001 state, this means we have visited the first and the fourth cities, and now we are looking for the next city to visit. If our next city is the second one, that is 0010, then as the logical AND, i.e. 1001AND0010 = 0000(zero), this means we can visit this city, and our next state becomes their logical OR, i.e. 1001OR0010=1011. But if, for example, our next assumed city was the first one(0001) again (which is not valid as we have already visited first city) then 1001AND0001=0001 which is not zero, so we should not go towards that direction.

To sum up, if the bitmask of the selected city on our state is zero, then we visit that city, if no then we move to another city option. Using this technique together with the DP approach we can write a code to solve TSP problem. It is also possible to use Bitmasking with Recursive Approach, which we did as well.

Chapter 4

Results Comparison and Discussion

For the TSP problem we have chosen two versions of coding solutions to implement on Matlab. First is recursion and the second one is Dynamic Programming. For both of them we have used bitmasking technique. The recursive algorithm works on O(n!) time and the DP works on O($n^2 2^n$) time. Here is the chart of both running times on various values of number of cities.

n	n!	$n^2 2^n$
1	1	2
2	2	16
3	6	72
4	24	256
5	120	800
6	720	2304
...	...	...
15	1307674368000	7372800
16	20922789888000	16777216
17	355687428096000	37879808

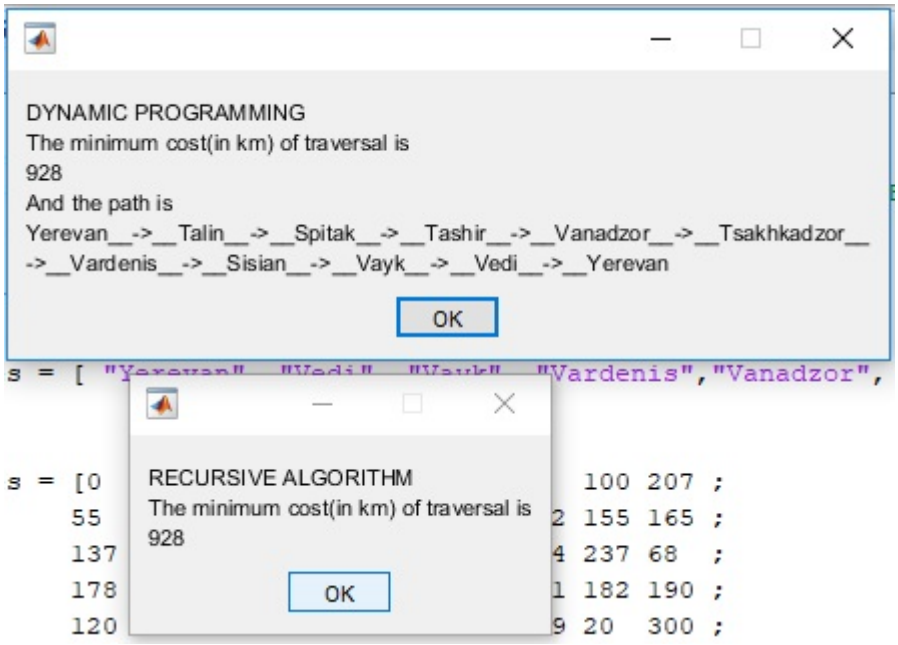
We can see that the difference between running times is huge for n = 17.

So we took 10 cities in Armenia and calculated their relative distances in km. Without loss of generality we assume we start from the first city in the matrix, as we can always change the positions of the cities in our adjacency matrix.

	A	B	C	D	E	F	G	H	I	J	K
1	in (km)	Yerevan	Vedi	Vayk	Vardenis	Vanadzor	Tsakhkadzor	Tashir	Talin	Spitak	Sisian
2	Yerevan	0	55	137	178	120	56	159	67	100	207
3	Vedi	55	0	95	233	175	111	214	122	155	165
4	Vayk	137	95	0	120	229	193	297	204	237	68
5	Vardenis	178	233	120	0	162	138	213	231	182	190
6	Vanadzor	120	175	229	162	0	106	51	119	20	300
7	Tsakhkadzor	56	111	193	138	106	0	157	111	126	263
8	Tashir	159	214	297	213	51	157	0	158	59	350
9	Talin	67	122	204	231	119	111	158	0	99	274
10	Spitak	100	155	237	182	20	126	59	99	0	320
11	Sisian	207	165	68	190	300	263	350	274	320	0
12											
13											
14											
15											
16											

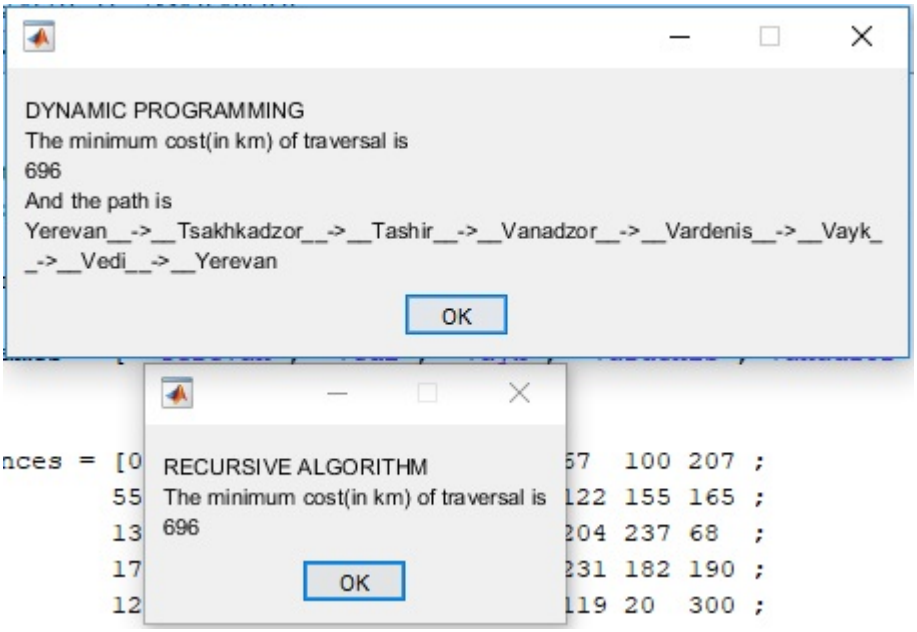
We wrote our code so that the tester can change the values of number of cities and see results for all cases. Besides that we put one of our examples we discussed in this paper as a comment in the Matlab, so you can just uncomment it and run.

Here is our Matlab result for all 10 cities.



Our DP Matlab solution gives the minimum cost and also the path, the recursive solution only gives the cost. We suggest not to test recursion for number of cities = 10 as it takes nearly 20 minutes to give the answer, whereas for cases equal to 6 or 7 recursion works fine.

Here is our Matlab result for first 7 cities.


DYNAMIC PROGRAMMING
The minimum cost(in km) of traversal is
696
And the path is
Yerevan__->__Tsakhkadzor__->__Tashir__->__Vanadzor__->__Vardenis__->__Vayk_
_->__Vedi__->__Yerevan
OK
RECURSIVE ALGORITHM
The minimum cost(in km) of traversal is
696
OK

Chapter 5

Conclusion

As a result of our project examination we can conclude that the TSP problem has various algorithms that work for different cases. In our project we wrote the Matlab codes for two algorithms: Recursive and DP. The recursive algorithm works well for number of cities less than 7, if the number of cities increases, the recursive algorithm works slower and slower. The Dynamic Programming algorithm gave us a better result. It works well for number of cities up to 12-13 (slower as gets bigger).

Bibliography

[1] D.L. Applegate, R.E. Bixby, V. ChvÃątal, and W.J. Cook. The Traveling Salesman Problem: A Computational Study. Princeton Series in Applied Mathematics. Princeton University Press, 2011.

[2] W. Cook. In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation. Princeton University Press, 2012.

[3] Mathworks. Retrieved from MathWorks.com

[4] Routing. Retrieved from Routing

[5] Heuristic Approaches to Solve Traveling Salesman Problem. Retrieved from TCP

[6] Naive and Dynamic Programming. Retrieved from GeeksForGeeks

[7] Approximate MST TSP. Retrieved from Approximate MST TCP

[8] Backtracking | Hamiltonian cycle. Retrieved from Hamiltonian cycle

[9] The Advantage of Intelligent Algorithms for TSP. Retrieved from Intechopen

[10] Tackling the travelling salesman problem. Retrieved from psychicorigami

[11] TSP and Applications. Retrieved from TSP and its Applications

[12] Transportation and logistics applications. Retrieved from math.uwaterloo.ca

[13] Genome. Retrieved from TSP

[14] Scan Chain Optimization. Retrieved from TSP