SOLVING TRAVELING SALESMAN PROBLEM BY DYNAMIC PROGRAMMING APPROACH IN JAVA PROGRAMMING

CE 6001 OPERATIONS MANAGEMENT AND INFRASTRUCTURE SYSTEM

ADITYA NUGROHO

HT083276E

DEPARTMENT OF CIVIL ENGINEERING
NATIONAL UNIVERSITY OF SINGAPORE

2010

1.0 INTRODUCTION

Traveling Salesman Problem (TSP) is one of the important method in the application on transportation science. TSP can be illustrated as a salesman who must travel through all the cities designated by the shortest distances, where each city may only be traversed once. Solution of the TSP is the path traversed by these salesmen. Surely the best or optimal solution of this problem is the path with the shortest distance or can be called with a minimum of travel routes.

A very simple depiction of the term TSP is, a salesman who must travel *n* cities with the rules: he must visit each city only once. He has to minimize the total travel distance and in the end he had to return to his origin city. Thus, what had he done called a tour. In order to ease the problem, mapping *n* the city will be illustrated with a graph, where the number of vertices and edges are limited (a vertex will represent a city and an edge will represent the distance between the two cities which are connected). Handling TSP problem is equivalent to find the shortest Hamilton circuit.

Dynamic Programming is a method of solving problems by breaking the solution into a set of steps or stages so that the solution of the problem can be viewed from a series of interrelated decisions. The inventor and the person responsible for the popularity of dynamic programming is Richard Bellman.

In dynamic programming, a series of optimal decisions are made by using the principle of optimality. The principle of optimality: if the optimal total solution, then the solution to the k th stage is also optimal. With the principle of optimality is guaranteed that at some stage of decision making is the right decision for the later stages. The essence of dynamic programming is to remove a small part of a problem at every step, and then solve the smaller problems and use the results of the settlement to remedy the solution is added back to the issue in the next step.

1.1 Objective

- How does the dynamic programming approach in solving the TSP in determining the most optimal path with minimum total distance or time by applied forward recursion.
- Assessing the TSP as a graph Hamiltonian by using dynamic programming to obtain the most minimum amount of weight and applying it into a java language program.

1.2 Method

- Describing examples of TSP in graph form.
- Applying dynamic programming for solving TSP.
- Apply dynamic programming on the TSP into a language programming.

In this paper TSP which would be covered only an asymmetric TSP, where the asymmetric TSP is the distance from city A to city B is not equal to the distance from city B to city A (asymmetry occurs because of a one-way)

2.0 THEORETICAL REVIEW

2.1 Hamilton graph

Hamilton path is a path through every vertex in the graph exactly one times. If the path back to the vertex of origin to form a closed path (circuit), then the closed trajectory is called the Hamilton circuit. Therefore Hamilton circuit is a circuit through each vertex in the graph exactly once. Graph that has Hamilton called the Hamilton circuit graph, whereas a trajectory Hamilton called semi-Hamiltonian graph. **Theorem: any complete graph is a Hamilton graph**

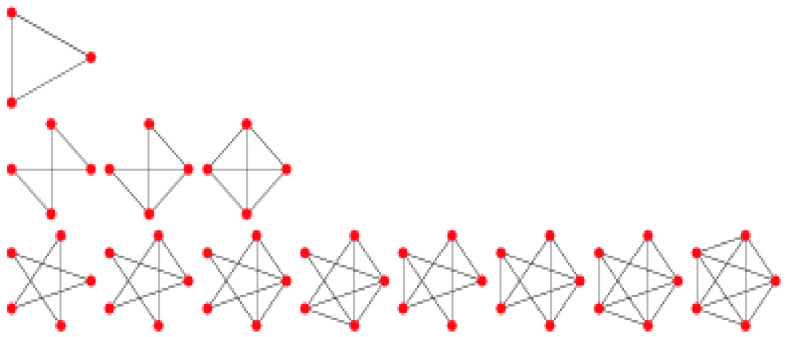


Figure 1 Hamilton graph

2.2 Graph representation in computer program

To illustrate a graph in a computer programming language, we need to translate a graph into another form, which known by the computer, it is cause the computer program could not recognize the image graph in the form of regular graph. The matrix could be used to represent a graph, if the graph is expressed as a matrix then the necessary calculations can be done easily. Graph representation on the computer can be done in several ways, such as:

- **Adjacency matrix** is defined as follows, suppose that order of *nxn* matrix (*n* rows and *n* columns). If between two vertices connected (adjacent), then the matrix element is 1, and if it is not connected is 0. Examples of adjacency matrix:

$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Figure 2 Adjacency matrix

- **Incidency matrix** is the matrix that represents relationship between the vertex and edge. Suppose matrix with *m* rows for every vertex and *n* columns for each edge. If the vertex connect with edge, then the matrix element is 1. Conversely, if the vertex is not connected with edge then the matrix element is 0. Examples of incidence matrices are:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} \qquad \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

Figure 3 Incidency matrix

2.3 Dynamic Programming in Traveling Salesman Problem

In general, dynamic programming model can be written as follows:

$$f\left(S_n, D_n\right) = R_n + f_{n-1}^{*}\left(S_{n-1}, D_{n-1}\right) \tag{1}$$

where,

$f\left(S_n, D_n\right)$	= return on-n stage of the value input status of S_n and decision-D_n
S_n	= initial conditions
S_{n-1}	= final condition
D_n	= decisions made at each stage
D_{n-1}	= decision on the final stage
R_n	= return function
$f_{n-1}^{*}\left(S_{n-1}, D_{n-1}\right)$	= return optimal on stage *n-1* of the value input status of S_{n-1} and decision-D_{n-1}

Now let consider application of DP in TSP. Let $G = (V, E)$ is a complete directed graph with its edge $C_{ij} \geq 0$ for every *i* and *j*, where *i* and *j* are the vertex inside of *V*. Suppose $|V| = n$ and $n \geq 1$, therefore each node is numbered by *1, 2, ..., n*.

$$f\left(1, v-\{1\}\right) = \min_{2 \leq k \leq n} \left\{c_{1k} + f\left(k, V-\{1,k\}\right)\right\} \tag{2}$$

where,

$f\left(1, v-\{1\}\right)$	= length of optimal tour
V	= non-empty finite set of vertices of a graph
$v-\{1\}$	= the set of vertices of a graph which vertices minus 1 (initial)
k	= vertex that is connected to the vertex 1 (initial)
c_{1k}	= weight of vertex 1 (the beginning) to *k*

From the equation 2 we can get the recurrence relationship as follow:

$$f(i, \varnothing) = c_{ij},\ 2 \leq i \leq n \tag{3}$$

$$f(i, s) = \min\left\{c_{ij} + f\left(j, S-\{j\}\right)\right\}$$

where,

$f(i,s)$	= weight of shortest path
$S-\{j\}$	= series of *S* minus of point vertex *j*
i and j	= vertices in *V*
c_{ij}	= weight of vertex i to j

Assume trip (tour) begins and ends at vertex 1. Each tour would consist of the side (*1*,*k*) for some $k \in V-\{1\}$ and a path from vertex *k* to the vertex 1. Path from vertex *k* to vertex 1 through each vertex in *V - {1, k}* is only once. Let $f(i,s)$ is the weight of the shortest path that starts from vertex *i*, which through all vertices in *S* and ends at vertex 1. The value of $f\left(1,v-\{1\}\right)$ is the weight of the shortest tour.

From equation (2) can be obtained equation (3) if we know *f (k, V - {1, k})* for all choices of k. The value *f* can be obtained by using equation (3). We use equation (3) to obtain *f (i, S)* for |S|=1, then we can obtain *f (i, S)* to, until |S| = *n-2*.

3.0 DYNAMIC PROGRAMMING APPLICATION IN TSP

In dynamic programming, some variables and the steps necessary to determine the shortest path with minimum weight using the forward recursive method.

3.1 Initialisation steps

Initialisation

- Complete directed graph (G)
- Non-empty finite set of vertices on a graph *(V), V = {1, 2, 3, ... n}*
- The set of edges in a graph (E)
- The distance from *i to j* (the distance between cities) c_{ij} , where $c_{ij} \neq c_{ji}$
- The series of lines (S), $S \subseteq \{2,\ 3,\ \ldots,\ n\}$
- The weight of the shortest path that starts at vertex *i* that through all vertices in *S* and ends at vertex 1 *(f (i, S)),* $i \notin S$ and $S \neq \varnothing$

Therefore the steps in solving the TSP with dynamic programming is as follows:

Step 1

Determining the basis of the graph Hamiltonian that has been represented to be a adjacency matrix to the equation:

$$f(i,\varnothing) = c_{i,1}, \quad 2 \leq i \leq n$$

Step 2

Calculate *f (i, S)* for |S| = 1, then we can obtain *f (i, S)* for |S| = 2, until |S| = n-1. With the equation:

$$f(i,s) = \min_{j \in S} \left\{ c_{ij} + f(j, S - \{j\}) \right\}$$

Step 3

Having obtained the results from step 2, then calculate the equation of a recursive relationship by following equation:

$$f\left(1, v - \{1\}\right) = \min_{2 \le k \le n} \left\{ c_{1k} + f(k, V - \{1, k\}) \right\}$$

Step 4

After calculating a recursive equation in step 3, will be obtained by weighting the shortest path, then to obtain an optimal solution / length of the shortest path for a graph is to calculate *f (1, {2, 3, 4, ..., n})* which means long lines from the initial vertex (1) to vertex 1 after passing through the vertices 2, 3, 4, .., n with any order (for the minimum) then locate the forming of the minimum optimal solution which has been obtained.

3.2 Numerical examples

Following are numerical examples of the solving TSP by using dynamic programming. Given a complete directed graph, with the TSP problem for *n* = 4. The series of lines {2, 3, 4} can be seen in Figure 4 as follow:

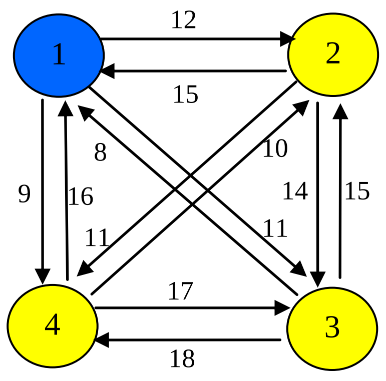


Figure 4 Graph with 4 vertices

Here distance from i to j (the distance between cities) c_{ij} , where $c_{ij} \neq c_{ji}$ have converted into matrix form:

$$c = \begin{bmatrix} 0 & 12 & 11 & 16 \\ 15 & 0 & 15 & 10 \\ 8 & 14 & 0 & 18 \\ 9 & 11 & 17 & 0 \end{bmatrix}$$

The following is solution steps:

Step 1

Calculate $f(i,\varnothing) = c_{i,1}, \quad 2 \le i \le n$

We get:

$$f(2,\varnothing) = c_{21}, = 15$$

$$f(3,\varnothing) = c_{31}, = 8$$

$$f(4,\varnothing) = c_{41}, = 9$$

Step 2

For | S | = 1, calculate $f(i,s) = \min_{j \in S}\left\{c_{ij} + f(j, S - \{j\})\right\}$

We get:

$$f(2,\{3\}) = \left\{c_{23} + f(3,\varnothing)\right\} = 15 + 8 = 23$$

$$f(3,\{2\}) = \left\{c_{32} + f(2,\varnothing)\right\} = 14 + 15 = 29$$

$$f(4,\{2\}) = \left\{c_{42} + f(2,\varnothing)\right\} = 11 + 15 = 26$$

$$f(2,\{4\}) = \left\{c_{24} + f(4,\varnothing)\right\} = 10 + 9 = 19$$

$$f(3,\{4\}) = \left\{c_{34} + f(4,\varnothing)\right\} = 18 + 9 = 27$$

$$f(4,\{3\}) = \left\{c_{43} + f(3,\varnothing)\right\} = 17 + 8 = 25$$

For | S | = 2, calculate $f(i,s) = \min_{j \in S}\left\{c_{ij} + f(j, S - \{j\})\right\}$

We get:

$$f(2,\{3,4\}) = \min\left\{c_{23} + f(3,\{4\}), c_{24} + f(4,\{3\})\right\}$$
= min {15+27,10+25}
= min {42,35} = 35

$$f(3,\{2,4\}) = \min\left\{c_{32} + f(2,\{4\}), c_{34} + f(4,\{2\})\right\}$$
= min {14+19,27+11}
= min {33,38} = 33

$$f(4,\{2,3\}) = \min\left\{c_{42} + f(2,\{4\}), c_{43} + f(3,\{2\})\right\}$$
= min {11+22,17+29}
= min {33,46} = 33

Step 3

By using the equation, $f\left(1, v - \{1\}\right) = \min_{2 \le k \le n}\left\{c_{1k} + f(k, V - \{1,k\})\right\}$

We get:

$$f(1,\{2,3,4\}) = \min\{c_{12}+f(2,\{3,4\}), c_{13}+f(3,\{2,4\}), c_{14}+f(4,\{2,3\})\}$$
$$= \min\{12+35, 11+33, 16+33\}$$
$$= \min\{47,44,49\} = 44$$

Therefore, the weight of the shortest path that starts and ends at the vertex 1 is 44.

Step 4

To know the path, first note that the value of 44 is a minimum value obtained from 11 + 33. Means the shortest path is $c_{13} + f(3,\{2,4\})$ where found that the initial part of the path is 1-3. Then find out the component of $f(3,\{2,4\})$ which resulting the minimum value is $c_{32} + f(2,\{4\})$. Therefore, it is known that the edge 3-2 is part of the shortest path. In the same way, then we trace the $f(2,\{4\})$ which is found that $c_{24} + f(4,\varnothing)$. The final step is to trace forming of $f(4,\varnothing)$ which found is c_{41} (Thus edge of 4-1 is also part of the shortest path). By assembling the edges that has been defined (edge 1-3, 3-2, 2-4, 4-1) from front to back. Means that the shortest path is 1 → 3 → 2 → 4 → 1 with the number of minimum weight 44. Following figure represent the results (bold edge is the shortest path):

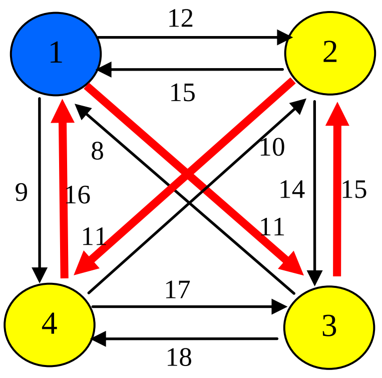


Figure 5 Graph with 4 vertices (shortest path)

4.0 APPLICATION IN COMPUTER PROGRAMMING

Implementation of dynamic programming for solving the Dynamic Programming of TSP (DPTSP) in this paper was conducted in the programming language Java. The code programming of DPTSP algorithm is solved by using binary number 2^n-1 (which is one binary consumed 4 bytes memory) therefore there is some limitations issue in the number of nodes due to memory capacity of my laptop. The interface consists of several forms that have their respective functions that appear in the order that has been programmed.

In general the process of solving TSP by dynamic programming which conducted in Java programme is represented in the flowchart as follow:

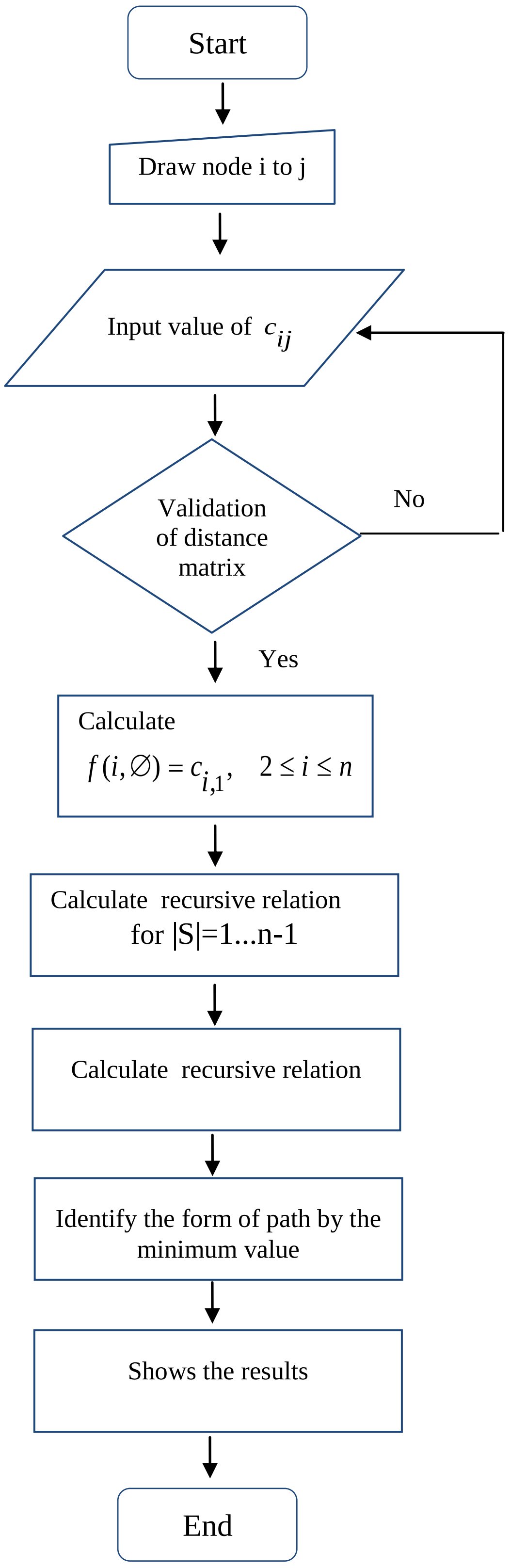
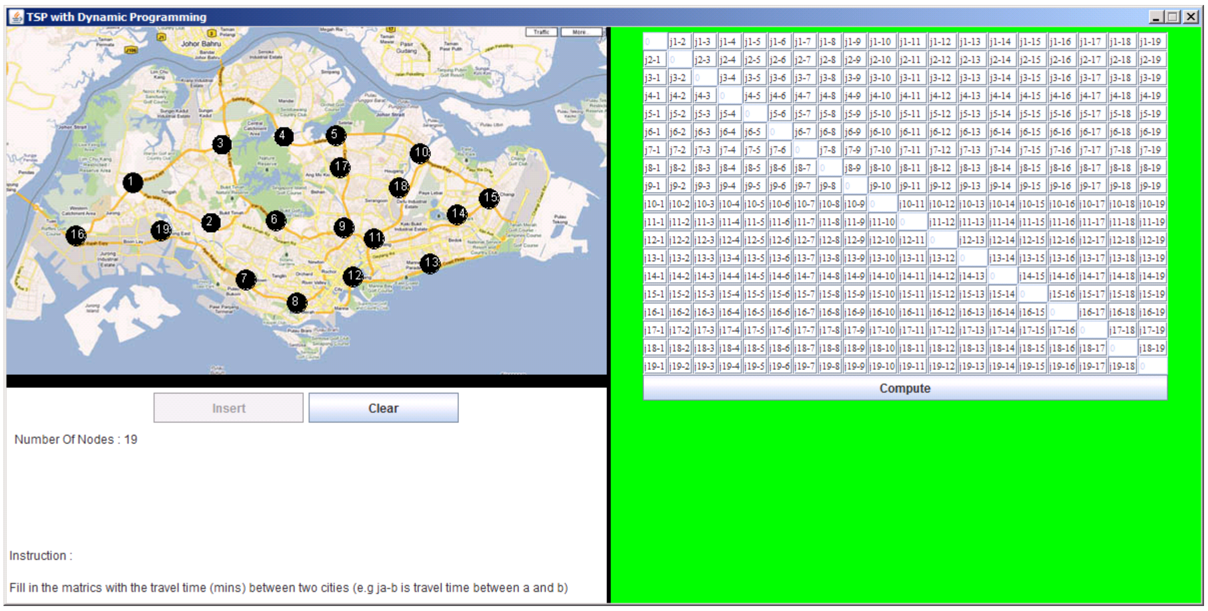

Start
Draw node i to j
Input value of c_{ij}
Validation of distance matrix
No
Yes
Calculate
$f(i,\varnothing) = c_{i,1}, \quad 2 \le i \le n$
Calculate recursive relation for |S|=1...n-1
Calculate recursive relation
Identify the form of path by the minimum value
Shows the results
End

Figure 5 Flowchart of dynammic programming in TSP by Java programme

4.1 Graphical user interface

The design of DPTSP graphical user interface is done using NetBeans IDE 6.9.1. Figure 6 is example the display of DPTSP applications in jar.file, where the user can choose any nodes represented in the map of Singapore city. Applications consist of nodes, graph, matrices and optimum number of city results including computational time (in milliseconds).



Number Of Nodes : 19

Instruction :

Fill in the matrics with the travel time (mins) between two cities (e.g ja-b is travel time between a and b)

0	j1-2	j1-3	j1-4	j1-5	j1-6	j1-7	j1-8	j1-9	j1-10	j1-11	j1-12	j1-13	j1-14	j1-15	j1-16	j1-17	j1-18	j1-19
j2-1	0	j2-3	j2-4	j2-5	j2-6	j2-7	j2-8	j2-9	j2-10	j2-11	j2-12	j2-13	j2-14	j2-15	j2-16	j2-17	j2-18	j2-19
j3-1	j3-2	0	j3-4	j3-5	j3-6	j3-7	j3-8	j3-9	j3-10	j3-11	j3-12	j3-13	j3-14	j3-15	j3-16	j3-17	j3-18	j3-19
j4-1	j4-2	j4-3	0	j4-5	j4-6	j4-7	j4-8	j4-9	j4-10	j4-11	j4-12	j4-13	j4-14	j4-15	j4-16	j4-17	j4-18	j4-19
j5-1	j5-2	j5-3	j5-4	0	j5-6	j5-7	j5-8	j5-9	j5-10	j5-11	j5-12	j5-13	j5-14	j5-15	j5-16	j5-17	j5-18	j5-19
j6-1	j6-2	j6-3	j6-4	j6-5	0	j6-7	j6-8	j6-9	j6-10	j6-11	j6-12	j6-13	j6-14	j6-15	j6-16	j6-17	j6-18	j6-19
j7-1	j7-2	j7-3	j7-4	j7-5	j7-6	0	j7-8	j7-9	j7-10	j7-11	j7-12	j7-13	j7-14	j7-15	j7-16	j7-17	j7-18	j7-19
j8-1	j8-2	j8-3	j8-4	j8-5	j8-6	j8-7	0	j8-9	j8-10	j8-11	j8-12	j8-13	j8-14	j8-15	j8-16	j8-17	j8-18	j8-19
j9-1	j9-2	j9-3	j9-4	j9-5	j9-6	j9-7	j9-8	0	j9-10	j9-11	j9-12	j9-13	j9-14	j9-15	j9-16	j9-17	j9-18	j9-19
j10-1	j10-2	j10-3	j10-4	j10-5	j10-6	j10-7	j10-8	j10-9	0	j10-11	j10-12	j10-13	j10-14	j10-15	j10-16	j10-17	j10-18	j10-19
j11-1	j11-2	j11-3	j11-4	j11-5	j11-6	j11-7	j11-8	j11-9	j11-10	0	j11-12	j11-13	j11-14	j11-15	j11-16	j11-17	j11-18	j11-19
j12-1	j12-2	j12-3	j12-4	j12-5	j12-6	j12-7	j12-8	j12-9	j12-10	j12-11	0	j12-13	j12-14	j12-15	j12-16	j12-17	j12-18	j12-19
j13-1	j13-2	j13-3	j13-4	j13-5	j13-6	j13-7	j13-8	j13-9	j13-10	j13-11	j13-12	0	j13-14	j13-15	j13-16	j13-17	j13-18	j13-19
j14-1	j14-2	j14-3	j14-4	j14-5	j14-6	j14-7	j14-8	j14-9	j14-10	j14-11	j14-12	j14-13	0	j14-15	j14-16	j14-17	j14-18	j14-19
j15-1	j15-2	j15-3	j15-4	j15-5	j15-6	j15-7	j15-8	j15-9	j15-10	j15-11	j15-12	j15-13	j15-14	0	j15-16	j15-17	j15-18	j15-19
j16-1	j16-2	j16-3	j16-4	j16-5	j16-6	j16-7	j16-8	j16-9	j16-10	j16-11	j16-12	j16-13	j16-14	j16-15	0	j16-17	j16-18	j16-19
j17-1	j17-2	j17-3	j17-4	j17-5	j17-6	j17-7	j17-8	j17-9	j17-10	j17-11	j17-12	j17-13	j17-14	j17-15	j17-16	0	j17-18	j17-19
j18-1	j18-2	j18-3	j18-4	j18-5	j18-6	j18-7	j18-8	j18-9	j18-10	j18-11	j18-12	j18-13	j18-14	j18-15	j18-16	j18-17	0	j18-19
j19-1	j19-2	j19-3	j19-4	j19-5	j19-6	j19-7	j19-8	j19-9	j19-10	j19-11	j19-12	j19-13	j19-14	j19-15	j19-16	j19-17	j19-18	0

Figure 6 GUI of the programme

As can be seen from Fig 6 a distance or travel time matrix is the form used to enter the travel time between cities into the matrix. Since the programme is developed by binary (using bitmask) therefore the value of travel time should be clearly defined in integer numbers and could not put comma or point in the matrix form. Matrix data thus inputted by the user.

4.2 Hardware requirement and coding for programming

The application of DPTSP was develop by using NetBeans IDE 6.9.1. however to achieve maximum results of the programme there is need high capacity of hardware requirement. In this term paper, the programme is develop by following hardware data:

- Pentium(R) Dual Core CPU T4200 @2.00 GHZ
- Memory (RAM) 1.00 GB
- System 32-bit Operating System Windows Vista

Once again as programming of DPTSP algorithm is solved by using binary number 2^n-1 (which is one binary consumed 4 bytes memory) and bitmaskthe issue of memory capacity is found as the rest of memory in my laptop is only 400MB out of 1.00 GB. Given example logic of binary, says I would like to call other rest cities from cities number 10 and 19, therefore the logic binary number as follow : $\frac{1}{1}\frac{1}{2}\frac{1}{3}\frac{1}{4}\frac{1}{5}\frac{1}{6}\frac{1}{7}\frac{1}{8}\frac{1}{9}\frac{0}{10}\frac{1}{11}\frac{1}{12}\frac{1}{13}\frac{1}{14}\frac{1}{15}\frac{1}{16}\frac{1}{17}\frac{1}{18}\frac{0}{19}$ in each respective binary would store 4 bytes memory. Since in this programme the bitmask is use to easily check the state of individual bits regardless of the other bits. Therefore the requirement the memory needed of programme is easily compute as follow 2^n-1*4^4=(134,217,728 bytes) or 134 MB of RAM. I use maximum 134 MB as my laptop memory should be allocate to other devices. Therefore by using binary number the code of java programming for DPTSP is given below.

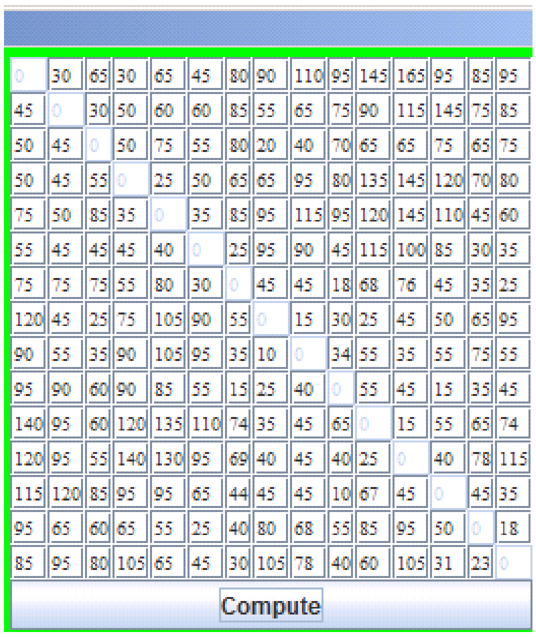
```
import java.util.ArrayList;

/**
 *
 * @author Aditya Nugroho
 */
public class TSPEngine {

    private ArrayList<Integer> outputArray = new ArrayList<Integer>();
    private int g[][], p[][], npow, N, d[][];
    public static long time;

    public TSPEngine() {
    }

    public ArrayList<Integer> computeTSP(int[][] inputArray, int n) {
        long start = System.nanoTime();

        N = n;
        npow = (int) Math.pow(2, n);
        g = new int[n][npow];
        p = new int[n][npow];
        d = inputArray;

        int i, j, k, l, m, s;

        for (i = 0; i < n; i++) {
            for (j = 0; j < npow; j++) {
                g[i][j] = -1;
                p[i][j] = -1;
            }
        }

        //initialize based on distance matrix
        for (i = 0; i < n; i++) {
            g[i][0] = inputArray[i][0];
        }
        int result = tsp(0, npow - 2);
        outputArray.add(0);
        getPath(0, npow - 2);
        outputArray.add(result);

        long end = System.nanoTime();
        time = (end - start) / 1000;
        return outputArray;
    }

    private int tsp(int start, int set) {
        int masked, mask, result = -1, temp;

        if (g[start][set] != -1) {
            return g[start][set];
        } else {
            for (int x = 0; x < N; x++) {
                mask = npow - 1 - (int) Math.pow(2, x);
                masked = set & mask;
                if (masked != set) {
                    temp = d[start][x] + tsp(x, masked);
                    if (result == -1 || result > temp) {
                        result = temp;
                        p[start][set] = x;
                    }
                }
            }
            g[start][set] = result;
            return result;
        }
    }

    private void getPath(int start, int set) {
        if (p[start][set] == -1) {
            return;
        }

        int x = p[start][set];
        int mask = npow - 1 - (int) Math.pow(2, x);
        int masked = set & mask;

        outputArray.add(x);
        getPath(x, masked);
    }
```

5.0 CASE OF PIZZA HUT DELIVERY PROBLEM BY USING DP TSP

Suppose there are only one Pizza Hut outlet in Bukit Panjang Singapore, employing about 20 staff. In the face of rapid business growth, Pizza Hut found that their need to optimize their pizza hut delivery system in order to satisfy their customers, meanwhile minimizing or reducing time and costs spent on day to day.

Suppose that the manager of the outlet need to seek minimizing time of travel for deliver the Pizza to the customers. Thus the manager has developed the business information intelligence system for Pizza Hut Delivery adopted from traffic information provided by Land Transport Authority ***traffic.smart.*** **onemotoring.com** Given this such condition in assymetric network problem, thus assume that Pizza Hut outlet has an order for deliver to 15 customers from Bukit Panjang to other rest city in Singapore, and manager has to decide to send only one riders, therefore the problem case can be given as follow:

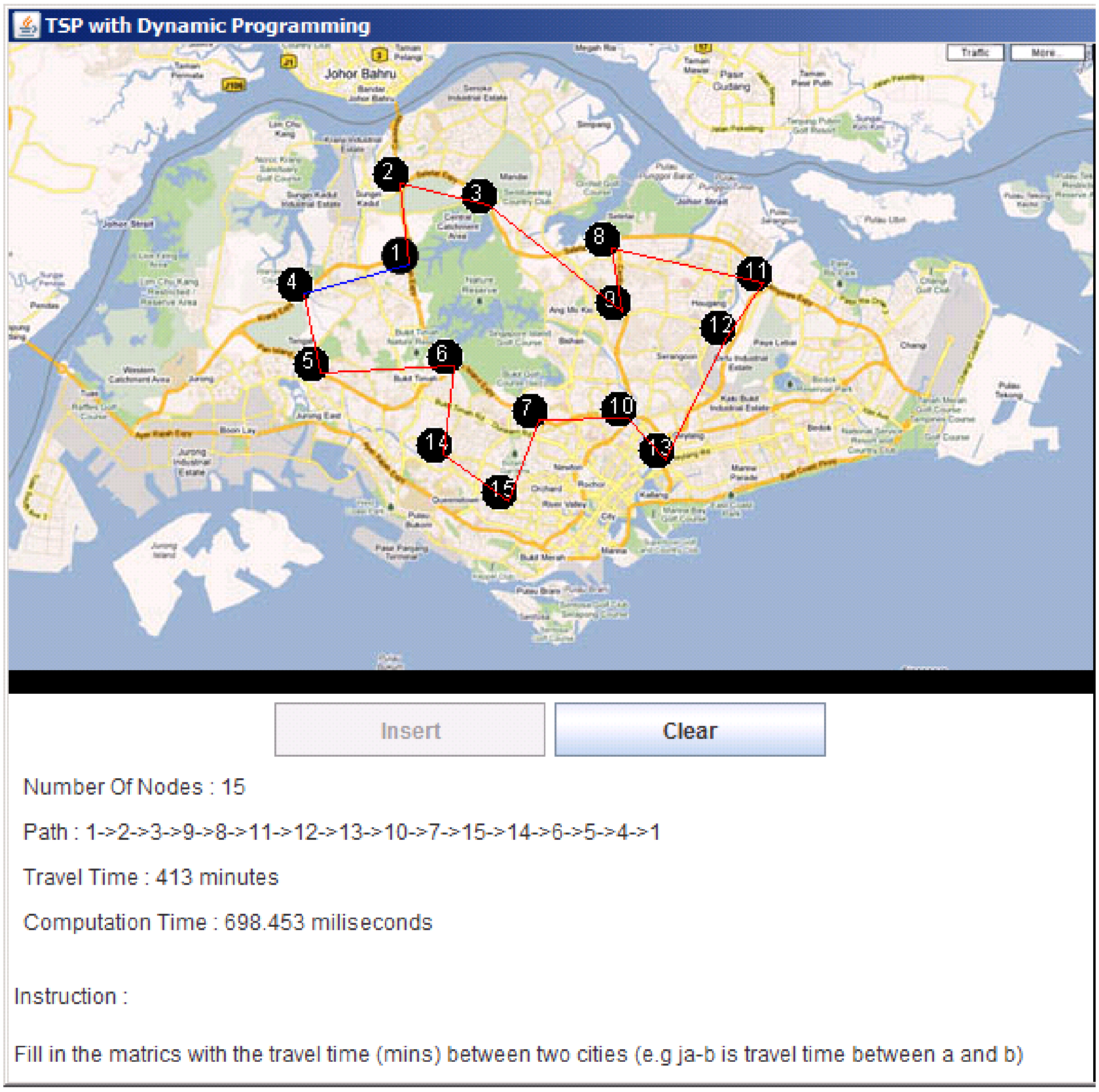
Assume the travel time is fixed between cities C*ij*without considering traffic obstruction such as accidents occured, traffic control delayed and so on. Therefore the ravel time is input into matrix form as illustrated in Fig 7.



0	30	65	30	65	45	80	90	110	95	145	165	95	85	95
45	0	30	50	60	60	85	55	65	75	90	115	145	75	85
50	45	0	50	75	55	80	20	40	70	65	65	75	65	75
50	45	55	0	25	50	65	65	95	80	135	145	120	70	80
75	50	85	35	0	35	85	95	115	95	120	145	110	45	60
55	45	45	45	40	0	25	95	90	45	115	100	85	30	35
75	75	75	55	80	30	0	45	45	18	68	76	45	35	25
120	45	25	75	105	90	55	0	15	30	25	45	50	65	95
90	55	35	90	105	95	35	10	0	34	55	35	55	75	55
95	90	60	90	85	55	15	25	40	0	55	45	15	35	45
140	95	60	120	135	110	74	35	45	65	0	15	55	65	74
120	95	55	140	130	95	69	40	45	40	25	0	40	78	115
115	120	85	95	95	65	44	45	45	10	67	45	0	45	35
95	65	60	65	55	25	40	80	68	55	85	95	50	0	18
85	95	80	105	65	45	30	105	78	40	60	105	31	23	0

Figure 7 Travel time (minutes) matrix between cities or customers C*ij* for pizza hut delivery

In addition when filled up node in the map, the ordered number is number of customer who is first called the pizza hut delivery center. Say for example, 2nd customer who located in Woodlands, 3rd customer who located in Yishun, 4th customer who located in Bukit Gombak and so on. Based on this requirements, the manager has to decide which customer should be served in order to minimize or reducing the travel time for delivery of pizza while the riders hould go back to the outlet. Given the initial results from the DP TSP algorithm as illusatrated in Fig 8. therefore the manager can decide to served customer number 1-2-3-9-8-11-12-13-10-7-15-14-6-5-4-1 with total travel time 413 minutes or 6.8 hours per day.



Number Of Nodes : 15

Path : 1->2->3->9->8->11->12->13->10->7->15->14->6->5->4->1

Travel Time : 413 minutes

Computation Time : 698.453 miliseconds

Instruction :

Fill in the matrics with the travel time (mins) between two cities (e.g ja-b is travel time between a and b)

6.0 DISCUSSION AND CONCLUSION

Based on the results application of dynamic programming algorithm in TSP by Java programme it can be concluded that:

- On application of TSP algorithm with dynamic programming is able to produce optimal route tour to served a customers including the shortest or minimum length of travel time or optimal travel time. In addition dynamic programming could reduce the enumeration of decisions that not leads to solution.
- Application of DP TSP can be applied for a small company such logistic or delivery services related as a decision tools who can help manager in deciding which city or customer should be served in order to achieve optimal value objective function.
- With minimum requirements in developing DP TSP algorithm in Java programming, therefore it is need more memory in order to compute number of *n* cities, therefore the

performance of computational times of algorithm can be compared according to number of cities being represent.

However development algorithm of DP TSP (forward recursive) in Java programming can be based to me to understood how to implement a theoretical base to support further research on how to minimum cost of travel or minimum time with more complex cases in real problem.

REFERENCES

Dreyfus, Stuart E., Law, Averill M.. 1977. The Art and Theory of Dynamic Programming. New Jersey San Fransisco London: Academic Press

Simonetti, N. 1998. Applications of a Dynamic Programming Approach to the Traveling Salesman Problem. Ph.D theses Carnegie Mellon University USA.

Evans. J. R. dan Minieka. E. 1992. Optimization Algorithms for Network and Graphs. Marcel Dekker, Inc