PROJECT REPORT

ON

**TRAVELING SALESMAN PROBLEM**

ALGORITHM, PSEUDOCODE AND DESIGN



**Submitted To: Submitted By:**

**Mr Sakkaravarthi Ramanathan Jatin Joshi (1893888)**

**Sharad Pahwa (1893664)**

**Shilpa Sharma (1893584)**

**INTRODUCTION**

Traveling Salesman Problem (TSP) is one of the important methods 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.

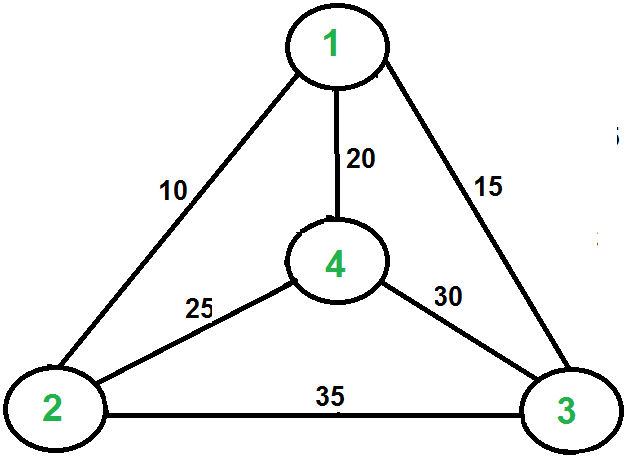
A very simple depiction of the term TSP is, a salesman who must travel ncities with the rules: he must visit each city only once. He has to minimize the total travel distance and, in the end, he had to return to his origin city. Thus, what had he done called a tour. In order to ease the problem, mapping *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.

**Problem Statement**

Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point.  
Note the difference between [Hamiltonian Cycle](https://www.geeksforgeeks.org/backtracking-set-7-hamiltonian-cycle/) and TSP. The Hamiltoninan cycle problem is to find if there exist a tour that visits every city exactly once. Here we know that Hamiltonian Tour exists (because the graph is complete) and in fact many such tours exist, the problem is to find a minimum weight Hamiltonian Cycle. For example, in this picture, A TSP tour in the graph is 1-2-4-3-1. The cost of the tour is 10+25+30+15 which is 80.

[](https://www.geeksforgeeks.org/wp-content/uploads/Euler12.png)

**Related Work**

In literature TSP is used in two forms: i) combinatorial optimization version and ii) decision version. In first version it is used to find a minimum Hamiltonian cycle and in later version to check the existence of smaller graph.

Theoretical computer science and operations research, both fields of combinatorial optimization contain TSP. In this problem, set of cities are given with their distances to find the shortest route to each city without visiting a city twice. In 1930, it was first formulated as a mathematical model and applied to so many areas to find their optimal solutions e.g., Clustering of array of data, Handling of a warehouse materials and crystal structure analysis. Resource constrained scheduling problem with aggregate deadline also solved with TSP. Researches took orienteering and prize collection problems as special cases of resource constrained TSP. One of the best known and more complex combinatorial problem is Vehicle routing problem, to determine the order of vehicle for customers serving from fleet of vehicles, is being solved by TSP.

TSP is easy to understand but it’s really difficult to solve it. In 1972, Richard M. Karp showed that it is NP-complete to solve Hamiltonian cycle problem, which explains the NP-hardness of TSP. This can be judged that how much computational difficulty is attached to find the optimal route. With the passage of time TSP is getting more and more sophisticated and solving instances are larger now. A brief view of TSP milestones is given below in Table 1.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Year** | **Research Team** | **Size of Instance** |
|  | 1954 | G. Dantzig, R. Fulkerson, and S. Johnson [14] | 49 cities |
|  | 1971 | M. Held and R.M. Karp [15] | 64 cities |
|  | 1975 | P.M. Camerini, L. Fratta, and F. Maffioli [16] | 67 cities |
|  | 1977 | M. Grötschel [17] | 120 cities |
|  | 1980 | H. Crowder and M.W. Padberg [18] | 318 cities |
|  | 1987 | M. Padberg and G. Rinaldi [19] | 532 cities |
|  | 1987 | M. Grötschel and O. Holland [20] | 666 cities |
|  | 1987 | M. Padberg and G. Rinaldi [19] | 2,392 cities |
|  | 1994 | D. Applegate, R. Bixby, V. Chvátal, and W. Cook [21] | 7,397 cities |
|  | 1998 | D. Applegate, R. Bixby, V. Chvátal, and W. Cook [22] | 13,509 cities |
|  | 2001 | D. Applegate, R. Bixby, V. Chvátal, and W. Cook [23] | 15,112 cities |
|  | 2006 | D. Applegate, R. Bixby, V. Chvátal, W. Cook, | 24,978 cities |
|  |  | and K. Helsgaun [23] |  |

Table 1. Summary of the Milestones achieved in TSP

**Different approaches**

**Heuristic Solution Techniques**

There are various heuristics and approximate solution approaches, which had been devised during last decades, to find solution within reasonable time and with 2-3 % optimality gap. There are numerous approaches used to solve TSP shown in literature. Some important and mostly used approaches are enlisted here with their application algorithms. Variables for these algorithms are illustrated in Table 2

|  |  |
| --- | --- |
| **Description** | **Variable** |
| Initial solution provided by the user | S |
| Objective function value | Z |
| The best solution | S\* |
| Cities in TSP (Nodes in transportation network) | i, j |
| Population | P |
| Generations | G |
| Generation counter | Ngen |
| Initial temperature | T |
| Cooling factor | R |
| Number of times the temperature *T* is decreased | *ITEMP* |
| Maximum number of new solutions to be accepted at each temperature | *NLIMIT* |
| Maximum number of solutions evaluated at each temperature | *NOVER* |
| Gain parameter | *L* |

**Brute Force Method**

Algorithm for TSP using Brute-force method contains the following steps:

Algorithm 1: TSP using Brute Force Method

Step 1: calculate the total number of tours (where cities represent the number of nodes).

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.

**Greedy Approach**

Greedy approach solves TSP by using the five steps, given in Algorithm 2.

Algorithm 2: TSP using Greedy Approach

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.

**Nearest Neighbor Heuristic**

Algorithm 3 shows the methodology to solve TSP by Nearest Neighbor Heuristic.

Algorithm 3: TSP using Nearest Neighbor Heuristic

.

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.

**Branch and Bound Method**

Branch and bound technique is commonly used optimization technique. Formulation of TSP by using branch and bound technique is given in Algorithm 4.

Algorithm 4: TSP using Branch and Bound Method

.

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.

**2-Opt Algorithm;**

For n nodes in the TSP problem, the 2-Opt algorithm consists of the steps shown in Algorithm 5.

Algorithm 5: TSP using 2-Opt Algorithm

Step 1: Let S be the initial solution provided by the user and z its objective function value. Set S\*=s,

z\*=z, i=1 and j=i+1=2.

Step 2: Consider the exchange results in a solution S that has objective function value z’<z\*, set z\*=z’

and S\*=S’. If j<n repeat step 2; otherwise set i=i+1 and j=i+1. If i<n, repeat step 2; otherwise go

to step 3.

Step 3: If S≠S\*, set S=S\*, z=z\*, i=1, j=i+1=2 and go to step 2. Otherwise, output S\* as the best solution and terminate the process.

**Genetic Algorithm**

This algorithm is based on genetics. The Genetic Algorithm works as shown in Algorithm 6.

Algorithm 6: TSP using Genetic Algorithm

Step 0: Obtain the maximum number of individuals in the population P and the maximum number of generations G from the user, generate P solutions for the first generation’s population randomly, and represent each solution as a string. Set generation counter Ngen=1.

Step 1: Determine the fitness of each solution in the current generation’s population and record the

string that has the best fitness.

Step 2:

Generate solutions for the next generation’s population as follows:

1. Retain 0.1P of the solutions with the best fitness in the previous population.
2. Generate 0.89P solutions via mating, and
3. Select 0.01P solutions from the previous population randomly and mutate them.

Step 3: Update Ngen= Ngen+1. If Ngen ≤ G, go to Step 1. Otherwise stop.

**Simulated Annealing (SA)**

Statistical mechanics is the basic idea of simulated annealing (SA). Analogy of the behavior of physical systems in the heat bath is main motivation of SA. Solution state is represented by the temperature. Initiating with an initial temperature, algorithm moves to the next temperature until it reaches a frozen state.

Algorithm 7: TSP using Simulated Annealing

Step0: Set *S* = initial feasible solution

Step 1: Repeat step 2 *NOVER* times or until the number of successful new solutions is equal to

*NLIMIT*

Step 2: Pick a pair of machines randomly and exchange their positions.

Step 3: Set *T = rT* and *ITEMP = ITEMP + 1*. If *ITEMP* <= 100, go to step 1; otherwise stop.

**Comparison Between Different Algorithms**

Previous section demonstrates the different methodologies to solve NP-hard TSP problem approximately. Although, different techniques had been devised previously, but, all available techniques are not efficient in terms of solution time and quality. Comparison by Maredia [ shows that Nearest Neighbor heuristic works well but it is not sure that it will give us solutions good as brute force. Moreover, greedy approach is not a good approximation technique for TSP. Comparison of Branch and bound technique with brute force method is presented in Figure 3 (data extracted from Maredia ). SA algorithm obtains has ability to find the good quality final solutions by mechanism of gradually going from one solution to the next. The main difference of SA from the 2-opt is that the local optimization algorithm is often restrict their search for the optimal solution in a downhill direction which mean that the initial solution is changed only if it results in a decrease in the objective function value**.** However, it is shown by Kim et al. that 2-opt algorithm works well when the problem size is less than 50 cities. For comparison of the approaches we are referring to data Kim et al. Data extracted from Kim et al is plotted in Figure 4 and 5, according to results it is obvious that the solution of TSP using the neural network outperforms than all other approaches for all problem sizes. However, if we compare the time consumed to solve the problems, it is easy to realize that neural network approach is taking much more time as compared to others. Genetic algorithm has been used for many optimization problems; however, the use of genetic algorithm for TSP has disadvantages of premature convergence and poor local search capability. These disadvantages can be overcome by adaptation of other efficient working algorithms e.g., artificial immune systems.

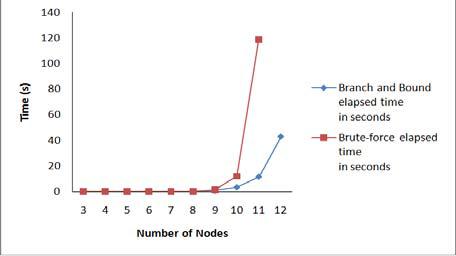


Figure. Comparison of Branch and Bound Technique with Brute Force Method

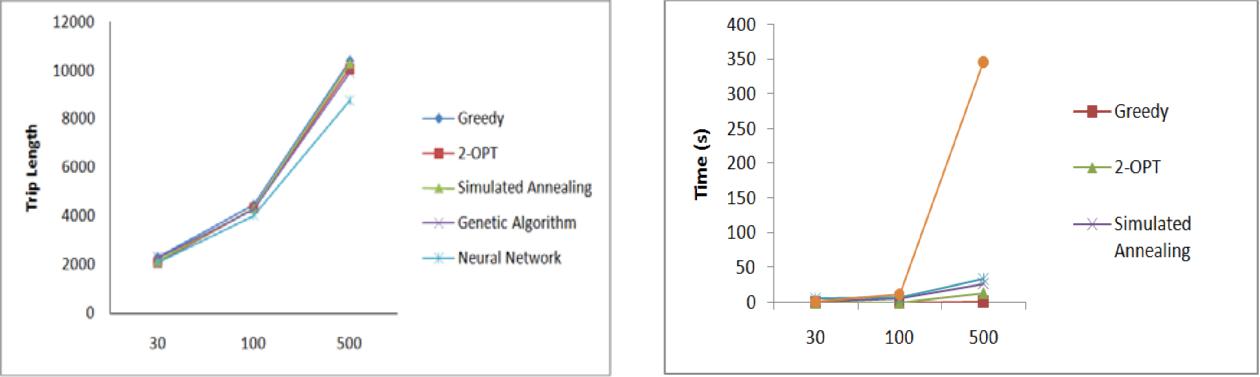
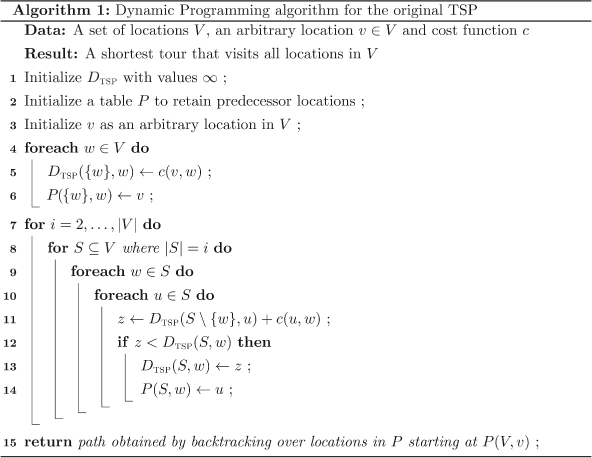


Figure . Length Comparison of TSP

Heuristics

Figure. Time Comparison of TSP Heuristic

**# Dynamic Programming**



**Pseudocode**

function algorithm TSP (G, n)

for k := 2 to n do

C({k}, k) := d1,k

end for

Pseudocode

for s := 2 to n-1 do

for all S ⊆ {2, . . . , n}, |S| = s do

for all k ∈ S do

C(S, k) := minm≠k,m∈S [C(S\{k}, m) + dm,k]

end for

end for

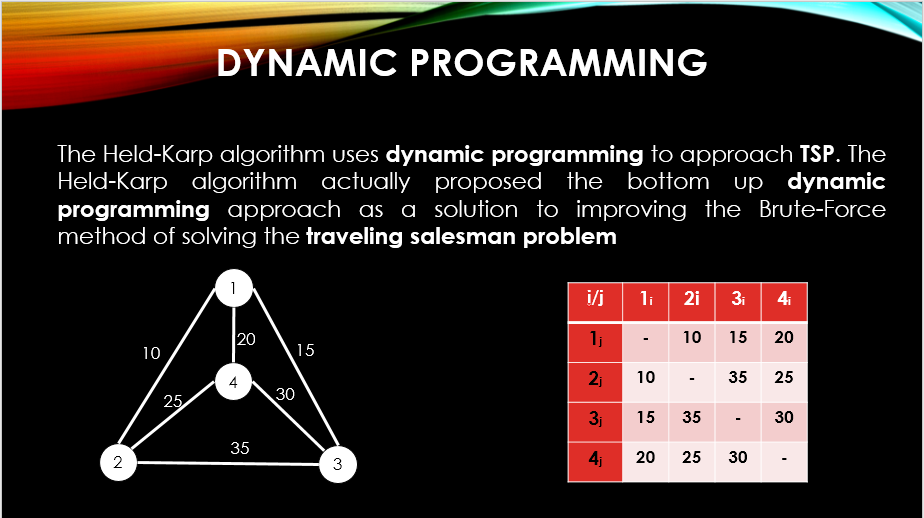
end for

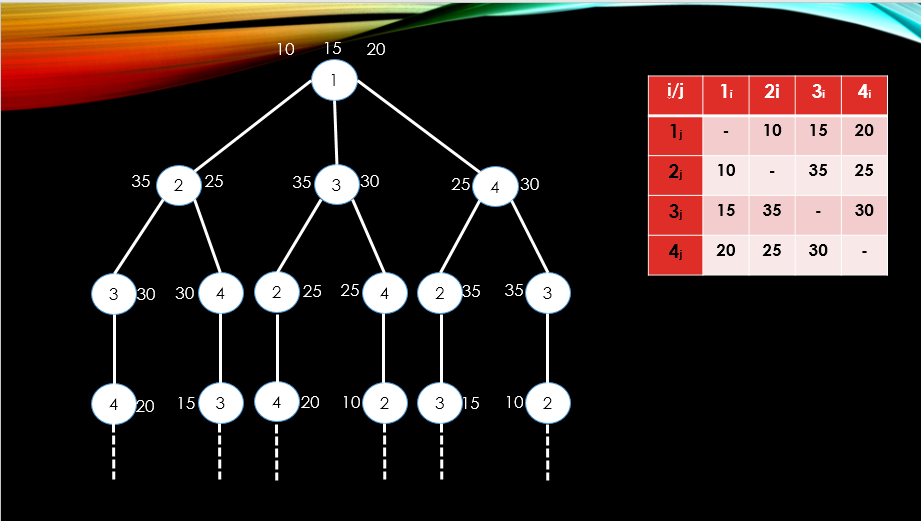
opt := mink≠1 [C({2, 3, . . . , n}, k) + dk,1]

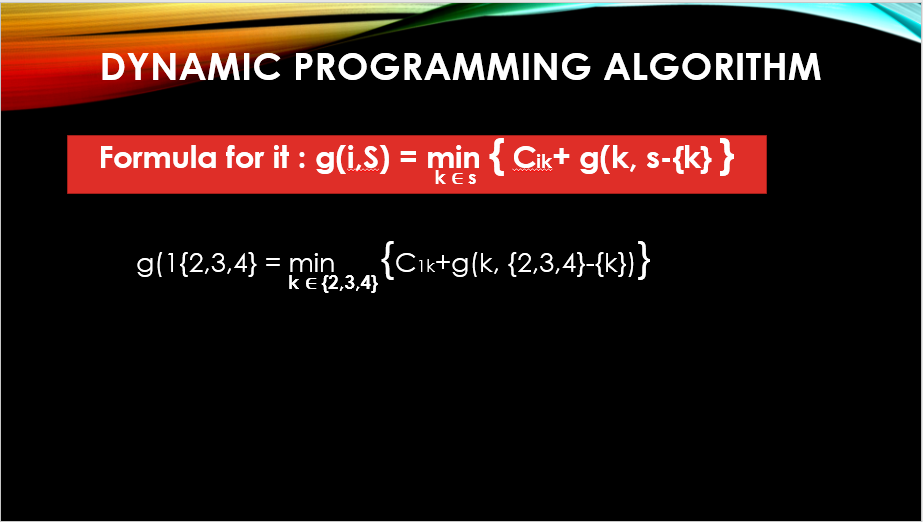
return (opt)

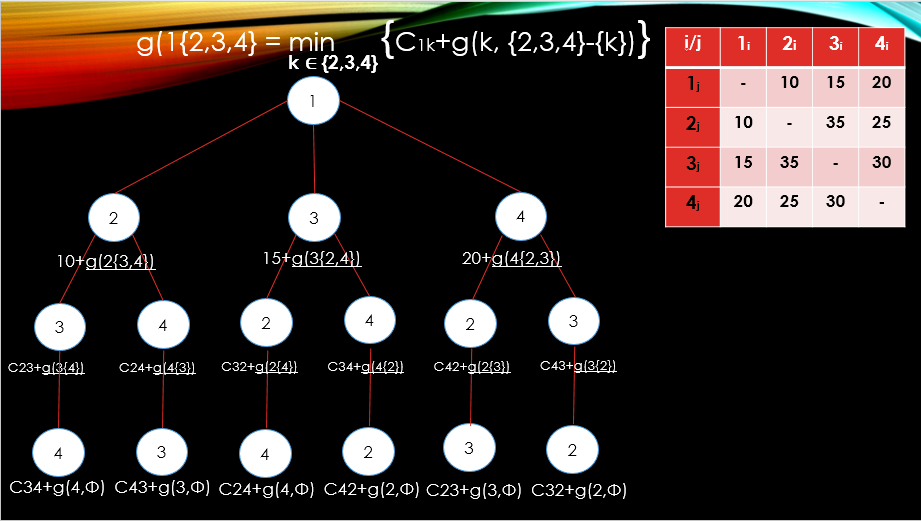
end

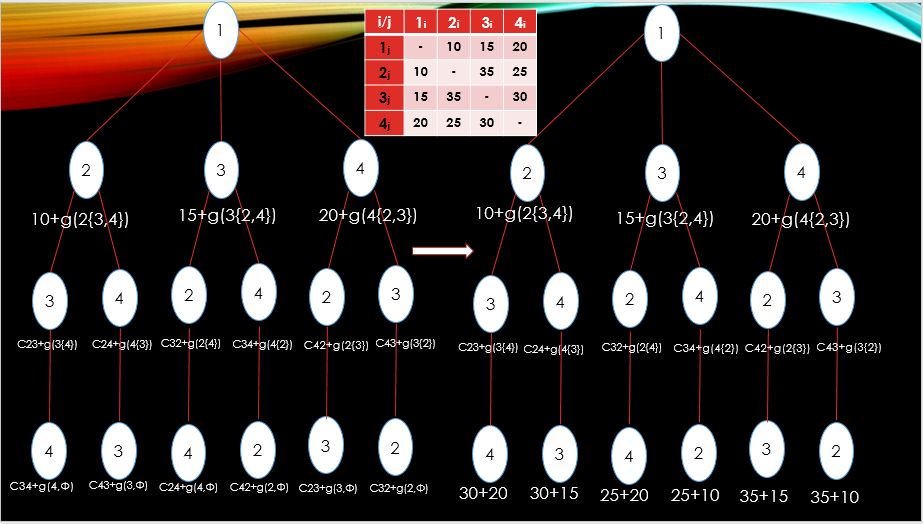
**Example of Dynamic Programming**

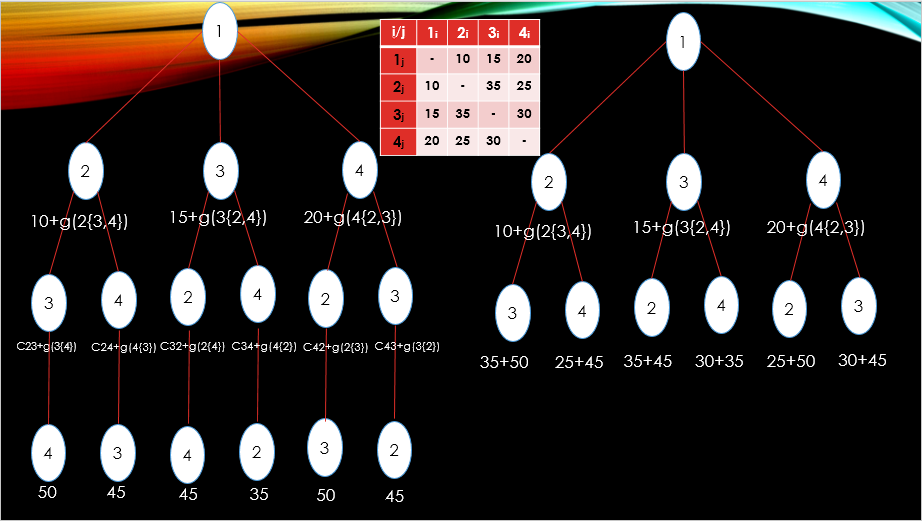


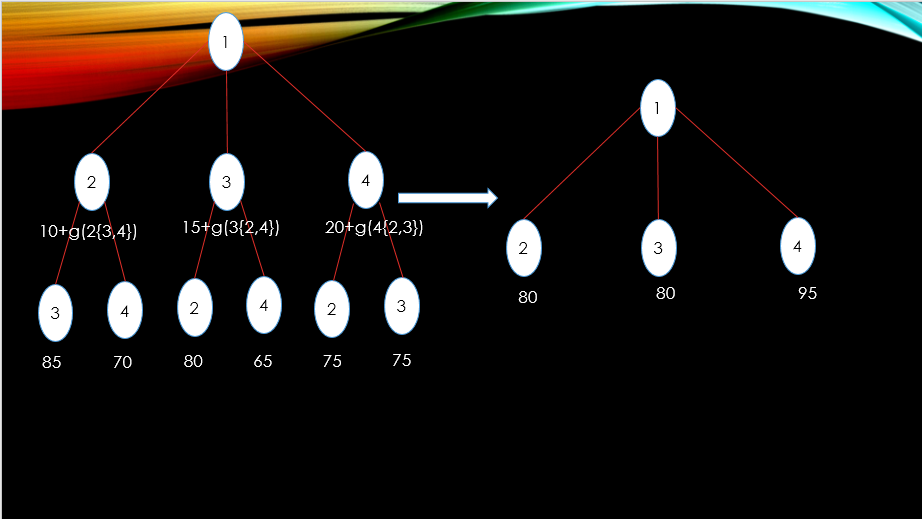


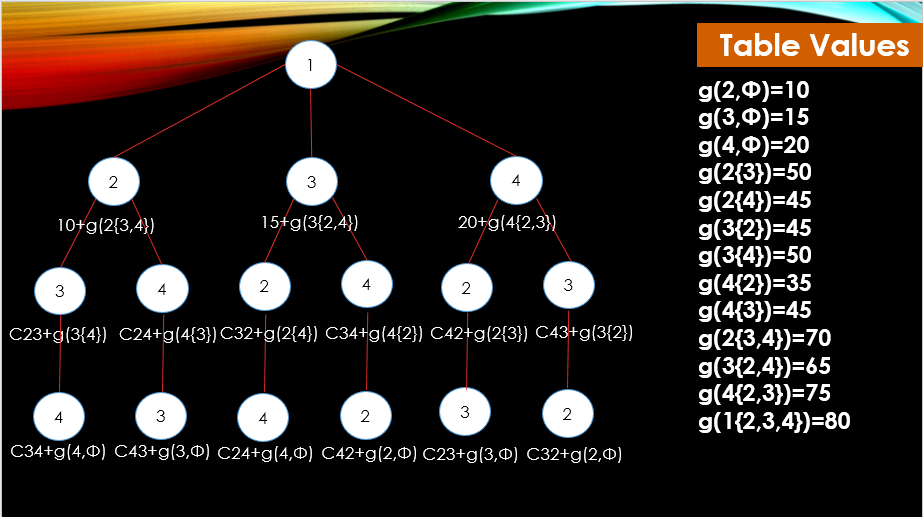






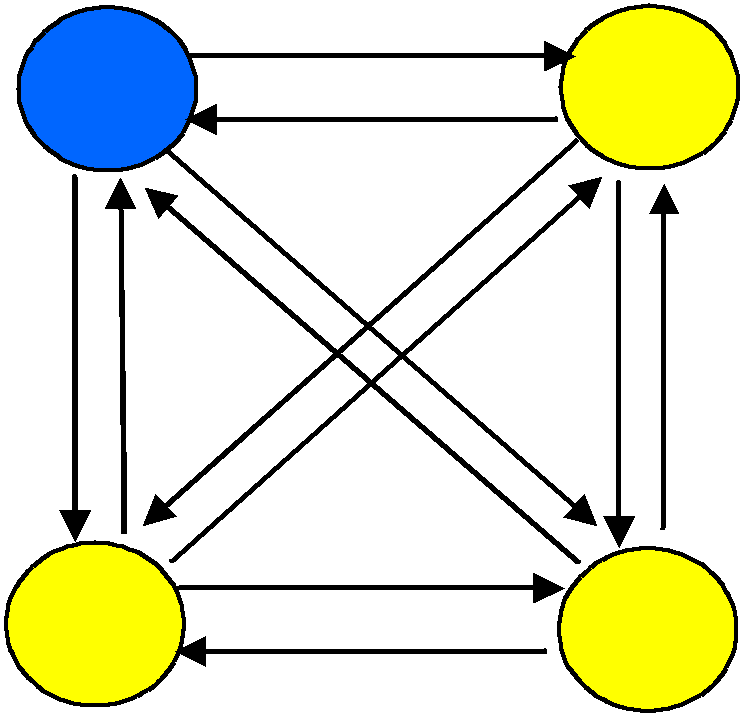






**Numerical examples**

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:

12

1 2

15

8

9 16

10

14 15

11 11

17

4 3

18

Figure 4 Graph with 4 vertices

Here distance from i to j (the distance between cities) cij

, where c *ij* have converted

into matrix form:

0 12 11 16

15 0 15 10

*c*

8 14 0 18

9 11 17 0

The following is solution steps:

**Step 1**

Calculate

We get:

*f* (*i*, ) *c* ,1, 2 *i*  *n*

*i*

*f* (2, ) *c21*,15

*f* (3, ) *c31*,8

*f* (4, ) c41,9

**Step 2**

For | S | = 1, calculate

We get:

*f* (*i*, *s*) min*cij* *f* ( *j*,*Sj*)

*jS*

*f* (2,3) *c*23 *f* (3,) 15 8 23

*f* (3,2) *c*32 *f* (2,) 14 15 29

*f* (4,2) *c*42 *f* (2,) 1115 26

*f* (2,4) *c*24 *f* (4,) 10 9 19

*f* (3,4) *c*34 *f* (4,) 18 9 27

*f* (4,3) *c*43 *f* (3,) 17 8 25

For | S | = 2, calculate

We get:

*f* (*i*, *s*) min*cij* *f* ( *j*,*Sj*)

*jS*

*f* (2,3,4) min*c*23 *f* (3,4),*c*24 *f* (4,3)

= min {15+27,10+25}

= min {42,35} = 35

*f* (3,2,4) min*c*32 *f* (2,4),*c*34 *f* (4,2)

= min {14+19,27+11}

= min {33,38} = 33

*f* (4,2,3) min*c*42 *f* (2,4),*c*43 *f* (3,2)

= min {11+22,17+29}

= min {33,46} = 33

**Step 3**

By using the equation,

We get:

*f* 1*,v*1*=* min *c*1*k*  *f* (*k*,*V* 1,*k*)

2*kn*

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*  *f* (3,2,4) where found that the initial part of the path is 1-3.

13

Then find out the component of

*f* (3,2,4) which resulting the minimum value is

*c*  *f* (2,4) .

32

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*  *f* (4, ) . The final step is to trace forming of

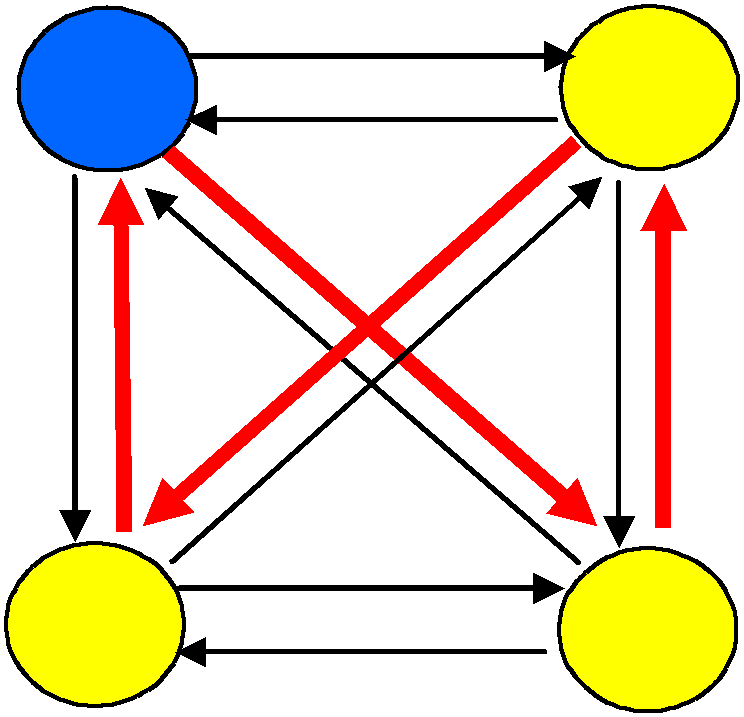
24

*f* (4, ) which found is

*c* (Thus edge of 4-1 is also part of the shortest path). By assembling

41

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):

12

1 2

15

8

9 16

10

14

**JAVA PROGRAM**

**import** java.util.\*;

**import** java.text.\*;

**class** TSP

{

**int** weight[][],n,tour[],finalCost;

**final** **int** INF=1000;

**public** TSP()

{

Scanner s=**new** Scanner(System.***in***);

System.***out***.println("Enter no. of nodes:=>");

n=s.nextInt();

weight=**new** **int**[n][n];

tour=**new** **int**[n-1];

**for**(**int** i=0;i<n;i++)

{

**for**(**int** j=0;j<n;j++)

{

**if**(i!=j)

{

System.***out***.print("Enter weight of "+(i+1)+" to "+(j+1)+":=>");

weight[i][j]=s.nextInt();

}

}

}

System.***out***.println();

System.***out***.println("Starting node assumed to be node 1.");

eval();

}

**public** **int** COST(**int** currentNode,**int** inputSet[],**int** setSize)

{

**if**(setSize==0)

**return** weight[currentNode][0];

**int** min=INF,minindex=0;

**int** setToBePassedOnToNextCallOfCOST[]=**new** **int**[n-1];

**for**(**int** i=0;i<setSize;i++)

{

**int** k=0;//initialise new set

**for**(**int** j=0;j<setSize;j++)

{

**if**(inputSet[i]!=inputSet[j])

setToBePassedOnToNextCallOfCOST[k++]=inputSet[j];

}

**int** temp=COST(inputSet[i],setToBePassedOnToNextCallOfCOST,setSize-1);

**if**((weight[currentNode][inputSet[i]]+temp) < min)

{

min=weight[currentNode][inputSet[i]]+temp;

minindex=inputSet[i];

}

}

**return** min;

}

**public** **int** MIN(**int** currentNode,**int** inputSet[],**int** setSize)

{

**if**(setSize==0)

**return** weight[currentNode][0];

**int** min=INF,minindex=0;

**int** setToBePassedOnToNextCallOfCOST[]=**new** **int**[n-1];

**for**(**int** i=0;i<setSize;i++)//considers each node of inputSet

{

**int** k=0;

**for**(**int** j=0;j<setSize;j++)

{

**if**(inputSet[i]!=inputSet[j])

setToBePassedOnToNextCallOfCOST[k++]=inputSet[j];

}

**int** temp=COST(inputSet[i],setToBePassedOnToNextCallOfCOST,setSize-1);

**if**((weight[currentNode][inputSet[i]]+temp) < min)

{

min=weight[currentNode][inputSet[i]]+temp;

minindex=inputSet[i];

}

}

**return** minindex;

}

**public** **void** eval()

{

**int** dummySet[]=**new** **int**[n-1];

**for**(**int** i=1;i<n;i++)

dummySet[i-1]=i;

finalCost=COST(0,dummySet,n-1);

constructTour();

}

**public** **void** constructTour()

{

**int** previousSet[]=**new** **int**[n-1];

**int** nextSet[]=**new** **int**[n-2]; **for**(**int** i=1;i<n;i++)

previousSet[i-1]=i;

**int** setSize=n-1;

tour[0]=MIN(0,previousSet,setSize);

**for**(**int** i=1;i<n-1;i++)

{

**int** k=0;

**for**(**int** j=0;j<setSize;j++)

{

**if**(tour[i-1]!=previousSet[j])

nextSet[k++]=previousSet[j];

}

--setSize;

tour[i]=MIN(tour[i-1],nextSet,setSize);

**for**(**int** j=0;j<setSize;j++)

previousSet[j]=nextSet[j];

}

display();

}

**public** **void** display()

{

System.***out***.println();

System.***out***.print("The tour is 1-");

**for**(**int** i=0;i<n-1;i++)

System.***out***.print((tour[i]+1)+"-");

System.***out***.print("1");

System.***out***.println();

System.***out***.println("The final cost is "+finalCost);

}

**public** **static** **void** main(String args[])

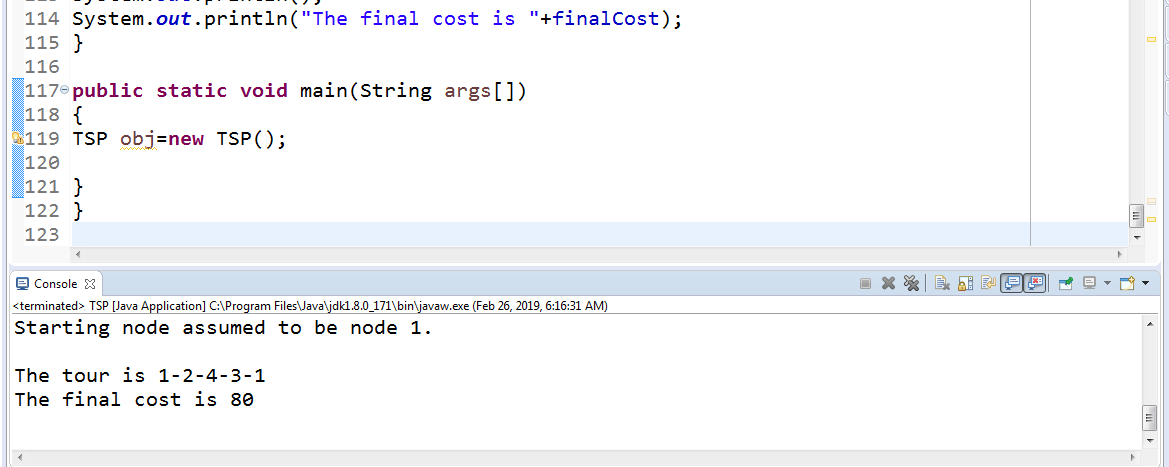
{

TSP obj=**new** TSP();

}

}

**Output**



**Conclusion**

TSP is easy to understand; however, it is very difficult to solve. It is NP hard problem. Due to complexity involved with exact solution approaches it is hard to solve TSP within feasible time. That’s why different heuristics are generally applied to solve TSP. Heuristics to solve TSP are presented here with detailed algorithms. At the end, comparison among selected approaches shows the efficiency of approaches in terms of solution quality and time consumed to solve TSP.

Advantages of these heuristics’ approaches are that they are simple and easy to understand. These require less programming and storage requirements and produce multiple solutions with in less time. However, heuristics local improvements in heuristics can be source of lack in global prospective of objective function.

**REFERENCES**

1. Lawler EL, Lenstra JK, Rinnooy AHG, Shmoys DB. The Traveling Salesman Problem. Chichester: John Wiley. 1985.
2. Ratliff HD, Rosenthal AS. *Order-Picking in a Rectangular Warehouse: A Solvable Case for the* *Traveling Salesman Problem*. PDRC Report Series No. 81-10. 1981.
3. Bland RE, Shallcross DF. *Large Traveling Salesman Problem Arising from Experiments in X-ray* *Crystallography: a Preliminary Report on Computation.* Technical Report No. 730. 1987.
4. Miller, Pekny J. Exact Solution of Large Asymmetric Traveling Salesman Problems. *Science 251.* 1991: 754-761.
5. Balas E. The Prize Collecting Traveling Salesman Problem. *Networks 19.* 1989: 621-636.
6. Golden BL, Levy L, Vohra R. The Orienteering Problem. *Naval Research Logistics 34.* 1987: 307-318.
7. Christofides N. Vehicle Routing, in The Traveling Salesman Problem. In: Lawler, Lenstra, Rinooy Kan