**Travelling Salesman Problem using Dynamic Programming**

**Government College University, Lahore**

**Name: Muhammad Hussain**

**Roll No. 180-bscs-20**

**Section ‘C’**

**Submitted To: Mr. Saeed Ur Rehman Turk**

**ABSTRACT**

In this term paper, the Travelling Salesman Problems using dynamic programming is discussed, which will help the salesmen to travel determined cities in minimum time. This paper presents and exact solution for the traveling salesman problem based on dynamic programming to reach an optimal solution. A dynamic programming algorithm solves each subproblem just once and saves its answer in a table, which later can be used in the other subproblems. In this term paper experiments on the traveling salesman problem are applied in which the problem will take minimum time.

**GENERAL TERMS**

Travelling salesman problem (TSP)

1. **INTRODUCTION**

Advancement in technology and internet leads to fast problem solving, several internet retailers and restaurants which deliver food and products for their clients need the fasted delivery system, to provide maximum service in minimum time. Moreover, TSP has several applications in planning, logistics, manufacture of microchip, mostly, it appears as a sub-problem in many areas. The traveling salesman problem (TSP) is an [algorithmic](https://www.techtarget.com/whatis/definition/algorithm) problem, the task is to find the shortest route between a set of points and locations that must be visited. TSP describes a salesman who must travel between N cities. The order in which he travels the cities are not important, as long as he visits each city only once during his trip, and finishes where he started. Each city is connected to other close cities, or nodes, by [airplanes](https://simple.wikipedia.org/wiki/Airplane), or by [road](https://simple.wikipedia.org/wiki/Road) or [railway](https://simple.wikipedia.org/wiki/Railway). Each of those links between the cities has one or more weights (cost) attached. The cost describes how "difficult" it is to traverse this edge on the graph, and may be given, for example, by the cost of an airplane ticket or train ticket, or perhaps by the length of the edge, or time required to complete the traversal. The salesman wants to keep both the travel costs, as well as the [distance](https://simple.wikipedia.org/wiki/Distance) he travels as low as possible.

The Traveling Salesman Problem is typical of a large class of "hard" optimization problems that have made

Mathematicians and computer scientists curious for years. Most important, it has applications in science and engineering. For example, in the manufacture of a circuit board, it is important to determine the best order in which a laser will drill thousands of holes. An efficient solution to this problem reduces production costs for the manufacturer.

This problem has been treated by a number of different people using variety of solutions and techniques, the purpose of this term paper is to determine that this problem can be easily formulated and solved by dynamic programming approach.

* 1. **BACKGROUND**

Traveling Salesman Problem(TSP) a NP hard (nondeterministic polynomial time) problem which is generally calculated in fields like Computer Science and Mathematics. TSP was first defined approximately 150 years ago and it used to be known as a complex problem till 1940’s. After a decade of research, still there wasn’t a complete solution for the TSP this caused it to appear in the interest of computer scientists and mathematicians. A solution for TSP could be very valuable as it could solve various problems.

Typically Travelling Salesman problem is declared as a dilemma when a salesman needs a trip of some given cities one time, starting from any arbitrary city and visiting all cities only once and returning to the original place of departure. So, the question arises that what route should he pick in order to decrease the total distance traveled. The difficulty was to get the shortest possible route once when the salesmen need to stopover their clients and end where they begin. This same problem now applies to a huge number of activities and forms the basis of solution to these modern area issues. For instance, distributing ndist stock to super marts, delivering products or food to clients, providing manufacturing lines, sky travel control, and even chromosome progression.

TSP was first considered mathematically in 1930s by Merrill Meeks Flood who was trying to solve a school bus routing problem. [Hassler Whitney](https://en.wikipedia.org/wiki/Hassler_Whitney) at [Princeton University](https://en.wikipedia.org/wiki/Princeton_University) was interest in this problem, which he called the "48 states problem". The earliest publication

using the phrase "travelling salesman problem" was the 1949 [RAND Corporation](https://en.wikipedia.org/wiki/RAND_Corporation) report by [Julia Robinson](https://en.wikipedia.org/wiki/Julia_Robinson), “On the Hamiltonian game (a traveling salesman problem).”

In the 1950s and 1960s, the problem became increasingly popular in scientific circles. In1962 Bellman and Held and Karp proposed a dynamic programming algorithm independently which is called The Held–Karp algorithm. The algorithm is to solve the [traveling salesman problem](https://en.wikipedia.org/wiki/Traveling_salesman_problem) (TSP), in which the input is a distance matrix between a set of cities, and the goal is to find a minimum-length tour that visits each city exactly once before returning to the starting point. It finds the exact solution to this problem, and to several related problems including the [Hamiltonian cycle problem](https://en.wikipedia.org/wiki/Hamiltonian_path_problem), in [exponential time](https://en.wikipedia.org/wiki/Exponential_time).

**PROBLEM DESCRIPTION**

A traveler needs to visit all the cities from a list, where distances between all the cities are known and each city should be visited just once. Travelling salesman problem can be modeled as a set of n locations or cities, n = {1, 2, 3, …, n} locations and a matrix which show the cost of every path. We assume that vertex 1 is our initiating location (selecting the starting vertex is arbitrary so we can choose any vertex as starting point) the salesman has to visit every vertex only and only one time and has to return to where he started. The objective is to find the minimum cost path for the traversal and return back to initial point. Naive or brute force approach can solve this problem by solving every possible route and chose the shortest path which takes O(n!) time, which is extremely huge time. dynamic programming approach can solve the subproblems, store their value and use them in other subproblems which will reduce the compilation time to O(2^n \* n^2)

**DYNAMIC PROGRAMMING APPROACH**

The Travelling Salesman Problem is a problem in combinatorial optimization, solution in short amount of time is required. With the particular set of cities and their pair-wise distances, scheme is to find a undeviating tour that each city is visited just one time with the length of the tour to be minimized.

Dynamic programming approach refers to simplifying a complicated problem by breaking it down into simpler sub-problems in a [recursive](https://en.wikipedia.org/wiki/Recursion) manner. While some decision problems cannot be taken apart this way, decisions that span several points in time do often break apart recursively. Likdistise, in computer science, if a problem can be solved optimally by breaking it into sub-problems and then recursively finding the optimal solutions to the sub-problems, then it is said to have [optimal substructure](https://en.wikipedia.org/wiki/Optimal_substructure).

Dynamic algorithm begins calculating the shortest path for each set of cities S ⊆ {2, 3, …, n}, 1(starting point) is not included in S, the shortest path to visit every city and return back to depot is the solution for the TSP. firstly ({k}, k) is calculated which is equal to dist(1, k).

Terms: S is a set of vertices, a history of visited vertices and k is a vertex. C (S, k) is the distance/ cost to start from the starting point or depot. Visit every vertex in S and end in k. P (S, k) is vertex previous visited before going to k in another words we can say it is k’s parent. dist (x, y) is the edge from x to y.

**Pseudocode**

**if size of S is e then there must be (1, i)**

**C(S, i) = dist(1, i)**

**If size of S is greater than 2**

**C(S, j) min{C(S – {i}, j) + dist(j, i)} where j belong to S, j != i and j != 1**

In this algorithm, we take a subset of the required cities needs to be visited, distance among the cities and starting city as inputs. Each city is identified by unique city id like. Initially, all cities are unvisited, and the visit starts from the city S. We assume that the initial travelling cost is equal to 0. Next, the TSP distance value is calculated based on a recursive function. If the number of cities in the subset is two, then the recursive

function returns their distance as a base case. On the other hand, if the number of cities is greater than 2, then we’ll calculate the distance from the current city to the nearest city, and the minimum distance among the remaining cities is calculated recursively.

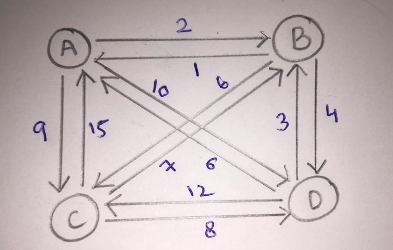
Finally, the algorithm returns the minimum distance as a TSP solution.

**Time Complexity**

Brute force approach will solve this problem in (n-1)! time which is really numerous and we don’t want that time and dynamic approach solves travelling salesman problem in exponential time of (n^2 \* 2^n), this is also not best time but it is far better than factorial time. As it is generating subsets of a set S and the maximum number of subsets it generates is (n\*2^n) and every subset takes n time to compute.

So, the total time required for travelling salesman problem is O(n^2 \* 2^n).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | A | B | C | D |
| A | 0 | 2 | 9 | 10 |
| B | 1 | 0 | 6 | 4 |
| C | 15 | 7 | 0 | 8 |
| D | 6 | 3 | 12 | 0 |



Step 1: Base cases (size = 1)

C ({B}, B) = 1🡪 cost of B to A

C ({C}, C) = 15🡪 cost of C to A

C ({D}, D) = 6🡪 cost of D to A

Step 2: (size = 2)

{B, C}, {B, D}, {C, D}

C ({B, C}, B) = C ({C}, C}+ dist (C, B) = 15 + 6 = 21

C ({B, C}, C) = C ({B}, B}+ dist (B, C) = 1 + 7 = 8

C ({B, D}, B) = C ({D}, D}+ dist (D, B) = 6 + 4 = 10

C ({B, D}, D) = C ({B}, B}+ dist (B, D) = 1 + 3 = 4

C ({C, D}, C) = C ({D}, D}+ dist (D, C) = 6 + 8 = 14

C ({C, D}, D) = C ({C}, C}+ dist (C, D) = 15 + 12 = 27

P(C ({B, C}, B) = C

P(C ({B, C}, C)) = B

P(C ({B, D}, B)) = D

P(C ({B, D}, D)) = B

P(C ({C, D}, C)) = D

P(C ({C, D}, C)) = C

Step 3: (size 3)

{B, C, D}

C({B, C, D}, B) = min[ C({C, D}, C) + dist(C, B), C({C, D}, D), dist(D, B)]

= min [14 + 6, 27 + 4] = min [20, 31] = 20

C ({B, C, D}, C) = min [C({B, D}, B) + dist(B, C), C({B, D}, D) + dist(D, C)]

= min [10 + 7, 4 + 8] = min [17, 12] = 12

C ({B, C, D}, D) = min [C({B, C}, B) + dist(B, D), C({B, C}, C) + dist(C, D)]

= min [21 + 3, 8 + 12] = min [24, 20] = 20

P ({B, C, D}, B) = C

P ({B, C, D}, C) = D

P ({B, C, D}, D) = C

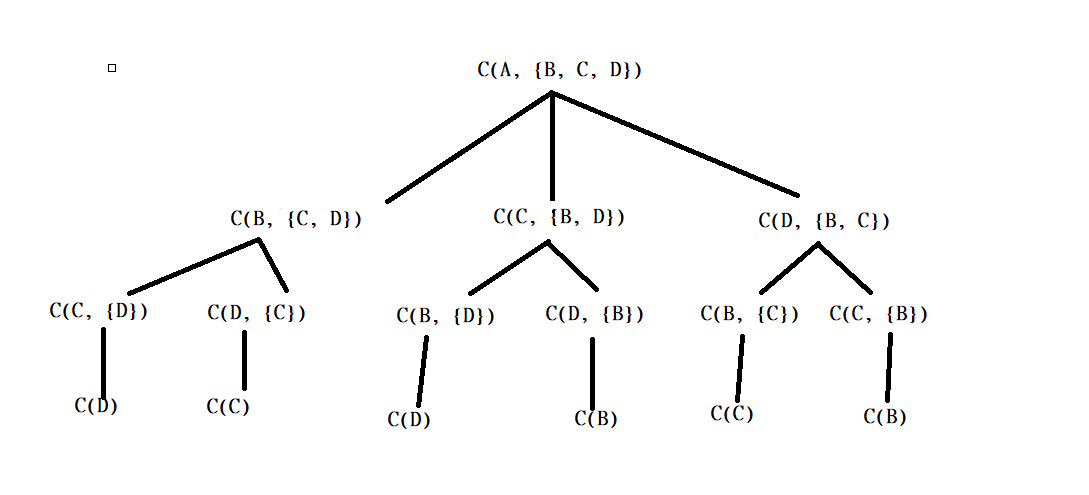
Step 4: Back to A (size 4)

C ({A, B, C, D}, A) = min [({B, C, D}, B) + dist(B, A) , ({B, C, D}, C) + dist(C, A), ({B, C, D}, D) + dist(D, A)]

= min [20 + 2, 12 + 9, 20 + 10] = min [22, 21, 30] = 21

P ({A, B, C, D}, A) = C

A 🡪 B 🡪 D 🡪 C 🡪 A (the selected root for this tour)



1

2

3

4

**Reference**

1. 'A dynamic programming approach to sequencing problems’, Michael Held and Richard M. Karp, Journal for the Society for Industrial and Applied Mathematics
2. ‘Dynamic programming treatment of the travelling salesman problem’, Richard Bellman, Journal of Assoc. *Computing Mach*
3. https://en.wikipedia.org/wiki/Held%E2%80%93Karp\_algorithm