Application of Travelling Salesman Problem in a Printed Circuit Board using Dynamic Programming

Kotla Karthik Reddy
181IT123
Information Technology
National Institute of
Technology,Karnataka
Surathkal,India 575025
karthikreddykotla1729@gmail.com

Yash Kumar Gupta
l81IT153
Information Technology
National Institute of
Technology,Karnataka
Surathkal,India 575025
yashkumar0707@gmail.com

ABSTRACT

The aim of the paper is an application of the travelling salesman problem. In this problem we are finding the minimum distance from any given point..The travelling salesman problem has been trending for a long time now as there are multiple applications of this problem/The objective of the problem is to efficiently traverse the given path such that the path is of minimum cost.. We will make use of dynamic programming to solve the problem and reduce the complexity to an exponential form from the factorial form. Using dynamic programming greatly saves on time complexity and makes the program so much more efficient than before. its deployment pertaining to network programming. We go through all the paths possible and see which one of the paths is optimal for the given problem. What dynamic programming does is it stores a few cases which will be repeated multiple times in memory and uses callback to get the value back instead of computing the whole process again saving computational time and making the program a lot more efficient. The application that we are implementing is of a Printed circuit board and we check the minimum time required to visit all the holes once. We take a user input for the number of holes and the distance between them and give the best path possible along with the time taken and minimum distance. The complexity of the code has been reduced to (2^n)*(n^2) from n! which is the brute force method to solve the problem as it is an NP-Hard problem and a polynomial solution to the problem does not exist.

I. INTRODUCTION

The traveling salesman problem (TSP) is a famous mathematics problem that gives us the most efficient trajectory possible given a set of points and distances that must all be visited. The traveling salesman problem (TSP) is an algorithmic problem with the objective of finding the shortest route between a set of points and locations that must be visited. In the problem statement, the points are all the holes that must be visited much like all the cities that must be visited by a salesperson might .The goal is to keep both the cost of the path, the distance traveled and the time taken on the process as low as possible.

A.Problem Statement

The traveling salesman problem consists of a salesman and a set of locations that he must visit. The salesman has to visit each one of the locations starting from a particular starting point (e.g. the hometown) and returning to the same city. The challenge of the problem is that the traveling salesman wants to minimize the total length along with the cost of the trip and the time taken to complete the trip. We are implementing this problem using dynamic programming which reduces time complexity and runtime of the problem by a considerable factor. This concept is used and implemented for a Printed Circuit Board.

B.Objectives

The primary goal of our project is to apply the current traveling salesperson problem into the working of a Printed Circuit Board. The Printed Circuit Board is very similar to a travel salesperson as in a Printed Circuit Board electric signals pass through all the holes of the circuit.. Using dynamic programming we the shortest distance of the given graph is found which gives us the minimum cost to traverse it. Dynamic Programming also gives us the most efficient way of solving the problem. As the problem is a NP-Hard problem there is no polynomial solution to the given problem and the best possible solution to the problem

is (2^n)*(n^2) which is much faster compared to the brute force solution of n!.

II. LITERATURE SURVEY

http://ijirt.org/master/publishedpaper/IJIRT101672_PAPER.pdf

The given paper discusses the different algorithms possible for solving the traveling salesman problem. The different algorithms compared in the paper are the Greedy Algorithm, Greedy Algorithm and Dynamic Programing, As mentioned in the paper the best time complexity O(log N) is using the greedy algorithm but the solution accuracy is too low as no best choices are considered. Exponential time is achieved in genetic algorithm with time complexity O(Kmn) but there is an approximation of the solution and no optimal solution is reached. Dynamic programming may be more expensive in terms of time and memory but gives the most accurate answer.

After going through the paper and understanding the various factors involved in the different algorithms dynamic programming is the most accurate and efficient way of solving the travel salesman problem therefore we are implementing dynamic programming for the application of the project

A comparative study of Travelling Salesman Problem and solution using different algorithm design techniques

https://ieeexplore.ieee.org/document/7746316

The paper compares backtracking, branch and bound and dynamic programming and shows that dynamic programming has the best runtime.

Dynamic Programming Approaches for the Traveling Salesman Problem with Drone

Numerical experiments have shown that the approach taken in this paper can solve larger problems than the mathematical programming approaches that have been presented in the literature thus far.

TRAVELLING SALESMAN PROBLEMS BY DYNAMIC PROGRAMMING ALGORITHM

https://www.researchgate.net/publication/300038752_TRAVELLING_SALESMAN_PROBLEMS_BY_DYNAMIC_PROGRAMMING_ALGORITHM

This paper shows that the proposed method for an optimal tour is very simple, easy to understand and apply. From this method salesman can visit more cities at a time with minimum cost and minimum time taken.

https://onlinelibrary.wiley.com/doi/full/10.1002/net.21864
The paper show that restrictions on the number of operations can help significantly reduce the solution times while having relatively little impact on the overall solution quality.

Solving Dynamic Traveling Salesman Problem Using Dynamic Gaussian Process Regression
https://www.hindawi.com/journals/jam/2014/818529/

As a case of dynamic combinatorial optimization problem, extends the classical traveling salesman problem and finds many practical importance in real-world applications, traffic jams, network load-balance routing, transportation, telecommunications, and network designing. This paper gives an understanding on how to use the travel salesman problem for a practical application. The study produces a good optimal solution with less computational time in a dynamic environment.

III. DESIGN

The design of the problem is using dynamic programming as it is the best way to solve the travelling salesman problem. The reason normal recursion is not suitable is because when our salesman only had to visit four cities, we made six recursive calls. But now, we have literally *quadrupled* our tree of “potential paths”, which seems really, really, *really* bad. Solving TSP for five cities means that we need to make (n-1)! in this case four factorial recursive calls using the brute-force technique. As it turns out, 4! equals 24, which means we have to now make 24 recursive calls in order to accommodate just one additional city in our traveling salesman’s map.

Using Dynamic programming the values of previous used recursion calls are stored in memory and can be

invoked at any point of time reducing the computation time for that given part.

Naive Solution:

1) Consider point 1 as the starting and ending point.

2) Generate all (n-1)! Permutations of the holes/points.

3) Calculate cost of every possible permutation and keep track of minimum cost permutation.

4) Return the permutation with minimum cost at the end of the computation,.

Time Complexity: Θ(n!)

Dynamic Programming:

The set of vertices are {1, 2, 3, 4,….n}. Let us consider 1 as starting and ending point to calculate output. For every other vertex i (other than 1), we find the minimum cost path with 1 as the starting point, i as the ending point and all vertices appearing exactly once(important that repetition does not occur). Let the cost of this path be cost(i), the cost of corresponding Cycle would be cost(i) + dist(i, 1) where dist(i, 1) is the distance from point i to starting point 1. Finally, we return the minimum of all [cost(i) + dist(i, 1)] values. This is simple so far.

Now to calculate cost(i) using Dynamic Programming, we need to have some recursive relation in terms of sub-problems. Let us define a term *C(S, i) be the cost of the minimum cost path visiting each vertex in set S exactly once, starting at base address (1) and ending at i*.

We start with all subsets of size 2 and calculate C(S, i) for all subsets where S is the subset, then we calculate C(S, i) for all subsets S of size 3 and so on till n. Note that 1 must be present in every subset as it is the base.

If size of S is 2, then S must be {1, i},

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

Else if size of S is greater than 2.

C(S, i) = min { C(S-{i}, j) + dis(j, i)} where j belongs to S, j != i and j != 1.

For a set of size n, we consider n-2 subsets each of size n-1 such that all subsets don't have nth in them.

Thee above recurrence relation gives us the dynamic programming based solution. The maximum number of O(n*2n) subproblems, and each one takes linear time to solve. The total running time is therefore O(n2*2n). The time complexity is much less when compared to O(n!), but it is still exponential.

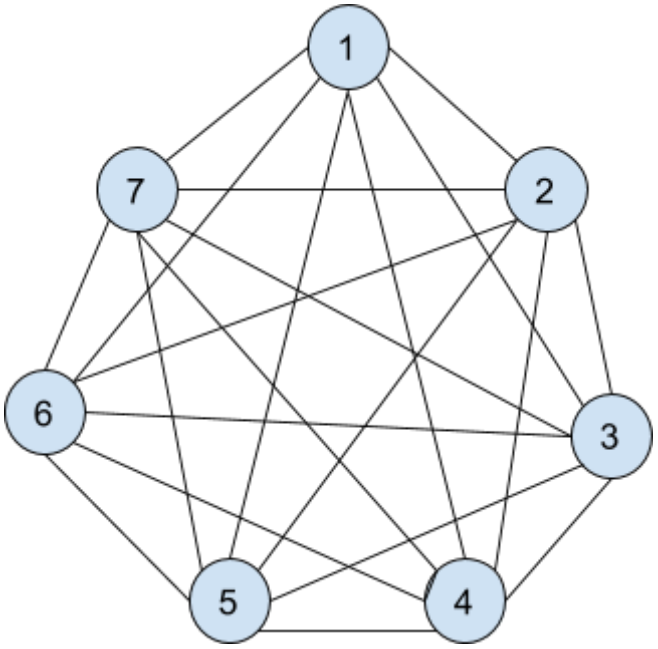
IV. METHODOLOGY

- As mentioned earlier, dynamic programming is the best way to solve this NP-Hard problem as it gives the optimal solution when compared to greedy algorithm and genetic algorithm.

Algorithm	Time	Complexity	Advantage	Disadvantage
Genetic Algorithm	Exponential time	O(Kmn)	Best solution using "fitness criteria" but no optimal solution	Approximation of solution is reached but not an optimal solution
Greedy Approach	5 seconds for 15 holes	O(logn)	Fastest but no solution accuracy	No best choices are considered and hence no accuracy is achieved
Dynamic Programming	9 seconds for 15 holes	O(n^2*2^n)	Global optimal solution and expensive for both memory and time	Expensive for both memory and time

- Dynamic programming gives us a better time complexity when compared to the brute force method which gives us n! factorial.
- Once the starting point is entered along with the weights of the paths an iteration is run to check the minimum distance along a particular path,

- This process is continued for all the different possibilities of nodes available
- At the end of every iteration the code check the current sum with the previous lowest sum saved in Fsum and replaces the value of the former with lowest sum if sum is lesser than the previous iteration
- This process continues till the end and every route is checked to get better results (the minimum path)



- The cost matrix for the above problem is :

node/	1	2	3	4	5	6	7
1	999	75	99	9	35	63	8
2	51	999	86	46	88	29	20
3	100	5	999	16	28	35	28
4	20	45	11	999	59	53	49
5	86	63	33	65	999	76	72
6	36	53	89	31	21	999	52
						9	
7	58	31	43	67	52	60	999

Here we enter the hole from which we want to start.After entering the first hole then we start going in all directions and try finding the path in which we get the less time.Initially we take the pathlength as 9999 and everytime we get the length of path less than pathlength then the pathlength gets updated.We have one more array fpath which stores the path with minimum time.If multiple paths take the minimum time then all the paths will be stored in the fpath array and all the minimum paths will be displayed.

```
--------------------WELCOME----------------------------

Please enter the number of holes to be drilled: 7

please enter the cost for 1 unit coating : 10.5

NOTE : in 1 unit time we can travel 1 unit distance

Please enter all the values of Adjacency Matrix:

enter the distance of all the 7 holes from hole 1
999 75 99 9 35 63 8

enter the distance of all the 7 holes from hole 2
51 999 86 46 88 29 20

enter the distance of all the 7 holes from hole 3
100 5 999 16 28 35 28

enter the distance of all the 7 holes from hole 4
20 45 11 999 59 53 49

enter the distance of all the 7 holes from hole 5
86 63 33 65 999 76 72

enter the distance of all the 7 holes from hole 6
36 53 89 31 21 999 52

enter the distance of all the 7 holes from hole 7
58 31 43 67 52 60 999

Please enter the hole number from which you wanted to start drilling : 1

the Minimum traveled distance is 158 units.

the Minimum time took to drill all the holes is 158 seconds.

the Minimum cost for coating = 1659.000000.

        path direction type 1: 1 --> 7 --> 2 --> 6 --> 5 --> 3 --> 4 --> 1.

Time taken by this program to drill all the holes and coating between the holes : 300.485000 s
```

- Output of the program:

----------WELCOME----------

Please enter the number of holes to be drilled: 7

please enter the cost for 1 unit coating : 10.5

NOTE : in 1 unit time we can travel 1 unit distance

Please enter all the values of Adjacency Matrix:

enter the distance of all the 7 holes from hole 1
999 75 99 9 35 63 8

enter the distance of all the 7 holes from hole 2
51 999 86 46 88 29 20

enter the distance of all the 7 holes from hole 3
100 5 999 16 28 35 28

enter the distance of all the 7 holes from hole 4
20 45 11 999 59 53 49

enter the distance of all the 7 holes from hole 5
86 63 33 65 999 76 72

enter the distance of all the 7 holes from hole 6
36 53 89 31 21 999 52

enter the distance of all the 7 holes from hole 7
58 31 43 67 52 60 999

Please enter the hole number from which you wanted to start drilling : 1

the Minimum traveled distance is 158 units.

the Minimum time took to drill all the holes is 158 seconds.

the Minimum cost for coating = 1659.000000.

path direction type 1: 1 --> 7 --> 2 --> 6 --> 5 --> 3 --> 4 --> 1.

Time taken by this program to drill all the holes and coating between the holes : 300.485000 s

V. IMPLEMENTATION

The code is written in the C programming language. We have coded the problem using dynamic programming. Here Dynamic Programing finds all the possible paths and finds the minimum path among them.If there multiple paths with minimum time then it displays all the paths with minimum cost. Here we input the cost adjacency matrix for all the holes and output will be the minimum time to drill the holes and cost in which those holes are drilled and all the minimum paths.

the program will be using the following functions:-

- clock() is used to start the clock.

```
start = clock();
```

- fpath array is used to store all the paths with minimum time which is found by path array.

```
int path[n-1];

int fpath[1000][n-1];
```

- TSP() function is used to find all the paths and find the minimum of all the paths and stores in the fpath array. If there are multiple paths then all the paths are stored in that.

```
TSP(C,A,path,fpath,&sum,&fsum,0,n,a-1,a-1,&sc);
```

- This is the function which prints all the minimum possible paths.

```
for(i=0;i<=sc;i++){

          printf("\n\n\tpath direction
type %d: %d -->",i+1,a);

        for(j=0;j<n-1;j++)

                              printf("  %d
-->",fpath[i][j]+1);

        printf(" %d.",a);

    }
```

- Here clock() is to count the final time.

```
end = clock();
```

- totaltime calculates the total time in which it got executed.

```
total_time    =    ((double)    (end    -
start))/CLOCKS_PER_SEC;
```

We will implement our idea by a well formed algorithm. First the clock will get started by using the clock() function.after that we enter all the values in the adjacency matrix which will be stored in a[][] array and we will initialize all the values in the array c[][] to zeros. Now the user will input the starting vertex from which we want to start drilling. The array c[][] finds all the possible paths and calculate the cost of that path and will be stored in the path array. Whenever we get the minimum cost path then all those paths will be stored in the fpath array and all those paths will be displayed. Here we find the cost of drilling all the holes and the minimum time in which we can drill all the holes and minimum distance travelled between the holes. At the end we again put the clock() function to find the time and by using those we will find the time in which the program got executed.

VI. RESULTS AND DISCUSSION

Please enter the hole number from which you wanted to start drilling : 1

the Minimum traveled distance is 158 units.

the Minimum time took to drill all the holes is 158 seconds.

the Minimum cost for coating = 1659.000000.

path direction type 1: 1 --> 7 --> 2 --> 6 --> 5 --> 3 --> 4 --> 1.

Time taken by this program to drill all the holes and coating between the holes : 300.485000 s

The output of the code gives us the minimum distance travelled, minimum cost for coating and the minimum time taken.
We achieve this result in O(2^n * n^2) which is much faster compared to the naive brute force solution and easier to compute for the system.

VII. CONCLUSION

We show that by using dynamic programming, we can solve larger problems than with the mathematical programming approaches(naive brute force) that have been presented in the literature so far. Moreover, we show that restrictions on the number of operations(reduction from n!) can help significantly reduce the solution times while having no impact on the overall solution quality.

While we have considered restricting the number of nodes per operation to reduce the running times of our algorithm, future research could investigate and

develop extensions of this approach. One promising direction for future research is to study different ways to reduce the complexity further with the same accuracy.

VIII. References

[1]http://ijirt.org/master/publishedpaper/IJIRT101672_PAPER.pdf

[2]https://www.researchgate.net/publication/300038752_TRAVELLING_SALESMAN_PROBLEMS_BY_DYNAMIC_PROGRAMMING_ALGORITHM

[3] https://ideas.repec.org/p/ems/eureri/101691.html

[4]https://www.hindawi.com/journals/jam/2014/818529/

[4]https://www.geeksforgeeks.org/dynamic-programming/

[5]https://onlinelibrary.wiley.com/doi/full/10.1002/net.21864

[6]https://ieeexplore.ieee.org/document/7746316

[7]https://ieeexplore.ieee.org/document/7938898

[8]https://www.researchgate.net/publication/272863824_Travelling_Salesman_Problem_using_Dynamic_Approach

[9]https://link.springer.com/article/10.1007/BF01185425

[10]https://ideas.repec.org/p/ems/eureri/101691.html

[11]https://www.semanticscholar.org/paper/Dynamic-programming-approaches-for-the-traveling-Bouman-Agatz/d35813d918793ea40209154d212ff9421308fd92