[1]CS575: Final Project Report

Project Title: Analysis of Traveling salesman problem (TSP) using four different methodologies

Team Members: Agrawal Shruti, Priyam Preeti, Rao Smriti

[1] CS575 (based on the IEEE conference template)

I. Problem

Given a set of n delivery spots and distance between each pair of the delivery spot, what is the shortest possible path such that the travel distance(cost) between each delivery spot is minimal in an attempt to get an optimal travel path with minimum travel distance and that each delivery spot is visited exactly once in the traversal.

II. Algorithms

A. Genetic Algorithm

This algorithm is a heuristic search algorithm inspired by the process that supports the evolution of life. The algorithms designed to replicate the natural selection process to carry generation.i.e. survival of the fittest of beings.
The steps involved in the algorithm are as following:
1. Initialize the population randomly.
2.Determine the fitness of the chromosome.
3. Until done repeat
    1. Select parents.
    2.Perform crossover and mutation.
    3. Calculate fitness of new population.
    4. Append it to the next generation.

B. Nearest Neighbor

This algorithm is a simple heuristic approach to solve the TSP problem. It was one of the first algorithms derived to solve this problem approximately and very easy to implement. The main idea behind this algorithm is to find the nearest shorter route between the cities using greedy approach until all the cities are visited. It relies on the local best decision to find the shortest route which might not be the optimal tour always. In the worst case, the algorithm results in a tour that is much longer than the optimal tour due to the greedy choices it makes at every step.

The algorithm can be summarized as below:

1. Initialize all vertices as unvisited.
2. Select any unvisited vertex v and set it as current vertex.
3. Mark v as visited.
4. Next find out the shortest edge connecting the current vertex v and an unvisited vertex u.
5. Set u as the current vertex. Mark u as visited.
6. If all the vertices in the given domain are visited, then terminate. Else, go to step 4.

C. *Dynamic Programming*

The main idea behind this algorithm is to compute the optimal solution for all the subpaths of length N while using information from already known optimal partial tours of length N-1. To compute the optimal solution for paths of length N, we need to remember (store) two things from each of the length (N-1) cases :

1. The set of visited nodes in the subpath
2. The index of the last visited node in the path

Together, these two form our dynamic programming state. There are N possible nodes that we could have visited last and 2^N possible subsets of visited nodes. Thus, space needed to store the answer to each subproblem is bounded by $O(N*2^N)$.To complete the TSP tour, we need to connect our tour back to the start node, loop over the end state in the dp_state table (which maintains the dynamic programming state) for every possible end position and minimize the lookup value plus the cost of going back to the start node.
The algorithm works as follows:

1. If the tour is completed, return the cost of going back to the start node from the end node
2. Lookup the dp_state table if the tour is already computed
3. If the next node has already been visited, skip it
4. The new next state is given by bitwise OR of the current state and the already visited next node left shifted by 1
5. The cost is calculated by looking up the adjacency matrix at position of start node and next node that was already visited added with a recursive call to the method with the arguments : the already visited next node, the new next node, the dp_state table and the table that holds the previous node of a node
6. If the new cost is less than the minimum cost, update the minimum cost to the new cost and save the value of the already visited next node.
7. Repeat steps 3 to 6 for all N nodes
8. Now, the prior node at position of start node and current state is the saved value of the already visited next node.
9. Save the value of minimum cost in the dp_state table at the position of the start node and the current state.
10. To find the tour, call the function (steps 1 to 9) to set up the dp_state table and the prior table holding the previous node of a node.
11. The optimal tour starts at the start node, if there are no more next nodes, all the cities have been visited, so exit the loop and now go back to the start node, else the next position can be found by looking up the prior table at the cell: previous position and current node.
12. The next state is found by a bitwise OR of the current state and left shift of next position by 1.
13. Set the current state as the next state and the position as the next position.
14. Repeat steps 11-13 till there are no next nodes left.
15. To calculate the cost, call the function(steps 1-9)

D. Backtracking Algorithm

In this algorithm, we recursively solve for the optimal path, building the solution in increments, removing those solutions that fail to satisfy the constraint for Travelling salesman problem, which is minimizing the cost of travelling from a starting city, visiting all the cities once and returning to the start city. Backtracking aims to optimize brute force approach. Brute force approach only takes into consideration the explicit constraints while backtracking considers implicit constraints and as implicit constraints are evaluated after every choice, an unfinished solution can be discarded if it does not satisfy the implicit constraints. The algorithm works as follows:

1. If you are at the last node , then compare the sum till now with the cost calculated added with the cost of going back to the start node from the end node, return the smaller value.
2. Now, we backtrack to find the remaining solutions and if we can find a tour with lesser cost, which will be the tour with optimum cost.
3. Finally, return the sum of that tour which gives the optimum cost.

III. Software Design and Implementation

A. Software Design :

a. Genetic Algorithm to solve Travelling Salesman Problem : The design is such that each city is considered as a genes.The string generated using these genes is called chromosome and for each chromosome fitness score is calculated(fitness score is the total distance route between all the cities mentioned).Further On each of these chromosome mutation is performed (a function to generate fittest chromosome).The fitness of newly generated chromosome is compared with previously generated chromosome and if found to have shorter path, they are passed to next generation.

b. Nearest Neighbour to solve Travelling Salesman Problem : The design of this algorithm is in such a way that, consider a reference vertex as start vertex and the distances between the cities are symmetric (i.e. the cost of travel from city X to city Y is exactly the same of that from Y to X). further, based on the greedy approach, we will find the shortest edge between any two unvisited cities. Repeat this process until each city is visited once, then we return to the start city. Now we should be able to calculate the minimal cost to reach all the cities by making local best choices.

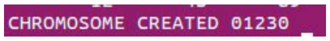
c. Dynamic Programming to solve the Travelling Salesman Problem: To apply dynamic programming , we first check whether Travelling Salesman Problem satisfies the Principle of Optimality. To find the optimal tour of N cities, we first have to find the optimal tour of (N-1) cities, which has to be a part of the optimal tour of N cities. This means we need to find the solution to the subproblems to find the solution to the whole problem. Thus, Travelling Salesman Problem satisfies the Principle of Optimality. In this approach, using bitwise manipulation, we generate the start node, end node, current state, next state and such.

B. Implementation and Tools Used -

a. Genetic Algorithm to solve TSP : The algorithm is implemented in Java. The tool used is visual studio code. The output is visualized on the console.

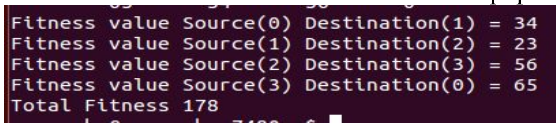
The software involves users inputting the number of cities and distance between each city. The implementation of the algorithm has various functions:

createChromosome(): This function creates String, which is route followed starting from city 0 and ending at city 0. For instance for city 0123 our chromosome looks like "01230"



```
CHROMOSOME CREATED 01230
```

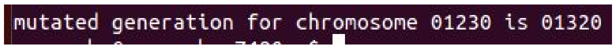
calculateFitness() : This function calculates the fitness(distance) for each chromosome in the population.i.e calculates the total distance for routes in the population set.



```
Fitness value Source(0) Destination(1) = 34
Fitness value Source(1) Destination(2) = 23
Fitness value Source(2) Destination(3) = 56
Fitness value Source(3) Destination(0) = 65
Total Fitness 178
```

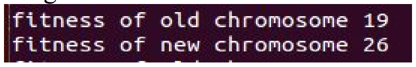
***mutatedGeneration**() : This function performs mutation on each chromosome of the population.Mutation is performed using a random generator function which generates a random number between 1 and V(total number of cities).*

NOTE: New fitness of mutated chromosome is calculated calculateFitness() itself.



```
mutated generation for chromosome 01230 is 01320
```

smallerThan() : This function compares the fitness calculated for the mutated chromosome with the fitness of the previous chromosome. The one with lesser fitness value is passed to the next generation.



```
fitness of old chromosome 19
fitness of new chromosome 26
```

b. Nearest Neighbour : The algorithm is implemented in Java. The tool used is visual studio code. The output is visualized on the console.

The software involves getting the number of cities and distance between the cities as input and the start city from the user and creates a distance matrix accordingly. The implementation of algorithm has the following methods:

TSPNearestNeighbour() : This method takes a distance matrix and start_node as an argument. The result matrix and the visited matrix keeps the track of the cities that should be visited in a sequential manner with respect to the distance between them.It starts with the start node that is specified by the user and then iteratively calls function findLowest to make a local or a greedy choice to the next node. Iteratively it tracks the nodes in the result matrix and the cost to reach all the nodes. This cost is minimal and same irrespective of whichever start node is selected.

findLowest() : This function takes the distance matrix, current element, and the visited matrix as the parameter. This function makes local or greedy choices for the current element to the nearest unvisited node. The initial minimum value is set to infinity and then it tracks all the distances from the current city to other cities and finds the minimum of all.

c. *Dynamic Programming : The algorithm has been implemented in Java. The output is visualised on the console.*

The software involves getting, as input, the number of cities from the user and the start node. The implementation has the following methods:

find_tsp() : This method takes the following arguments, the start node, the current state, the dp_state which is the dynamic programming state and the prior table which has the value of the previous node. It calculates the minimum cost and sets up the dynamic programming state on every recursive iteration. It returns the minimum cost of a path.

gettour() : This method calls the find_tsp function which sets the dp_state and prior tables. Using those two tables, the tour is calculated.

getcost() : This method calls the find_tsp method which returns the optimal cost.

d. *Backtracking : This algorithm has been implemented in Java. The output is visualised on the console.*

The software involves getting, as input, the number of cities from the user and the start node. The implementation has the following method :

tsp() : This method takes the following arguments, the node you are at now and the cost. If you are at the last node, that is, the length of the path is N, then compare the sum till now with the cost added with cost of going back to start node from the end node and return the smaller value. In order to backtrack, we keep track of which nodes are seen and which nodes are not.

ATTACHMENTS

1. HTTPS://GITHUB.COM/SMRITI0311/CS575.GIT
2. HTTPS://GITHUB.COM/SAGRAWA4/CS575-DESIGN-AND-ANALYSIS-OF-ALGORITHMS
3. HTTPS://GITHUB.COM/PPRIYAM1/CS575
4. https://youtu.be/Mq8nxsmDeBs
5. https://docs.google.com/presentation/d/1fmYUv1MMuv8RPNIgPx_zJUi1WqJpza4ubz27HB9pqEw/edit?usp=sharing

REFERENCES

[1] Abdulkarim, H. and Alshammari, I. (2015). Comparison of Algorithms for solving Travelling Salesman Problem. [online] Ijeat.org. Available at:https://www.ijeat.org/wp-content/uploads/papers/v4i6/F4173084615.pdf [Accessed 7 Feb. 2020].

[2] S Liu, "A Powerful Genetic Algorithm for Travelling Salesman Problem". Available at : https://arxiv.org/pdf/1402.4699.pdf [Accessed 12 Feb. 2020]

[3] Fiset, W., 2018. Travelling Salesman Problem | Dynamic Programming | Graph Theory. [online] YouTube. Available at: https://www.youtube.com/watch?v=cY4HiiFHO1o [Accessed 21 April 2020].

[4] NPTELHRD. 2010. Lec-24 Traveling Salesman Problem(TSP). [online] Available at: https://www.youtube.com/watch?v=-cLsEHP0qt0&list=PLP2Oz8vUR06U0SpPQetjjckhdtzt7Ihbw&index=8&t=0s [Accessed 21 April 2020].

[5] GeeksforGeeks. n.d. Travelling Salesman Problem Implementation Using Backtracking - Geeksforgeeks. [online] Available at: https://www.geeksforgeeks.org/travelling-salesman-problem-implementation-using-backtracking/ [Accessed 21 April 2020].

[6] GeeksforGeeks. n.d. Backtracking Algorithms - Geeksforgeeks. [online] Available at: https://www.geeksforgeeks.org/backtracking-algorithms/ [Accessed 21 April 2020].

[7] Stack Overflow. 2017. Differences Between Backtracking And Brute-Force Search. [online] Available at: https://stackoverflow.com/questions/44119627/differences-between-backtracking-and-brute-force-search

[8] Theseus.fi. 2013. Traveling Salesman Problem. [online] Available at: <https://www.theseus.fi/bitstream/handle/10024/59942/Jakub_Stencek_TSP_Bachelor_Thesis.pdf?sequence=1&isAllowed=y>

[9] En.wikipedia.org. 2020. Nearest Neighbour Algorithm. [online] Available at: https://en.wikipedia.org/wiki/Nearest_neighbour_algorithm

[10] GeeksforGeeks. n.d. *Traveling Salesman Problem Using Genetic Algorithm - Geeksforgeeks*. [online] Available at: <https://www.geeksforgeeks.org/traveling-salesman-problem-using-genetic-algorithm/> [Accessed 21 April 2020].