Solving Constraint Satisfaction Problem in TSP using GA and DFS algorithms

Md. Borhan Uddin
M.Sc. in Computer Science and Engineering
United International University
Dhaka, Bangladesh

Taki Uddin
M.Sc. in Computer Science and Engineering
United International University
Dhaka, Bangladesh

***Abstract*— Producing optimal tours in linear time through human subjects one of the most intensively studied problems in the area of optimization and the Traveling Salesman Problem (TSP) is a popular example of such optimization. The performance of TSP can be modeled by a graph, matrix, and different types of algorithms. The most frequently seen TSP problems are computer wiring, vehicle routing, job sequencing, combinatorial data analysis. The optimization problem of TSP is to find the salesman's route from a specific location and back, where each city will be visited once. If there are fewer cities then it is easy to solve and the complexity increases exponentially as the number of cities increases. The total distance and cost of other resources incurred should all be minimum. Our paper addresses the Constraint Satisfaction Problem (CSP) in TSP using Depth-First Search (DFS) and Genetic Algorithm (GA) to find a better solution and comparing the efficiency of both algorithms. Our result shows the GA performs better at solving TSP compared to DFS.**

Keywords—genetic algorithm, depth-first search, constraint satisfaction problem, traveling salesman problem, ant colony optimization

I. Introduction

CSP is defined with a set of variables that satisfy some conditions or constraints of our problem. It represents variables, conditions, and values or domains. In simple terms, it requires solutions under certain constraints.

Some of the problems such as shortest path algorithm, minimum spanning tree, depth-first search optimal stopping algorithm, genetic algorithm, etc. generalize the performance of TSP problem where optimization tasks like perception, decision making are important. It is a problem where the distance between cities and the position of cities is accurately known and the problem solver determines the nearly optimal route.

The DFS is a type of search algorithm that uses stacks to locate all the nodes. The method of DFS to find unvisited nodes simplifies the discoverability of best routes. We use this algorithm as our search algorithm using the data structure stack and backtracking method to evaluate each vertex that has not been visited. It is used to check each path to find the longest path in a graph.

The genetic algorithm belongs to the evolutionary algorithm class. It starts with different problem solutions that are encoded into the population, a fitness function is applied to determine the fitness of each generation, and after that, a new generation is generated via the selection, crossover, and mutation process. An optimal solution will be obtained after the termination of the genetic algorithm. If the completion condition is not met, the algorithm will proceed with the particular host.

We chose these two algorithms as DFS has low time complexity as it just finds the most efficient route from a given set of nodes and on the other hand GA creates multiple iterations and patterns from the given nodes to find the most efficient solution for TSP.

II. Related Works

Scott Graham in 2000 modeled an algorithm [22] and solved the visual version of TSP on human problem solving as the first study where the optimal tours were shorter than the subject's tour in "The traveling salesman problem: A hierarchical model". Here, the problem was solved producing a sequence of search and top-down steps by which the computational complexity made very low. Ivan Brezina in 2011 discussed a problem named ant colony optimization [27] which showed the techniques of group evolution in "Solving the Travelling Salesman Problem using Ant Colony Optimization" to find out the approach in the application.

Sheila Eka Putri in 2011 published a paper [6] to determine the longest path in a given list of nodes based on the size of the list.

A web application in 2012 named "The Project Spot" solved the TSP using a method of elitism and tournament [35]. The elitism method is for checking fitness and the tournament saving each tour/route.

These last two related topics inspired us to find algorithms to solve TSP and compare performance between the shortest path in the fastest time and the best tour/route in a finer accuracy.

III. Traveling Salesman Problem

The TSP is one such problem example of a constraint satisfaction problem which has no exact algorithm for solving. The solving purpose of this problem is to get an optimal solution in minimal time. It can be represented through undirected labeled graphs such that the vertices are the cities, edges will be the path and edge length will be the distance between cities. The problem will be minimized when it will specify the first and last vertex and will visit each vertex only for once. When there will be no path existing to be explored between two cities, a lengthy arbitrary edge will represent a complete graph without affecting the optimal route.

In our traveling salesman problem, we denote a set of cities, c as $\{c_1, c_2, c_3, \ldots c_n\}$ where n is the number of cities. All the permutations, p of the tours is denoted as $\{p_1, p_2, p_3, \ldots p_n!\}$. Our objective is to choose a permutation of the route between source, s, and destination, c which is most cost-effective.

In this paper, we utilize two algorithms to solve the traveling salesman problem. The first one is a depth-first search and the other one is a genetic algorithm. The link to our code both of the algorithms hosted on *Colaboratory* which can be found here.

IV. TSP USING DEPTH FIRST SEARCH

Using DFS, we traverse the nodes to find all paths for a given source and destination. In doing so, we stored the explored path in an array and marked each vertex that has been explored. In this way, we find all the possible paths from least expensive to most expensive for a traveling salesman to go from source to destination.

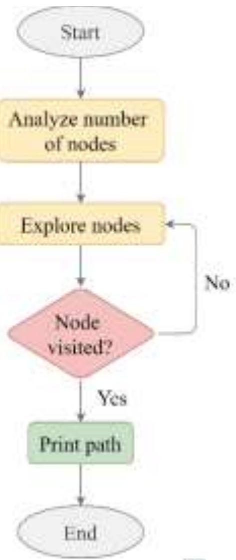

Fig. 1. Flowchart for all possible path search in DFS

The depth-first search algorithm for our traveling salesman problem is described in the flowchart in the figure. 1. We start with taking source and destination as input. The unexplored or unvisited nodes are analyzed and explored recursively. All the possible nodes are analyzed and paths generated are stored or printed. The output of the flowchart is the possible paths for our traveling salesman.

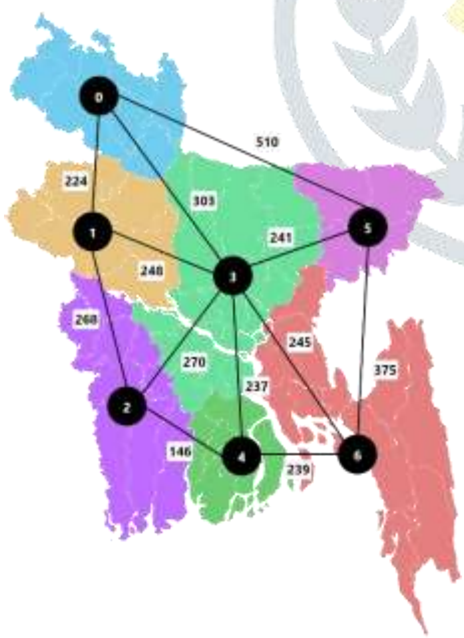

Fig. 2. Graph vertices and edge values (cost) in a map

The figure above represents the seven major cities of Bangladesh as vertices and the distance as edges. We represented the distance between the vertices as a weighted cost, that a traveling salesman could incur when traveling from a given source to destination. All printed all the possible paths between source and destination. These paths span from the shortest path (least expensive) to the longest path (most expensive).

The city nodes, c is defined in a graph array where each node connection is also defined. The source and destination nodes are defined as u and v respectively. We stored the permutations, p as a path array.

TABLE I. PSEUDOCODE FOR FINDING ALL POSSIBLE PATHS

```
Procedure dfs()
graph[]
input source, destination
class Graph
   initialize()
   addEdge()
   printAllPathsUtil()
      if visited == true then
         add to path[]
      end
      if source == destination then
         print path_array
      end
      if visited ==  false then
         printAllPathsUtil()
      end
   printPath()
      visited == false
      printAllPathsUtil()
```

V. GENETIC ALGORITHM FOR TSP

The genetic algorithm is influenced through a mechanism that facilitates the origin of species which is also considered as heuristic search. It is intended to replicate the process of natural selection to generate such as the existence of the fittest of creatures. The algorithm follows the steps which are the emergence of population creation, fitness calculation, best genes selection, crossing over, the mutation for introducing diversity. Different types of the optimization problem can be solved with the implementation of this algorithm where traveling sales problem is one of them. The shortest route needs to find by a salesman in a given city where the person will visit every city for once and after that, he will come back to the starting city.

In our algorithm, cities are considered as genes, and the chromosome is defined based on the characteristics of string generation. The cities path length is equal to the fitness score for targeting the population. The path length of a gene is defined as a fitness score which will work basis on the fitter length of the gene where the path is lesser. In the gene pool, the fittest of genes survive for the population evaluation and pass to the next iteration where the iteration varies upon the cooling variable. The cooling variable's value decreases matching with iteration and after certain times of iterations, it reaches the threshold.

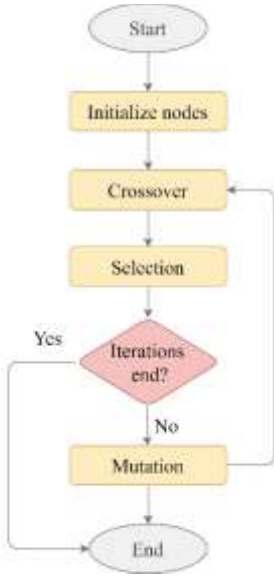

Fig. 3. Flowchart for best path search in GA

The traveling salesman problem is described with a genetic algorithm while solving our algorithm where the population is initialized randomly and the fitness of the chromosome is determined. Parents selection, crossover and mutation execution, new population's fitness calculation, and the appendication to gene pool need to run repeatedly while getting the outcome.

The function *gene_mutate()* creates two-parent genes using the crossover method. If we consider that we have 3 cities [1, 2, 3]. And the salesman wants to travel all the cities starting from city1 through the shortest route and back to city1 [1, 2, 3, 1]. Taking this as a list of chromosomes and mutating it but keeping the start and endpoint unchanged, gives us a mutated child chromosome. The parameters we used are *v*, *genes* to denote the size of our graphs, and the names of our nodes respectively. The variables *start* and *pop_size* defines the source of the traveling salesman route and the desired population size (possible routes). A variable *gen_thres* sets the number of iterations we would like to run for our program. The *TSPUtil()* function runs iteratively (*gen_thres*) to find all the possible routes and their fitness from the list of given *genes*.

TABLE II. PSEUDOCODE FOR FINDING ALL POSSIBLE PATHS

```
Procedure ga()
population[][]
   random_number()
   repeat()
      if character == string then
         return true
      return false
   gene_mutate()
      if parent1!= parent2 then
         return gnome
   create_gnome()
   fitness_calculate()
      map[][]
         return fitness
   cooldown()
   tsp_util()
```

VI. RESULTS AND OUTCOME

Our result for the DFS algorithm shows, the shortest possible paths for TSP. We set an environment for our algorithm providing the number of nodes and their connected edges. The algorithm takes source *s* and destination *d* as the input parameters. Once the code is run the pseudo code from TABLE I is executed. For initialization, our code takes the graph of seven divisions of Bangladesh denoted by numbers [0, 1, 2, 3, 4, 5, 6]. Now, for instance, if we provide an input of $s = 3$ and $d = 5$, the output with the pseudo-code gives us all the possible paths.

Fig. 4. The output of the DFS algorithm

With our pseudocode, we get the shortest and most efficient path for the given source and destination which is [3, 5] as shown in Fig. 4 which is also supported by our cost distribution from Fig. 2.

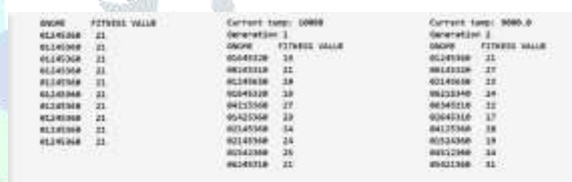

Fig. 5. The output of GA for TSP

On the other hand, our GA shows the best possible order of nodes with the same graph. Our population size is defined randomly, with each iteration a better value of path is found for the TSP.

In GA, we specify the number of cities $v = 7$, the population size *pop_size* = 10 and the number of iteration gen_thres = 2. On running the program, the pseudo-code from TABLE II is executed. Due to the *gen_thres* parameter set to 2, our output gives us two iterations of 10 (*pop_size*) new best paths for TSP. But due to the randomness as mentioned earlier, the iterative path cannot be supported by the graph in Fig. 2.

TABLE III. COMPARISON FOR TSP: DFS AND GA WITH ANT COLONY OPTIMIZATION

Algorithm	Parameter	Function	Process	Results	Remarks
Depth First Search	Source, destination	addEdge(), printAllPaths(), printAllPathsUtil()	Analyze number of nodes in an array, explore array size (nodes), record visited node or keep exploring unvisited node	All possible paths of nodes	Source and destination can't be the same
Genetic Algorithm	Number of nodes, start, iteraion	TSPUtil(), cooldown(), lessthan(), cal_fitness(), create_gnome(), mutatedGene(), repeat()	Initialize nodes, crossover between sets of parents, selection of child node, check if number of iterations is completed, record selected nodes or undergo mutation and crossover again	Source to destination (same) possible combination of nodes	The weight of the graph isn't established
Ant Colony Optimization	Source, destination	AntSolutionConstruction(), fitnessTest(), pheromoneUpdate(), repeat()	Initialize nodes, randomize path order, fitness calculation, update pheromone, run iteraion	Source to destination possible	Weight and graph isn't defined

	combination of nodes

VII. Conclusion and Future Scope

In the DFS algorithm, our limitations were that the source and destination were not the same. These led to partial solving of TSP and hence no fitness scale. Though our algorithm still found the shortest route from a given source to destination.

Crossing over and mutation methods are used which helps to find out a good solution for TSP in GA. This approach is helpful for few models like the vehicle navigation model, networking model task scheduling model, cellular network frequency allocation. In our DFS we were able to denote a graph of cities to find the shortest route. However, in our algorithm (GA) no such graph was established.

For our future scope, a merging between both the algorithms to find the best shortest path (DFS) from the fittest order of nodes (GA).

Acknowledgment

We are thankful to Dr. Swakkhar Shatabda, Associate Professor, United International University for his helpful feedback on the paper and suggestions for improvements.

References

[1] Haxhimusa, Yll & Carpenter, Edward & Catrambone, Joseph & Foldes, David & Stefanov, Emil & Arns, Laura & Pizlo, Zygmunt. (2011). 2D and 3D Traveling Salesman Problem. The Journal of Problem Solving. 3. 10.7771/1932-6246.1096.

[2] Gupta, Saloni & Panwar, Poonam. (2013). Solving Travelling Salesman Problem Using Genetic Algorithm. International Journal of Advanced Research in Computer Science and Software Engineering. 3. 376-380.

[3] Muneeb Abid, Malik & Iqbal, Muhammad. (2015). Heuristic Approaches to Solve Traveling Salesman Problem. TELKOMNIKA Indonesian Journal of Electrical Engineering. 15. 390-396. 10.11591/telkomnika.v15i2.8301.

[4] Gharote, Mangesh & Thomas, Dilys & Lodha, Sachin. (2015). Travelling Salesman Problem (TSP) - Presentation.

[5] Mijwil, Maad. (2016). Traveling Salesman Problem Mathematical Description.

[6] Putri, Sheila Eka & Tulus, Tulus & Napitupulu, Normalina. (2011). Implementation and Analysis of Depth-First Search (DFS) Algorithm for Finding The Longest Path. 10.13140/2.1.2878.2721.

[7] Mastan, Shaik & Balakrishnan, U. & Raju, G.. (2019). A Quick Heuristic Algorithm for Traveling Salesman Problem. SSRN Electronic Journal. 10.2139/ssrn.3497489.

[8] Weyland, Dennis & Bianchi, Leonora & Gambardella, Luca Maria. (2009). New Heuristics for the Probabilistic Traveling Salesman Problem.

[9] Maha, Al-Furhud & Ahmed, Zakir. (2020). Genetic Algorithms for the Multiple Travelling Salesman Problem. International Journal of Advanced Computer Science and Applications. 11. 553-560. 10.14569/IJACSA.2020.0110768.

[10] Burkard, Rainer & Deineko, Vladimir & Woeginger, Gerhard. (1998). The Travelling Salesman Problem on Permuted Monge Matrices. Journal of Combinatorial Optimization. 2. 333-350. 10.1023/A:1009768317347.

[11] Bang, Jeongho & Yoo, Seokwon & Lim, James & Ryu, Junghee & Lee, Changhyoup & Lee, Jinhyoung. (2010). A Quantum Heuristic Algorithm for the Traveling Salesman Problem. Journal- Korean Physical Society. 61. 10.3938/jkps.61.1944.

[12] Mastan, Shaik. (2019). Heuristics Designed For the Traveling Salesman Problem. 7. 209-213.

[13] Putponpai, Jutathip & Pongchairerks, Pisut. (2014). A Problem-Based Heuristic for Asymmetric Travelling Salesman Problems. Journal of Industrial and Intelligent Information. 3. 10.12720/jiii.3.3.258-265.

[14] Brest, Janez & Zerovnik, Janez. (2005). A heuristic for the asymmetric traveling salesman problem.

[15] Gutin, Gregory & Zverovitch, Alexei & Blokh, David. (2008). A Heuristic for the Resource-Constrained Traveling Salesman Problem.

[16] Pankratova, Yaroslavna & Tarashnina, Svetlana. (2017). On A Dynamic Traveling Salesman Problem.

[17] Flip, Phillips & Thomas, O'Connell & Oliver, Layton. (2010). The Traveling Salesman Problem in the Natural Environment. Journal of Vision. 9. 10.1038/npre.2010.4960.1.

[18] Li, Jun & Sun, Qirui & Zhou, Mengchu & Xiaolong, yu. (2014). Colored Traveling Salesman Problem and Solution. IFAC Proceedings Volumes (IFAC-PapersOnline). 19.

[19] Alkaya, Ali & Duman, Ekrem. (2010). A new generalization of the Traveling salesman problem. Applied and Computational Mathematics. 9. 162-175.

[20] Gutin, Gregory & Punnen, Abraham & Barvinok, Alexander & Gimadi, E. & Serdyukov, Anatoliy. (2001). The Traveling Salesman Problem and Its Variations.

[21] Sánchez, Sergio & Cocho, Germinal & Flores, Jorge & Gershenson, Carlos & Iñiguez, Gerardo & Pineda, Carlos. (2017). Trajectory Stability in the Traveling Salesman Problem. Complexity. 2018. 10.1155/2018/2826082.

[22] Graham, Scott & Joshi, Anupam & Pizlo, Zygmunt. (2000). The traveling salesman problem: A hierarchical model. Memory & cognition. 28. 1191-204. 10.3758/BF03211820.

[23] Pekár, Juraj & Brezina, Ivan & Kultan, Jaroslav & Ushakova, Iryna & Dorokhov, Oleksandr. (2020). Computer tools for solving the traveling salesman problem. Development Management. 18. 25-39. 10.21511/dm.18(1).2020.03.

[24] Dinh, Lau & Chien, Tran. (2015). Traveling Salesman Problem in Distributed Environment. Computer Science & Information Technology. 5. 19-28. 10.5121/csit.2015.51503.

[25] Hu, Bin & Raidl, Gü & R, nther. (2008). Solving the Railway Traveling Salesman Problem via a Transformation into the Classical Traveling Salesman Problem. Proceedings - 8th International Conference on Hybrid Intelligent Systems, HIS 2008. 73-77. 10.1109/HIS.2008.30.

[26] Lipowski, Adam & Lipowska, Dorota. (2005). Traveling salesman problem with a center. Physical review. E, Statistical, nonlinear, and soft matter physics. 71. 067701. 10.1103/PhysRevE.71.067701.

[27] Brezina, Ivan & Čičková, Zuzana. (2011). Solving the Travelling Salesman Problem Using the Ant Colony Optimization. International Scientific Journal of Management Information Systems. 6.

[28] Huiru, Ma & Jia, Limin & Xingchen, Zhang & Jianrui, Miao & Jiandong, Sun. (2015). Travelling Salesman Problem in Uncertain Environments. The Open Cybernetics & Systemics Journal. 9. 313-317. 10.2174/1874110X01509010313.

[29] Gharote, Mangesh & Thomas, Dilys & Lodha, Sachin. (2012). Excel Solvers for the Traveling Salesman Problem.

[30] Silberholz, John & Raiconi, Andrea & Cerulli, Raffaele & Gentili, Monica & Golden, Bruce & Chen, S.. (2013). Comparison of heuristics for the colorful traveling salesman problem. International Journal of Metaheuristics. 2. 141-173. 10.1504/IJMHEUR.2013.054143.

[31] Fortini, Matteo. (2020). LP-based heuristics for the Traveling Salesman Problem.

[32] Váňa, Petr & Faigl, Jan. (2015). On the Dubins Traveling Salesman Problem with Neighborhoods. 4029-4034. 10.1109/IROS.2015.7353945.

[33] Saad, Shakila & Wan Jaafar, Wan Nurhadani & Jamil, Siti. (2013). Solving Standard Traveling Salesman Problem and Multiple Traveling Salesman Problem by Using Branch-and-Bound. AIP Conference Proceedings. 1522. 1406-1411. 10.1063/1.4801294.

[34] Martonák, Roman & Santoro, Giuseppe & Tosatti, Erio. (2004). Quantum annealing of the Traveling Salesman Problem. Physical review. E, Statistical, nonlinear, and soft matter physics. 70. 057701. 10.1103/PhysRevE.70.057701.

[35] Lee Jacobson (2012) Applying a genetic algorithm to the traveling salesman problem. Retrieved from The Project Spot: http://www.theprojectspot.com/tutorial-post/applying-a-genetic-algorithm-to-the-travelling-salesman-problem/5

[36] Whitley, Darrell & Ochoa, Gabriela. (2011). Partial neighborhoods of the Traveling Salesman Problem. Genetic and Evolutionary Computation Conference, GECCO'11. 529-536. 10.1145/2001576.2001649.

[37] Michail, Othon & Spirakis, Paul. (2014). Traveling Salesman Problems in Temporal Graphs. Theoretical Computer Science. 553-564. 10.1007/978-3-662-44465-8_47.

[38] Hameed, Ibrahim. (2020). Multi-objective Solution of Traveling Salesman Problem with Time. 10.1007/978-3-030-14118-9_13.

[39] Barach, Gilad & Fort, Hugo & Mehlman, Yoni & Zypman, Fredy. (2020). Information in the Traveling Salesman Problem.
[40] Sangwan, Shabnam. (2018). Literature Review on Traveling Salesman Problem. International Journal of Research. 5. 1152.
[41] Shahab, M. (2019). New heuristic algorithm for traveling salesman problem. Journal of Physics: Conference Series. 1218. 012038. 10.1088/1742-6596/1218/1/012038.
[42] Mohanta, Susanta. (2019). On Optimal Solution for Traveling Salesman Problem: Direct Approach. 14. 3226-3230.
[43] Barach, Gilad & Fort, Hugo & Mehlman, Yoni & Zypman, Fredy. (2020). Information in the Traveling Salesman Problem.
[44] Hussain, Abid & Muhammad, Yousaf shad & Sajid, Nauman & Hussain, Ijaz & Shoukry, Alaa & Gani, Showkat. (2017). Genetic Algorithm for Traveling Salesman Problem with Modified Cycle Crossover Operator. Computational Intelligence and Neuroscience. 2017. 10.1155/2017/7430125.
[45] Önder, Ilter. (2007). A Genetic Algorithm for TSP with Backhauls based on Conventional Heuristics. 10.13140/RG.2.2.19401.80481.
[46] Rexhepi, Avni & Maxhuni, Adnan & Dika, Agni. (2013).
[47] Analysis of the impact of parameter values on the Genetic Algorithm for TSP. International Journal of Computer Science Issues. Volume 10. pp 158-164.
[48] Chudasama, Chetan & Shah, S.M. & Panchal, Mahesh. (2011). Comparison of parents selection methods of genetic algorithm for TSP. Proc. of the Int. Conf. on Computer Communication and Networks CSI-COMNET-2011. 1. 102-105.
[49] Pratik Basu (2020) Introduction to Ant Colony Optimization. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/introduction-to-ant-colony-optimization/
[50] Blum, Christian. (2007). Ant Colony Optimization: Introduction and Hybridizations. Proceedings - 7th International Conference on Hybrid Intelligent Systems, HIS 2007. 24 - 29. 10.1109/HIS.2007.36.