FINDING OPTIMAL SOLUTION FOR TRAVELLING SALESMAN PROBLEM

Team members :

- Chandra Teja Kommineni
- Rajesh Karumachi
- Sai Vineeth Kaza

Git-hub Link: https://github.com/Rajesh-karumanchi/Algorithm-project
Presentation Link: https://www.youtube.com/watch?v=j2SJ7QKgadE

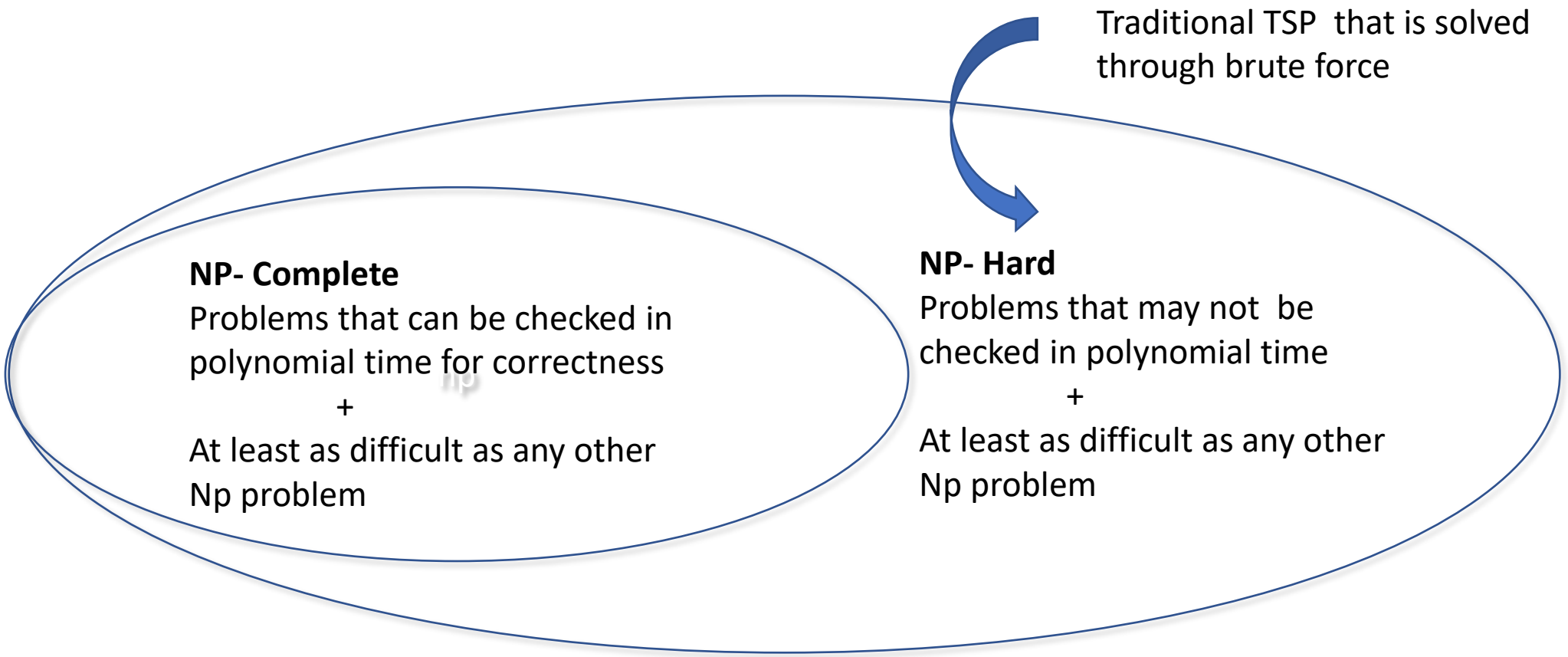
Travelling Salesman Problem(TSP)

- Travelling Salesman Problem(TSP) is the problem where we have to find the optimal path for a salesman to visit the given list of cities and distances between them by starting from a randomly selected starting city and returning to the same starting city.

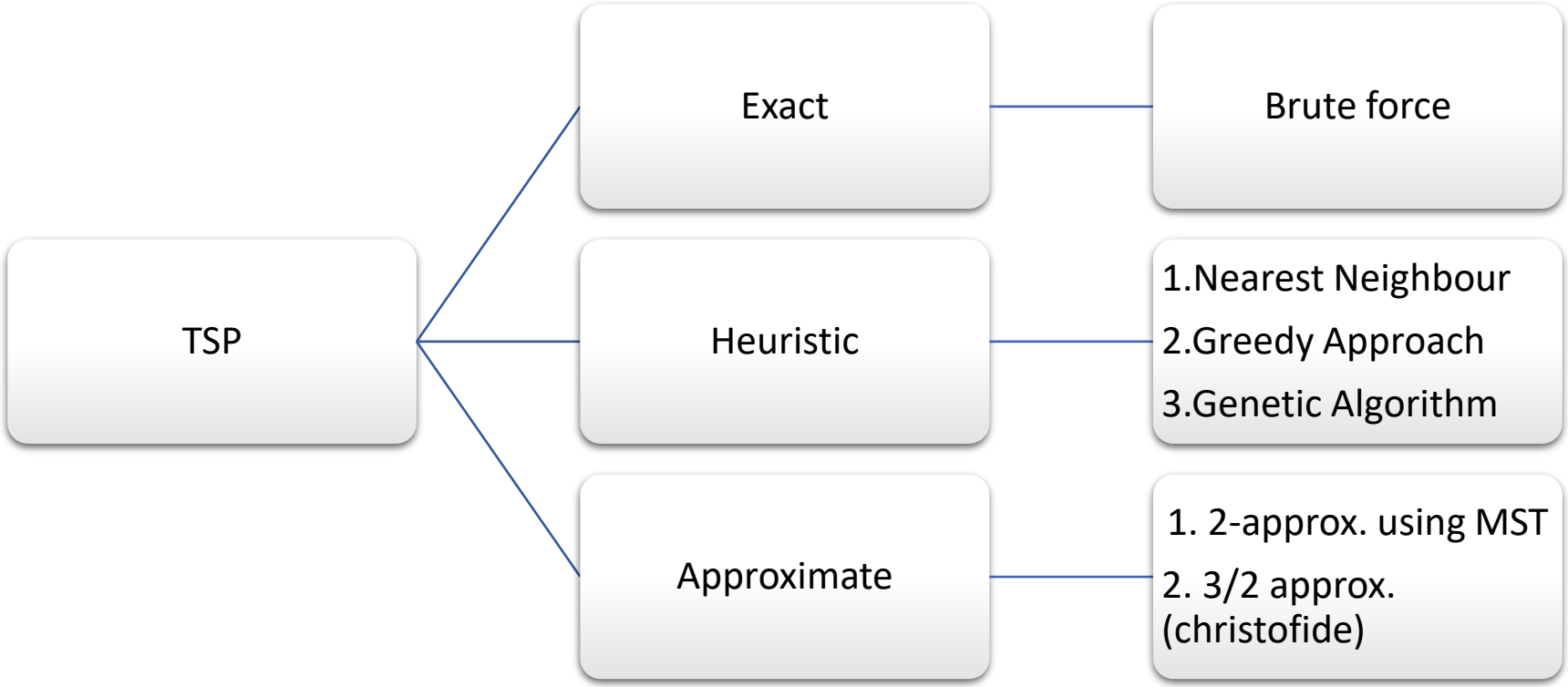
- As there are a lot of routes and as we have the find the shortest/ the most efficient route, this makes it the most challenging computational problem. It is so easy to describe the problem, yet it is so difficult to solve the problem.

- As, there is no perfect or exact solution available for TSP, the question we are wondering about is which algorithm can be as close to an efficient, effective, and optimal solution for the Travelling Salesman Problem?

TSP is NP-Hard

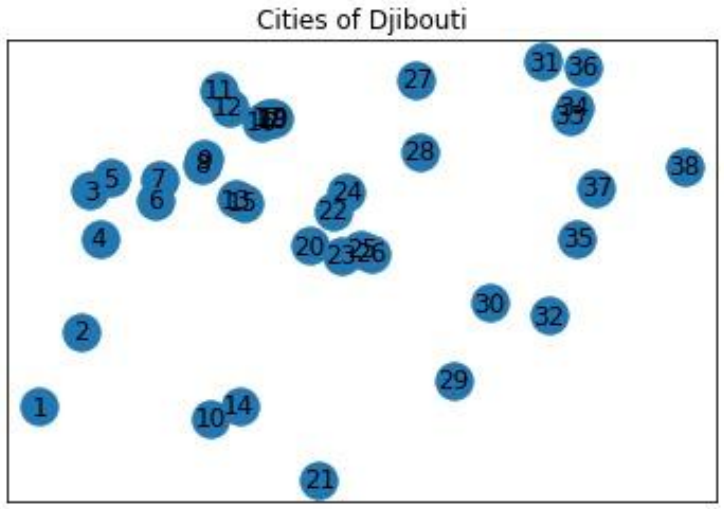


Algorithms chosen to solve TSP



	Time complexity	Solution	Input size
Exact	High	Optimal	Low
Heuristic	Low	Varies based on input	High
Approximate	Low	α * Optimal	High

Dataset



- To validate our results on real-world problems, we used data from http://www.math.uwaterloo.ca/tsp/world/countries.html. It contains TSPLIB data points for cities in 25 countries.
- These TSPLIBs were derived from the National Imagery and Mapping Agency's database of geographical feature names. TSPLIB is a file format that uses the.tsp extension.
- The .txt view contains the file name, description of the data source and the 'X' and 'Y' coordinates of all city nodes.
- The Euclidean distance between these coordinates is the distance between each city.
- We conducted our experiments on Djibouti, which have 38 cities, respectively.
- We generate a Euclidean distance matrix using the 'X' and 'Y' coordinates of city nodes, which explains the distance from one city to all others.

NAÏVE

Naïve approach is the basic brute force approach of the algorithm where we calculate the minimum distance by calculating all the possible permutations of the paths and find the shortest unique solution. In this approach we first list the total number of paths possible. We will then calculate the distance of required by each path to travel through all the cities and return back to the starting city. Finally, we will pick the minimum distance path as the shortest and optimal solution for TSP.

Pseudocode:

1. Consider city 1 as the starting node and ending node.
2. All possible path permutations between the cities which will be (n-1)! are generated.
3. Calculate the distance of each path and return the minimum distance as the optimal path.

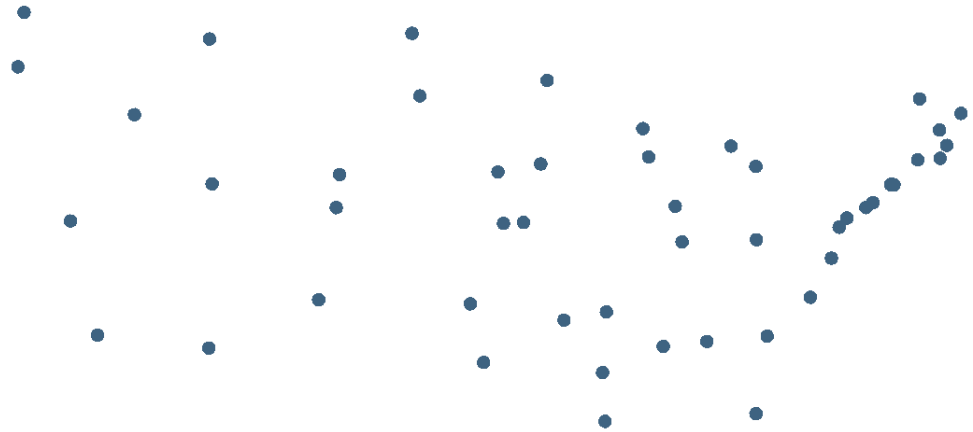
Time complexity of the algorithm is O(n!) as it is generating all the possible path permutations from a starting node using the formula (n-1)! and iterating it over all the available cities as the starting points i.e., O(n*(n-1)!) = O(n!).

- **GREEDY**

- Greedy approach is a heuristic algorithm which is better and optimized version of naïve approach. We can say that greedy approach is finding the shortest Hamilton cycle from all the possible cycles. Hamilton cycle is forming a cycle by visiting all the nodes of the graph only once. As stated in the name of the approach 'greedy', we use greedy choices to optimize the algorithm for TSP. This approach finds the local optima for the problem and then optimizes it to form the global optima solution for the problem. So, for our current TSP problem at each step of the journey we update the final distance with the minimum obtained distance and return the path.

- **Pseudocode:**
  - Consider city 1 as the starting node and ending node.
  - Sort all the available edges of the graph.
  - Create a list to track the visited nodes.
  - Create a list to track the edges of the tour and add the minimum distance edge to the tour at each and every step.
  - If all the nodes are visited, we will end the algorithm and return the tour with the shortest path.

- The time complexity of greedy approach algorithm is O(n^2 * logn).



Nearest Neighbour Algorithm:

- The idea here is to start with a random city and visit nearby ones repeatedly until we have visited all of them. In other words, given a set of cities, this algorithm joins their nearest neighbors. This heuristic is also a greedy algorithm variant

- Step-1 : Choose a starting vertex and make it the current node.
- Step-2: Examine all of the other nodes that have not yet been selected and select the closest one.
- Step-3: Repeat step 2 until all nodes have been selected at least once.
- Step-4: Return to the starting vertex if all nodes are selected.
- Step-5: Draw and print the path, then calculate the distance.

- Runtime: We check a vertex ‘I’ with rest ‘n-I’ vertices. It tajes O(n^2). Printing path and calculating distance takes linear time. So time complexity is O(n^2)

2- Approximation with MST:

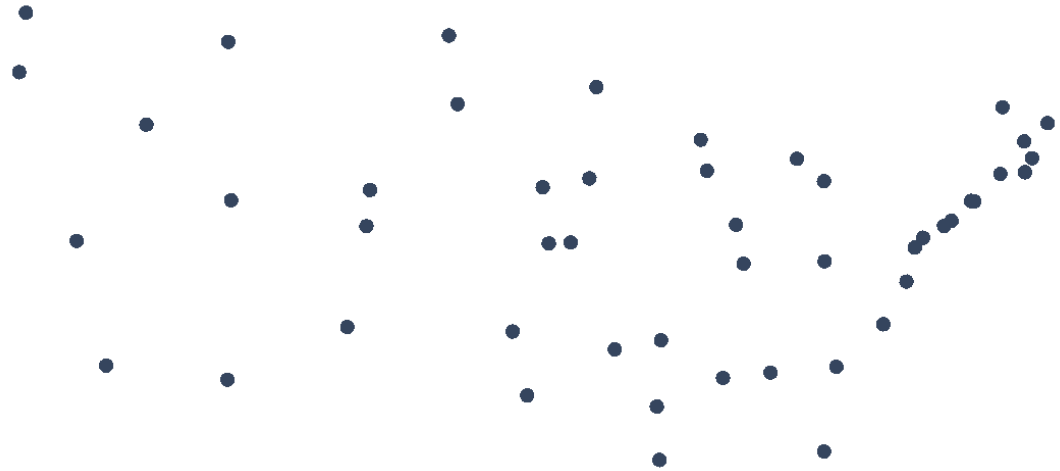
- 2- The approximation algorithm performs well if the set of given nodes adheres to the triangle inequality. The Traingle inequality states that the distance between vertex 'A' and vertex 'B' is shortest if we travel directly from 'A' to 'B' rather than via some intermediate vertex 'C'.
- Distance(A,B) <= Distance(A,C) + Distance(C,B)
- This algorithm employs Prim's MST and Triangle inequality concepts to find a path that is always less than twice the cost of optimal path.

Algorithm:

- Step-1. Choose a random node as the starting and ending point.
- Step-2: Create a minimum spanning tree with the start point as the root node using Prim's algorithm.
- Step-3: For path consider the path in which vertices are visited in Depth First Search order of the minimum spanning tree obtained and add starting vertex at the end.

Time complexity: The time complexity of this algorithm is O(n^2).

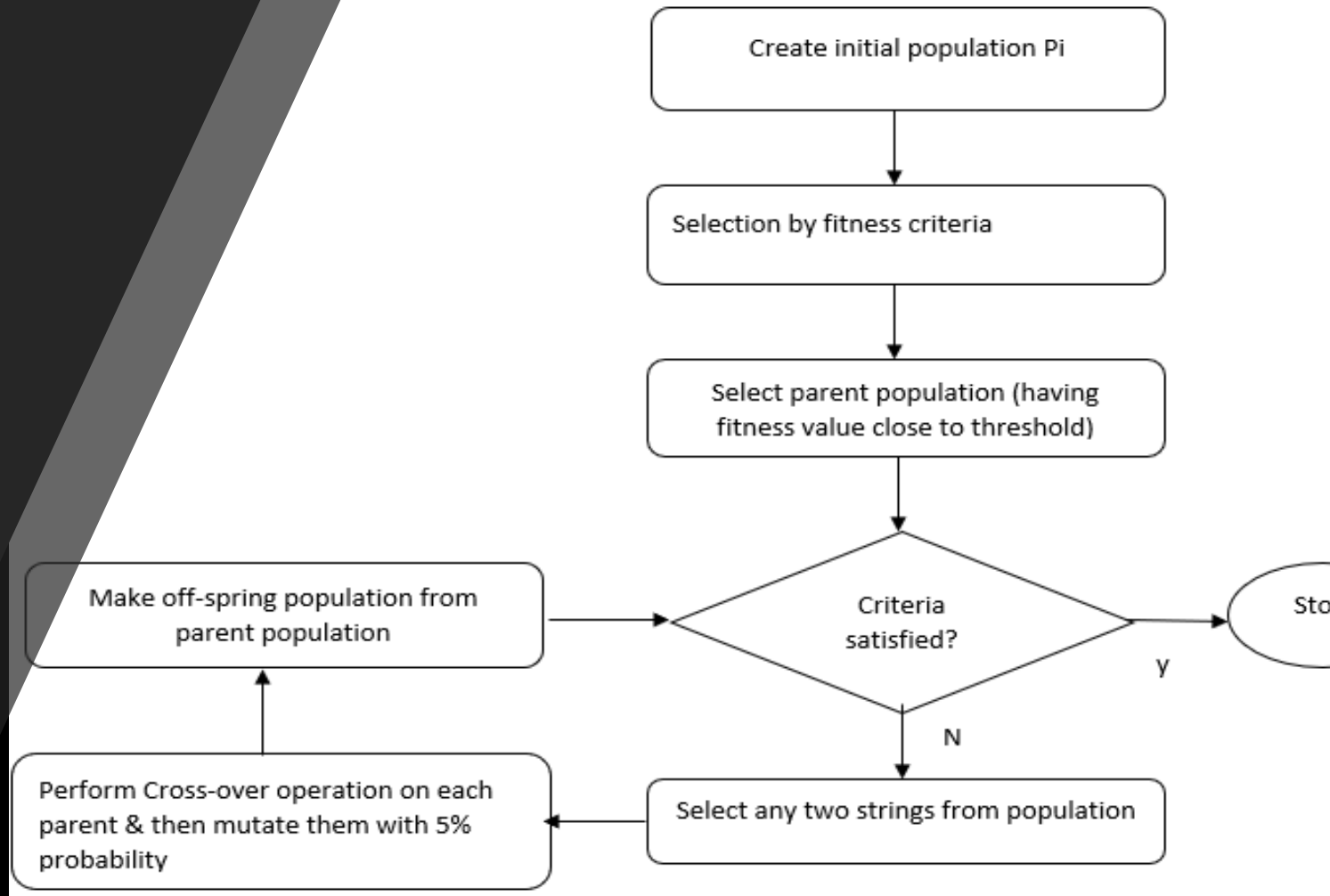
- In step-3 pf the algorithm, we take DFS path to determine the route. i.e if the naïve DFS path is 1->2->3->2->4, we bypass this path and write it as 1->2->3->4. We ran this algorithm on Djibouti country and custom generated data. It was able to give approximate(<= 2* optimal cost) answer in polynomial time. But, the approximate answer is more compared to Nearest neighbour and Christofide algorithm.



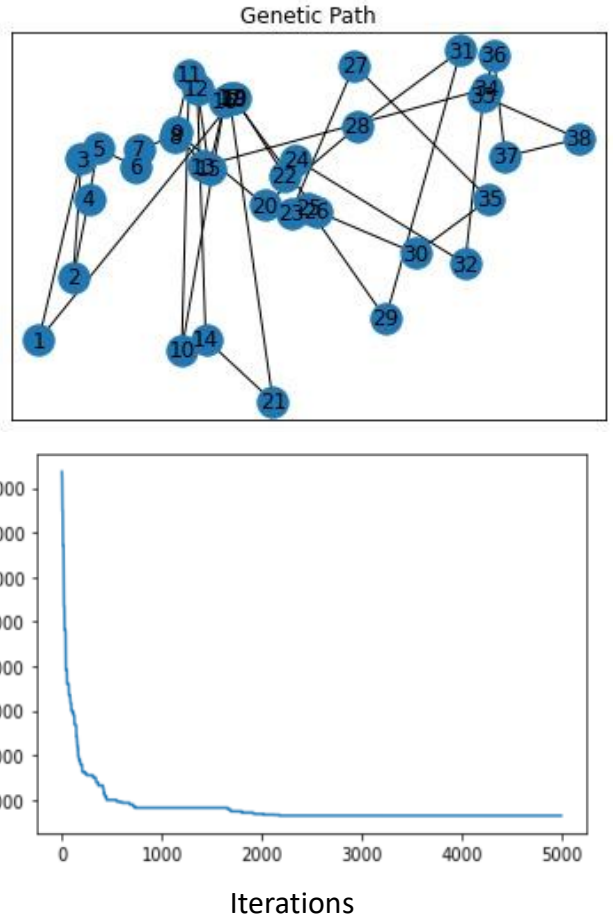
The Travelling Salesman Problem (TSP) using Genetic Algorithms:

- Search technique for finding approximate solutions to combinatorial optimization problems
- Metaheuristic based on the Theory of Evolution by Natural Selection
- Incorporated the evolutionary mechanism of Survival of the Fittest

- **Terms used in Algorithm:**
  - **Chromosome**: Each chromosome represents an individual solution or a particular path of cities in our case.
  - **Crossover**: New chromosomes are obtained from the old ones using crossover operation. This method chooses the old chromosomes with probability of crossover proportional to the fitness value. After choosing parents from a mating pool, we perform crossover operation, where genes after a random point are swapped in both parent chromosomes without duplication.
  - **Mutation**: Individuals are chosen randomly that are to be mutated, and then 2 distinct mutation points for are chosen randomly. Genes corresponding to the mutation point are swapped to produce a new chromosome.
  - **Fitness function: It is an** objective function that minimizes the sum of all costs of all edges in the path

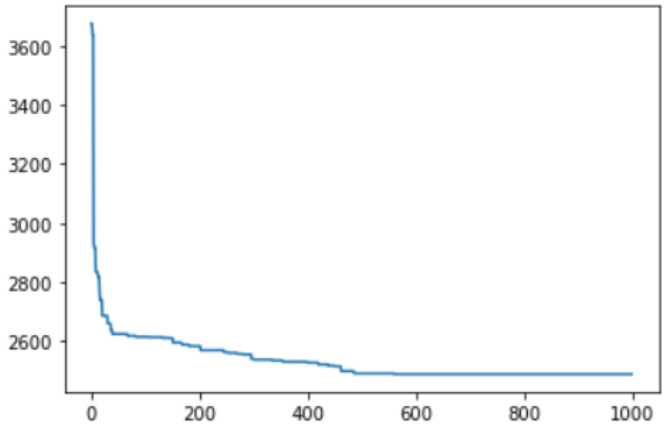


Djibouti data set




```
Total Population: 20
Mutation Probability: 5%
Number of cities: 38
Optimal path cost: 9289.9313 Best Route:
[20, 28, 31, 29, 25, 23, 27, 35, 30, 26, 18, 17,
21, 14, 12, 15, 16,10, 11, 8, 7, 6, 5, 4, 2, 3, 1,
19, 22, 24, 32, 33, 38, 37, 36, 34, 13, 9]
```

Customized data set



```
Total Population: 20
Mutation Probability: 5%
Number of cities: 26
Optimal path cost: 2488.5582999999992
Best Route:
[21, 26, 25, 16, 6, 4, 7, 5, 24, 22,
20, 17, 18, 19, 13, 8, 9, 1, 12, 10,
3, 2, 14, 11, 15, 23]
```

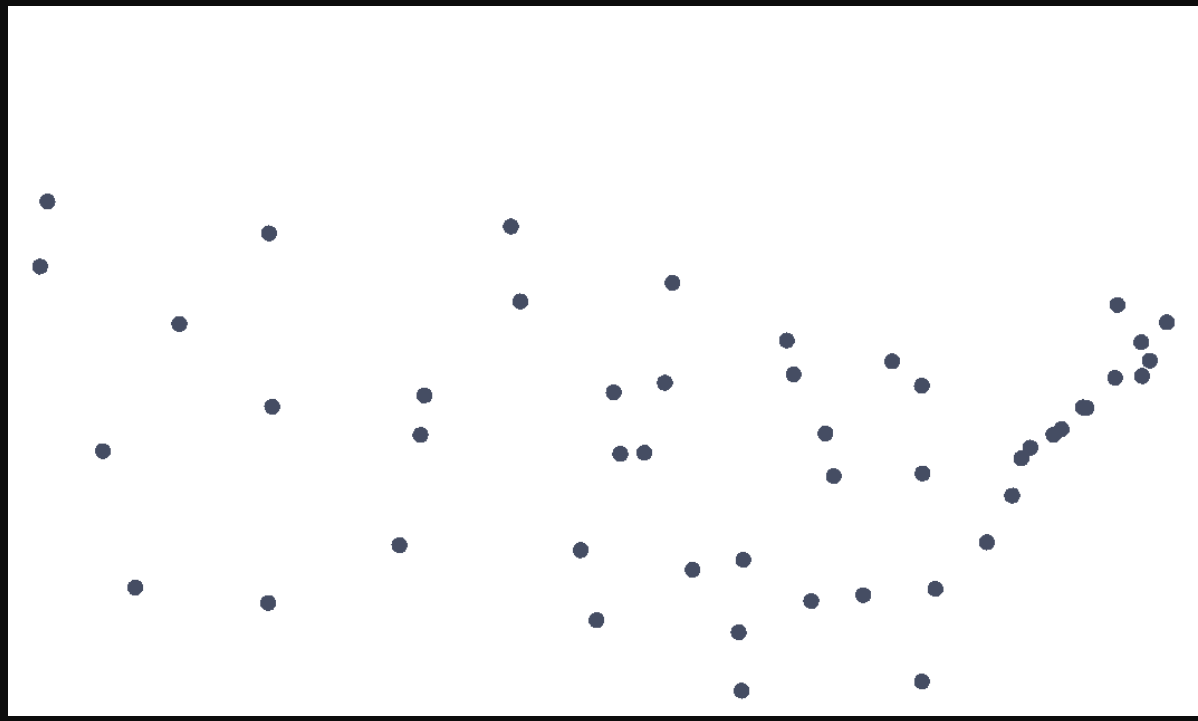
Performance of a GA on TSP depends on:

- Method for selecting initial population of solutions
  - Random vs using a heuristic
- Frequency/probability of operators
  - Selection method, Crossover method, Mutation method
- Stopping criteria

Christofides algorithm

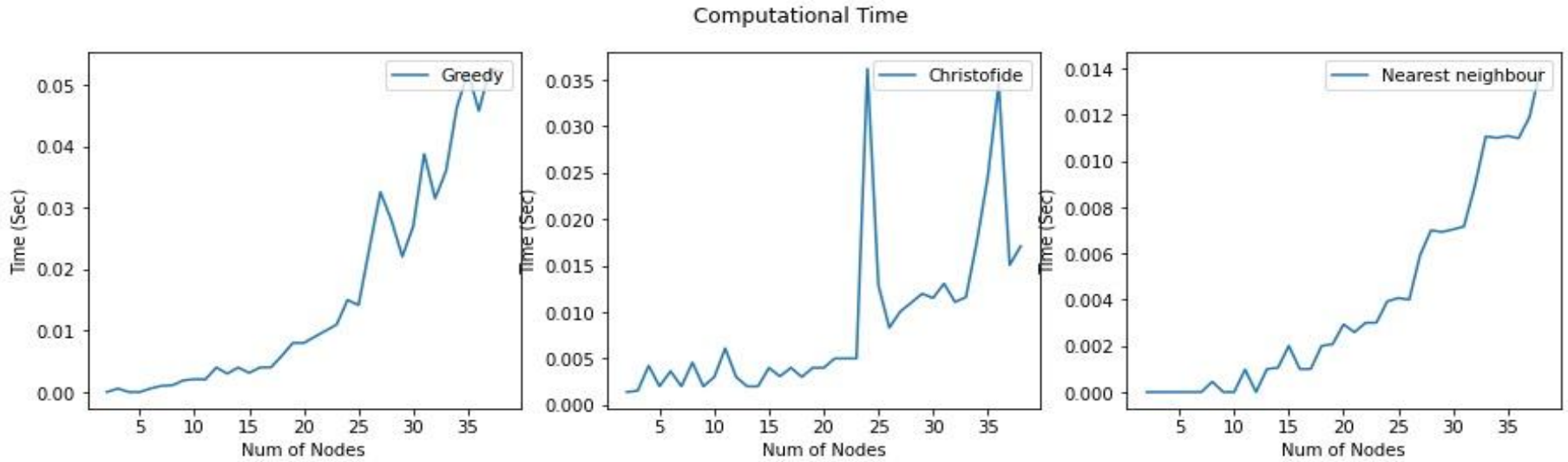
- It finds approximate solutions to the travelling salesman problem, on cities where the distances between them form a metric space (d(a-b) < d(a-c) + d(c-a)). This algorithm provides a solution that will be within a factor of 3/2 of the optimal solution length.

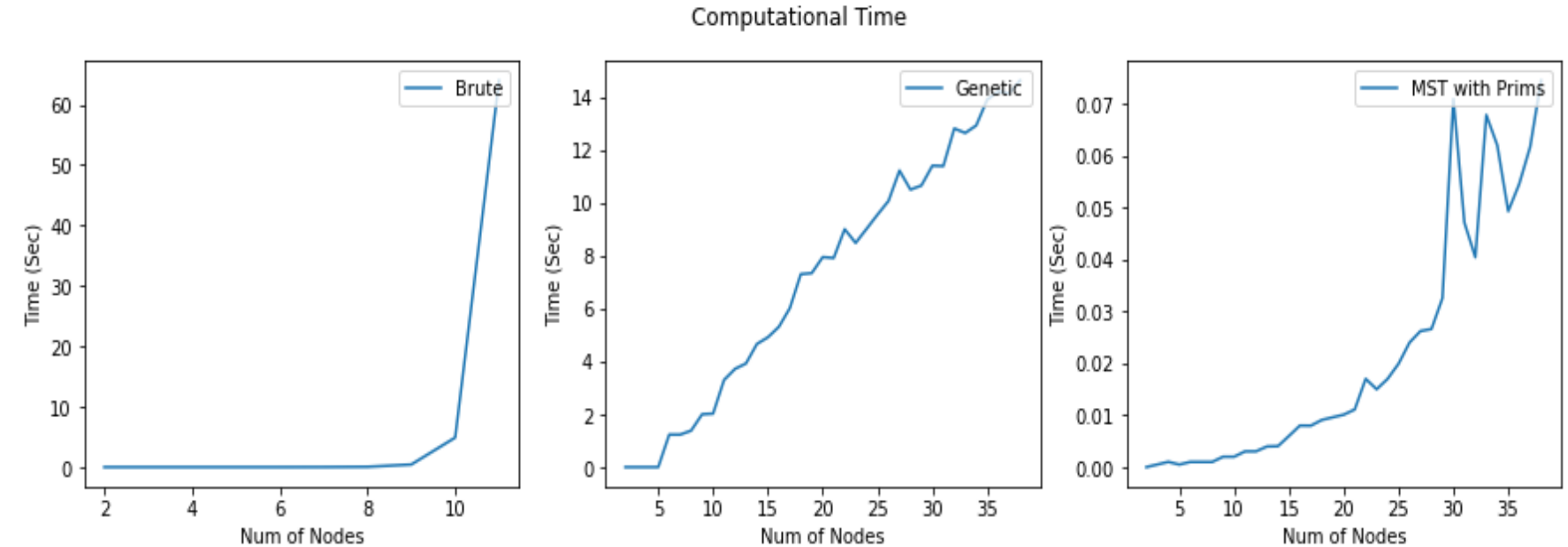
- **Steps in the Algorithm:**
  - Using Kruskal's algorithm form a minimum spanning tree from the given graph G.
  - Identify set of vertices (S) that have an odd degree in MST.
  - Find minimum cost perfect matching (M) from subgraph S.
  - Combine the set of edges from (M) to (T) to form a new graph where all the edges have even degree.
  - Build a Eulerian circuit C using edges from M &T.
  - Remove all repeated vertices of C by shortcutting them & thereby forming Hamiltonian cycle.



RESULTS

Algorithm Name	Minimum Distance	Deviation from Optimal solution	Time Complexity	Run time(s)	Advantages	Disadvantages
Naïve (10 cities)	1070.991	-	O(n!)	67.03	Gives the best solution by considering all possibilities	Time complexity, Exponential growth in time with increase in nodes
Greedy	7838.011	1182.011	O(n^2*logn)	0.092	Easy to implement Runs Faster	Doesn't always guarantee an optimal solution
Nearest Neighbor	6771.477	115.477	O(n^2)	0.036	Easy to implement Runs Faster	Fails in outlier cases(mentioned above)
2-Approximation with MST	15369.215	8713.215	O(n^2)	0.184	Polynomial time complexity	Approximate solution twice the optimal solution
Genetic	9118.846	2462.846	O(g(nm+mn+n))	14.344	Evaluate the quality of solution in each iteration	High computation time
Christofides	8345.973	1689.973	O(n^3)	0.018	Approximate solution is closer to the optimal solution	Approximation Ration can be improved


Computational Time
Greedy
Christofide
Nearest neighbour
Time (Sec)
Num of Nodes


Computational Time
Brute
Genetic
MST with Prims
Time (Sec)
Num of Nodes

Limitations and conclusion

- We were able to provide a solution for the countries of Djibouti.
- We couldn't fully test our algorithms on all 25 countries in the dataset mentioned. However, we can guarantee that the results of our algorithms for finding optimal tours are exact.

- We had to rely on custom generated data because the countries we tested were unable to exploit the limitations of the Nearest Neighbour algorithm
- There are other algorithms such as Tabu search, Simulated annealing, and Branch and Bound, Lin-Kernighan, Linear programming variations that we wanted to investigate.

- However, Concorde TSP Solver will be the core focus of our future efforts. Concorde is a callable library written in the ANSI C programming language. It has over 700 functions that allow users to work on TSP problems in their own unique way. It is currently the best solver for TSP-like solutions.

- So, for any given scenario, based on criteria such as time complexity, proximity to best-found solution, and input size. We can recommend a better algorithm for the given requirements.

- For example, if the number of nodes is less than 10, we can use the greedy algorithm; if a simple implementation and a fast algorithm are required, we can recommend the Nearest Neighbor algorithm. If the input data is large and we still want the best possible solution, we can recommend the Christofide solution. If there are no outliers or if the cities are not too far apart, we can suggest Nearest neighbour Heuristic

FINAL PROJECT PROPOSAL CS5800
FINDING OPTIMAL SOLUTION FOR TRAVELLING SALESMAN PROBLEM

Sai Vineeth Kaza | Rajesh Karumanchi | Chandra Teja Kommineni

ABSTRACT

In this project, we would like to explore different algorithms that solve TSP, implement them, and compare and analyze the results to see which algorithm gives an efficient and optimal solution for TSP. Our main aim of the project is to explore the following algorithms and their approaches to solve TSP: naïve approach, greedy approach, nearest neighbors approach, 2-approximation with MST approach, genetic algorithm and christofides approach. After implementing the algorithms, we compared the obtained results of minimum distance and the time taken and found that nearest neighbors and christofides approach gave us the better results with some limitations for the nearest neighbors approach. Even though our motivations mentioned below explained the various real-life examples which are applications of TSP, we will not cover the solution for those applications in this project.

PROBLEM STATEMENT

The Traveling Salesman Problem(TSP) was first formulated in 1930 by Merrill M. Flood, who was trying to solve a school bus routing problem. It is one of the most intensively studied computational problems used mainly for optimization methods. Travelling Salesman Problem(TSP) is the problem where we have to find the optimal path for a salesman to visit the given list of cities and distances between them by starting from a randomly selected starting city and returning to the same starting city. As there are a lot of routes and as we have the find the shortest/ the most efficient route, this makes it the most challenging computational problem. It is so easy to describe the problem, yet it is so difficult to solve the problem. The naïve approach can be solved by checking each round-trip path possible and find the shortest path. TSP doesn't have a quick solution and also more the number of cities i.e., nodes, more the computational time required which makes TSP classify as a NP Hard problem. For example, in the naïve approach for 10 cities there will be around 320000 possibilities and for 16 cities there will be around 2 trillion possibilities. As, there is no perfect or exact solution available for TSP, the question we are wondering about is which algorithm can be as close to an efficient, effective, and optimal solution for the Travelling Salesman Problem?

PROBLEM MOTIVATION

Sai Vineeth Kaza:

In the current world we can see a huge spike in the number of cabs and taxis on the road over the last few years. More number of people using the cab services will lead to further increase in number of cars on the road. This spike in the cars will lead to lot of traffic and congestion on the roads in big and congested cities like New York, San Francisco etc. It also contributes to a

significant increase in the emission of harmful gases that is resulting in the global warming which is having adverse effects on the world. We can treat this problem as one of the applications of the travelling salesman problem, where the problem is to find optimal route for a salesman to travel given a set of destinations. Here, we can find the optimal solution for the problem by using the data related to current drop off location and next pickup location and the time taken between them. We are planning to explore different algorithms to solve the Travelling Salesman Problem and find an optimal solution for TSP.

Rajesh Karumanchi:

With the rapid expansion of E-commerce, nearly everyone orders things online. It is critical for the organization to deliver the products in a timely and cost-effective manner. Package delivery addresses this issue by taking the shortest available path to save time and fuel. However, it can be improved. Given a list of destinations and distances (or constraints such as time, priorities, and so on), we must find the best route possible. This is comparable to the difficulty of the traveling salesman. Despite the fact that there are certain traditional ways to it, TSP is still regarded as an NP-Hard problem. In other words, there is no specific solution. What excites me is the opportunity to learn more about the inner workings of navigation software such as Google Maps, Apple Maps, and others, as well as what it takes to improve navigation suggestions. In addition, I'm using the mathematical analysis and graph theory that I studied in the algorithm course to a real-world data set. I'm hoping to find something useful that will help to improve the present delivery system.

Chandra Teja Kommineni:

Before joining Northeastern, I worked at Tata motors, an automobile company. During my initial year, I was given the task to optimize supply chain of automotive parts distribution from central warehouses to distributors. With limited resources & knowledge back then, I have devised various logistic models that optimize cost for distribution of frequently used parts & some models that optimize time in case of vehicle off road situations. Since payload contains parts to be delivered to multiple distributors an optimized path would be desirable in terms of transportation cost & time. After knowing travelling sales problem, I now believe that I could design more efficient models for logistics. Travelling sales problem in general have wide range of applications ranging from logistics, last mile delivery planning, carpooling, manufacturing & planning. Given how ubiquitous the applications of TSP are, I would like to explore & analyze various exact & heuristic approaches for solving travelling sales problem.

DATASET

To validate our results on real-world problems, we used data from http://www.math.uwaterloo.ca/tsp/world/countries.html. It contains TSPLIB data points for cities in 25 countries. These TSPLIBs were derived from the National Imagery and Mapping Agency's database of geographical feature names. TSPLIB is a file format that uses the.tsp extension. The .txt view contains the file name, description of the data source and the 'X' and 'Y' coordinates of all city nodes. The Euclidean distance between these coordinates is the distance between each city. We conducted our experiments on Djibouti which has 38 cities. The best optimal cost value for

this country is 6656 (by Concorde). We generate a Euclidean distance matrix using the 'X' and 'Y' coordinates of city nodes, which explains the distance from one city to all others.

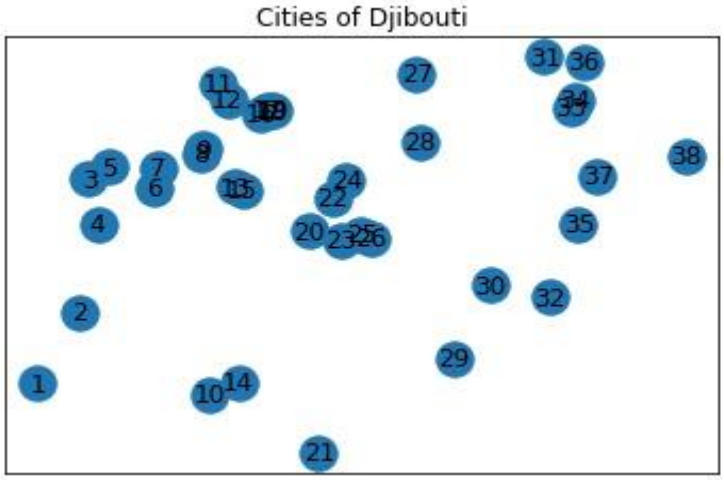


Fig. Dataset

ALGORITHMS

We implemented a total of 6 algorithms namely Naïve, Greedy, 2-approximation with MST, Nearest Neighbors, Genetic and Christofides. Mentioned below is a brief explanation, pseudocode, and time complexity of each algorithm.

NAÏVE

Naïve approach is the basic brute force approach of the algorithm where we calculate the minimum distance by calculating all the possible permutations of the paths and find the shortest unique solution. In this approach we first list the total number of paths possible. We will then calculate the distance of required by each path to travel through all the cities and return back to the starting city. Finally, we will pick the minimum distance path as the shortest and optimal solution for TSP.

Pseudocode:

1. Consider a random city as the starting node and ending node and iterate over all other cities.
2. All possible path permutations between the cities which will be (n-1)! are generated.
3. Calculate the distance of each path and return the minimum distance as the optimal path.

As we observe from the pseudocode, we can see the time complexity of the algorithm is O(n!) as it is generating all the possible path permutations from a starting node using the formula (n-1)! and iterating it over all the available cities as the starting points i.e., O(n*(n-1)!) = O(n!). The time complexity of this algorithm makes it the worst available solution for TSP. Time taken to run this algorithm will increase exponentially with the increase in the number of cities. Due to this limitation, we took only 10 out of 38 available cities in the dataset. Our algorithm resulted in a minimum distance of **1070.991 (for 10 cities)** and time taken is **67.03** seconds.

GREEDY

Greedy approach is a heuristic algorithm which is better and optimized version of naïve approach. We can say that greedy approach is finding the shortest Hamilton cycle from all the possible cycles. Hamilton cycle is forming a cycle by visiting all the nodes of the graph only once. As stated in the name of the approach 'greedy', we use greedy choices to optimize the algorithm for TSP. This approach finds the local optima for the problem and then optimizes it to form the global optima solution for the problem. So, for our current TSP problem at each step of the journey we update the final distance with the minimum obtained distance and return the path.

Pseudocode:

1. Consider city 1 as the starting node and ending node.
2. Sort all the available edges of the graph.
3. Create a list to track the visited nodes.
4. Create a list to track the edges of the tour and add the minimum distance edge to the tour at each and every step.
5. If all the nodes are visited, we will end the algorithm and return the tour with the shortest path.

The time complexity of greedy approach algorithm is O(n^2 * logn). For a single node finding the minimum distance path with a particular node as a starting node will take O(n*logn). Iterating over the remaining nodes to get the minimum distance tour possible will make the time complexity O(n*nlogn) = O(n^2 * logn). It is easy to implement, and the algorithm runs faster than the naïve approach. But this approach does not guarantee us the best optimal solution in all the cases. Our algorithm resulted in a minimum distance of **7838.011** and time taken is **0.092** seconds.

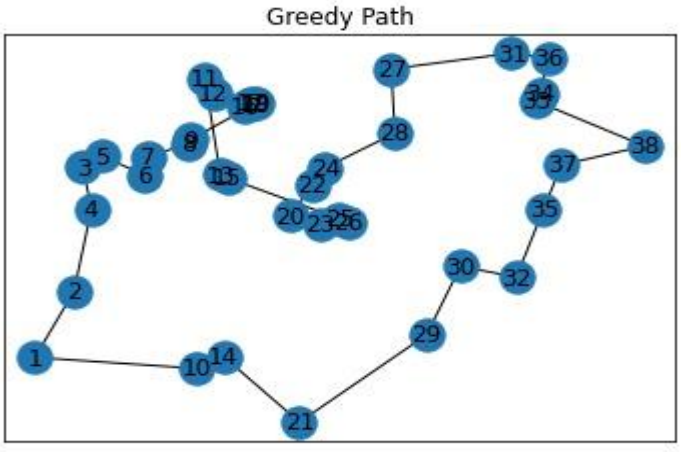


Fig. Optimal Path for Greedy Algorithm

NEAREST NEIGHBOR

The idea here is to start with a random city and visit nearby ones repeatedly until we have visited all of them. In other words, given a set of cities, this algorithm joins their nearest neighbours. This heuristic is also a greedy algorithm variant.

Pseudocode:

1. Choose a starting vertex and make it the current node.
2. Examine all the other nodes that have not yet been selected and select the closest one.
3. Repeat step 2 until all nodes have been selected at least once.
4. Return to the starting vertex if all nodes are selected.
5. Draw and print the path, then calculate the distance.

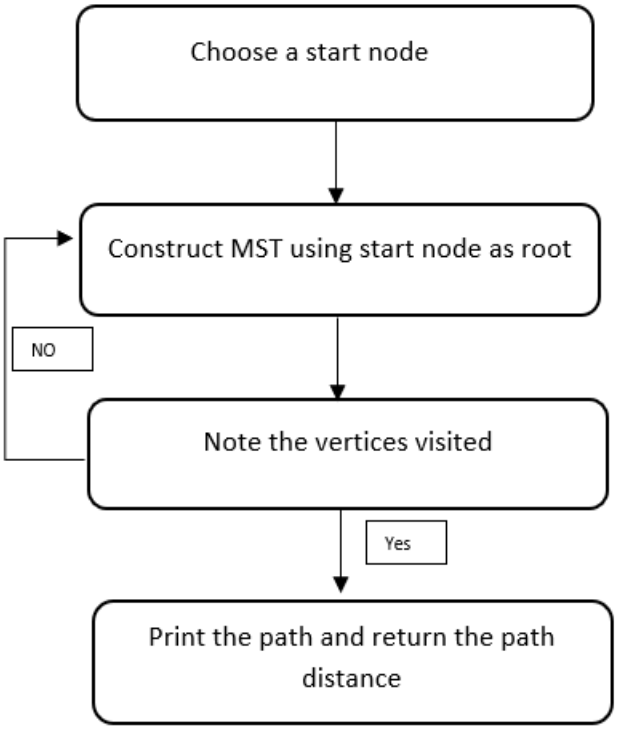


Fig. Flow Diagram Nearest Neighbors Approach

We check a vertex 'I' with rest 'n-I' vertices. It takes O(n^2). Printing path and calculating distance takes linear time. So, time complexity is O(n^2). The nearest neighbor algorithm is simple to implement and runs quickly (in polynomial time). However, due to its greedy character, it sometimes misses the shortest paths and fails in the presence of outliers. That is, if some vertices are too far apart from the others. The Nearest Neighbor algorithm will then cover a group of nearby cities before moving on to the outliers. This could lengthen the path. For Djibouti country this algorithm has given best approximation. We also used the Nearest Neighbor algorithm with multiple starting points O(n^3) to find the best travel route in Djibouti. Our algorithm resulted in a minimum distance of **6771.47657** and time taken is **0.036025** seconds.

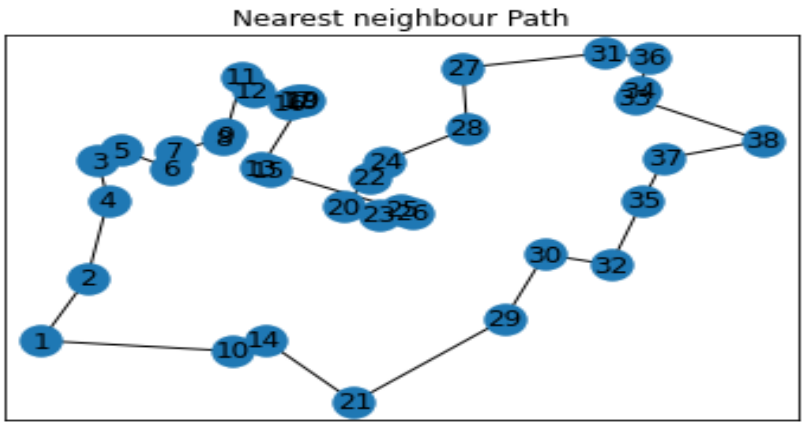


Fig. Optimal Path for Nearest Neighbour Algorithm

2-APPROXIMATION WITH MST:

2- The approximation algorithm performs well if the set of given nodes adheres to the triangle inequality. The Triangle inequality states that the distance between vertex 'A' and vertex 'B' is shortest if we travel directly from 'A' to 'B' rather than via some intermediate vertex 'C'. Distance (A, B) <= Distance (A, C) + Distance (C, B). This algorithm employs Prim's MST and Triangle inequality concepts to find a path that is always less than twice the cost of optimal path.

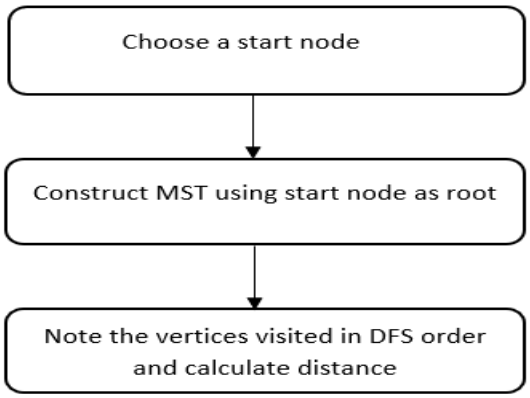


Fig. Flow Diagram 2-Approximation with MST

Pseudocode:

1. Choose a random node as the starting and ending point.
2. Create a minimum spanning tree with the start point as the root node using Prim's algorithm.
3. For path consider the path in which vertices are visited in Depth First Search order of the minimum spanning tree obtained and add starting vertex at the end.

The time complexity of this algorithm is O(n^2). In step-3 pf the algorithm, we take DFS path to determine the route. i.e., if the naïve DFS path is 1->2->3->2->4, we bypass this path and write it as 1->2->3->4. We ran this algorithm on Djibouti country. It was able to give approximate (<= 2* optimal cost) answer in polynomial time. But the approximate answer is more compared to

Nearest neighbour and Christofides algorithm. Our algorithm resulted in a minimum distance of **15369.21517** and time taken is **0.183861** seconds.

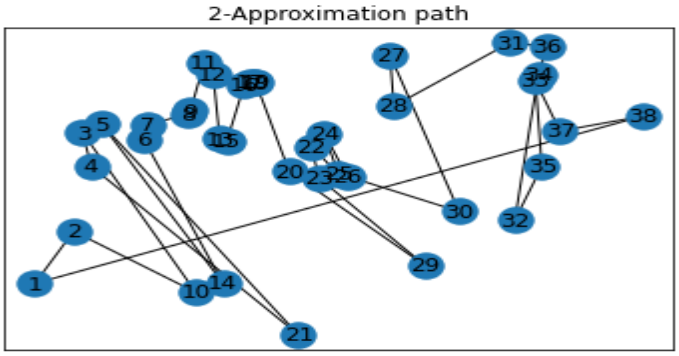


Fig. Optimal Path for 2 -Approx. with MST

GENETIC

Genetic approach is a search technique for finding approximate solutions to combinatorial optimisation problems. It is a metaheuristic based on the theory of evolution by natural selection. This approach incorporates the evolutionary mechanism of survival of the fittest. This approach uses chromosome which represents an individual solution or a particular path of cities in our case. New chromosomes are obtained from the old ones using crossover operation. This method chooses the old chromosomes with probability of crossover proportional to the fitness value. After choosing parents from a mating pool, we perform crossover operation, where genes after a random point are swapped in both parent chromosomes without duplication. Individuals are chosen randomly that are to be mutated, and then 2 distinct mutation points for are chosen randomly. Genes corresponding to the mutation point are swapped to produce a new chromosome. Fitness function is an objective function that minimizes the sum of all costs of all edges in the path.

Pseudocode:

1. Starts with randomly generating k chromosomes called population.
2. The chromosomes crossover and goes through mutation to create k offspring.
3. Within k offspring & k population, we detect k best chromosomes using survival of fittest function.
4. These new k chromosomes become the new population
5. The process of evolution (crossover & mutation) loops until iteration threshold is reached.

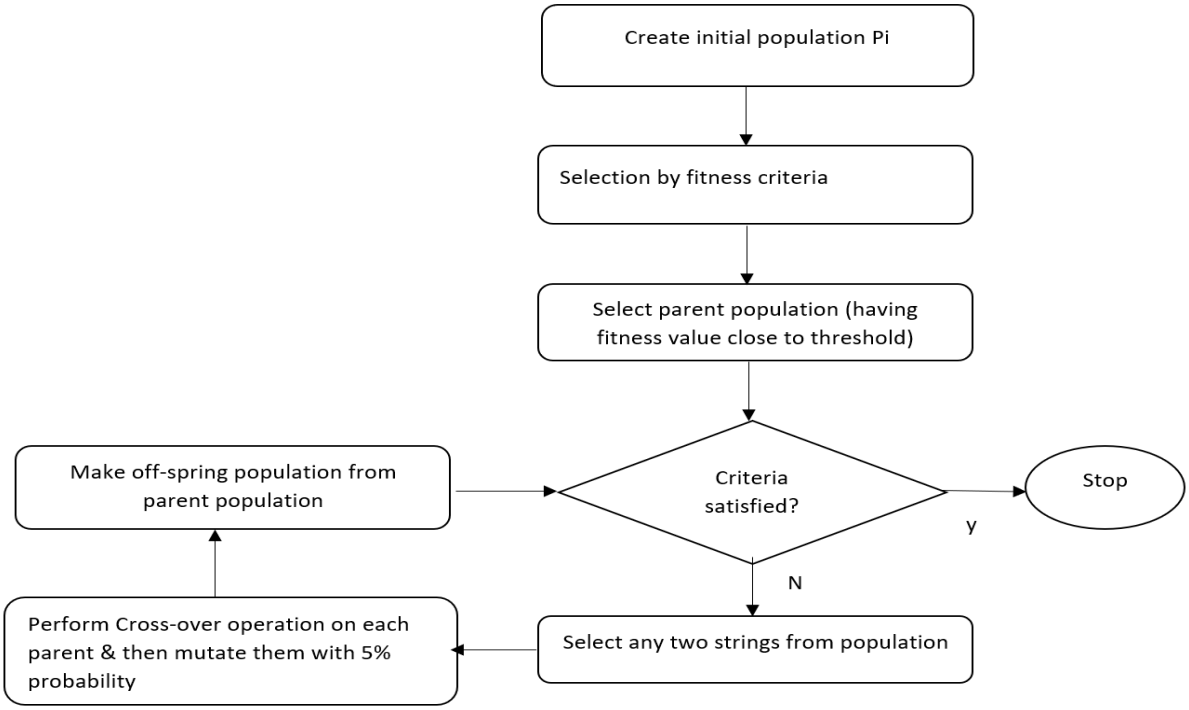


Fig. Flow Diagram Genetic Algorithm

Time complexity for genetic algorithm is O(g(nm + nm + n)), where g is no. of generations, n is population size, m is no. of nodes. Pros of this approach is there is no need to know how to solve the problem, we have to only evaluate the quality of solution coming in every iteration. Cons of this approach is it finds only an approximate solution rather than an optimal solution. Performance of this approach on TSP depends on the method for selecting initial population of solutions i.e., random vs using a heuristic. It also depends on frequency or probability of operators (Selection method, Crossover method, and Mutation method) and stopping criteria. Our algorithm resulted in a minimum distance of **9118.846** and time taken is **14.3435** seconds.

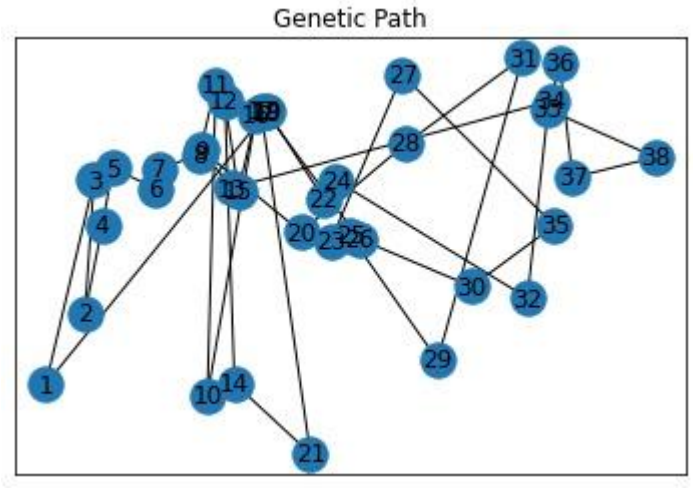


Fig. Optimal Path for Genetic Approach

CHRISTOFIDES

Christofides approach finds the approximate solutions to the travelling salesman problem, on cities where the distances between them form a metric space (distance(a-b) < distance(a-c) + distance(c-a)). This algorithm provides a solution that will be within a factor of 3/2 of the optimal solution length.

Pseudocode:

1. Using Kruskal's algorithm form a minimum spanning tree from the given graph G.
2. Identify set of vertices (S) that have an odd degree in MST.
3. Find minimum cost perfect matching (M) from subgraph S.
4. Combine the set of edges from (M) to (T) to form a new graph where all the edges have even degree.
5. Build a Eulerian circuit C using edges from M &T.
6. Remove all repeated vertices of C by shortcutting them & thereby forming Hamiltonian cycle.

The time complexity of the algorithm is O(n^3), where n is number of nodes. Pros of this approach is that this algorithm provides approximate solution which is close to optimal solution with a worst-case ratio of 1.5. Cons of this approach is that the computational time is of higher order compared to Nearest neighbour & Greedy approach. Our algorithm resulted in a minimum distance of **8345.973** and time taken is **0.0178** seconds. The key terms used in the above pseudocode are explained below

Minimum Spanning Tree: MST is a weighted, connected, and undirected graph is a tree that spans all the vertices of the given graph with a weight less than or equal all possible spanning trees.
Kruskal's algorithm: Sort the edges of the graph based on their weights. Select the edges (& perform union operation on the Vertices) having minimum weight & doesn't a loop with selected edges iteratively. Stop the algorithm when reaching a union list of total V-1 nodes.
Finding degree of Vertex: From the adjacency matrix check if the vertex has a path to other vertex, every time a new path is found increment the degree of the vertex by 1. Iteratively perform this operation for all the vertices. Vertices with odd degree are found to find perfect matches & making the degree of every vertex even.
Perfect Matching: Matching graph is a sub graph of a graph G where there are no edges adjacent to each other.
Eulerian Circuit: Circuit is a closed walk that repeats no edges. Eulerian circuit in a graph G is a circuit that passes through every edge in G. A connected graph is said to be Eulerian if and only if all its vertices have even degrees.

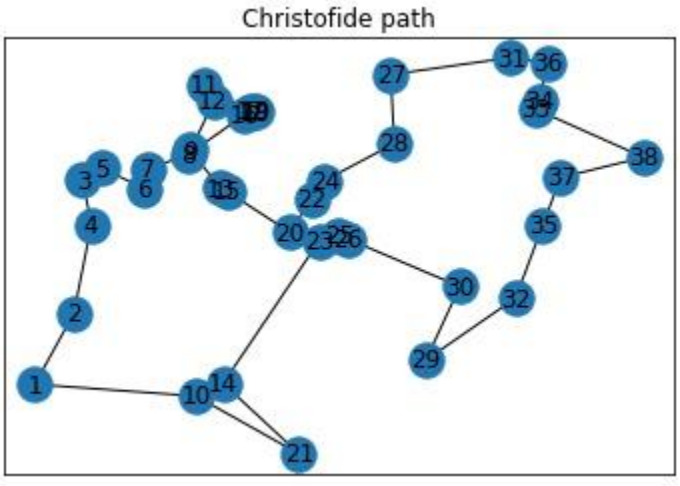


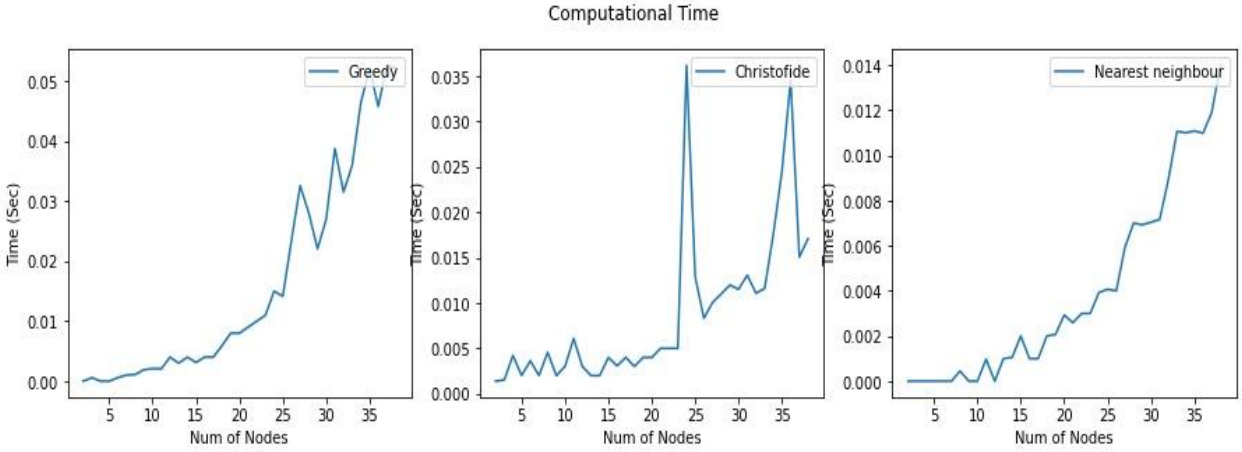
Fig. Optimal Path for Christofides Approach

RESULTS:

	Time complexity	Solution	Input size
Exact	High	Optimal	Low
Heuristic	Low	Varies based on input	High
Approximate	Low	α * Optimal	High

Table. Algorithm Characteristics

In the above table, we can see the different characteristics of the algorithm types. We can refer to the above table to pick the algorithm based on the type of solution required, time complexity and input size. For example, we pick naïve approach (exact) if the input size is low, and it can give optimal solution. In the same manner, we can pick the approach based on our requirements.



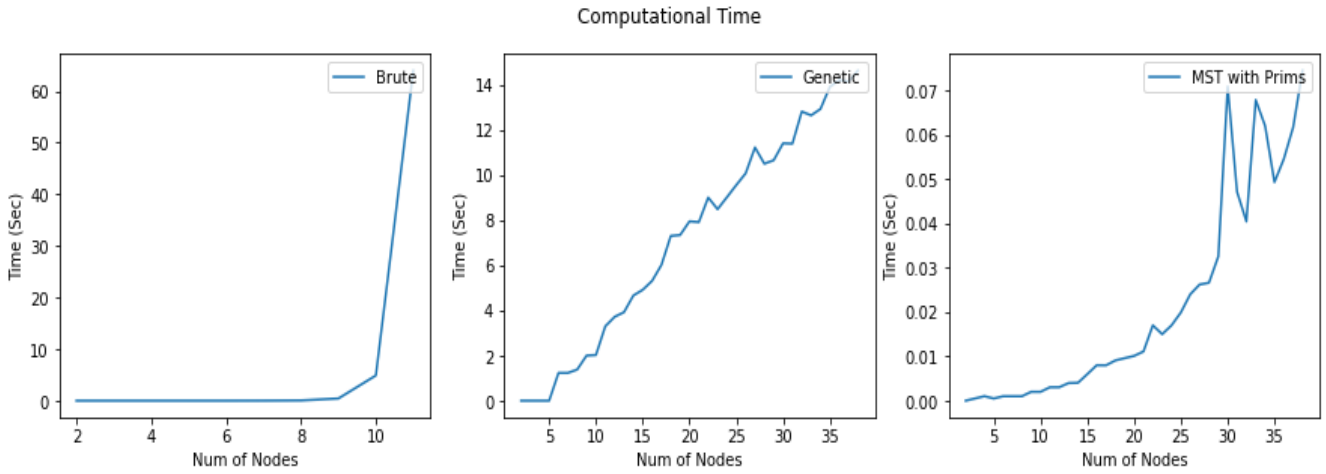


Fig. Computational Time vs Number of Nodes for different algorithms

From the figure above, we can see that the Naïve (Brute) approach has an exponential time complexity. As the number of nodes increases, the computational time increases exponentially. We can observe that greedy, nearest neighbours, 2-way approx. with MST and christofides has a polynomial time complexity. Even from the above graphs, we can say that nearest neighbours are giving the optimal solution in less time. Genetic algorithm has high computational time as observed in the graphs shown above.

Algorithm Name	Minimum Distance	Deviation from Optimal solution	Time Complexity	Run time(s)	Advantages	Disadvantages
Naïve (10 cities)	1070.991	-	O(n!)	67.03	Gives the best solution by considering all possibilities	Time complexity, Exponential growth in time with increase in nodes
Greedy	7838.011	1182.011	O(n^2*logn)	0.092	Easy to implement Runs Faster	Doesn't always guarantee an optimal solution
Nearest Neighbor	6771.477	115.477	O(n^2)	0.036	Easy to implement Runs Faster	Fails in outlier cases(mentioned above)
2-Approximation with MST	15369.215	8713.215	O(n^2)	0.184	Polynomial time complexity	Approximate solution twice the optimal solution
Genetic	9118.846	2462.846	O(g(nm+mn+n))	14.344	Evaluate the quality of solution in each iteration	High computation time
Christofides	8345.973	1689.973	O(n^3)	0.018	Approximate solution is closer to the optimal solution	Approximation Ration can be improved

Table. Implemented Algorithms Comparison Table

From the above table, we can clearly see that we got nearest neighbour algorithm as the best approach for the current dataset of 38 cities with the lowest deviation of 115 from the given optimal solution 6656 and with a best minimal distance value of 6771.477 compared to other

approaches. We can also see that christofides algorithm has the fastest computation time out of all the approaches. We can clearly see that 2 – approximation with MST and Genetic approach are not the best approach to solve this dataset TSP, the former with high deviation and the latter with highest computation time. From the above table, you can see the advantages and disadvantage of each algorithm as well.

LIMITATIONS, CONCLUSION AND FUTURE WORK

We are restricted in a variety of ways due to time constraints. Even though we mentioned several real-life examples of TSP applications. We were able to provide a solution for the countries of Djibouti. We were unable to cover the solution for the applications mentioned above. We couldn't fully test our algorithms on all 25 countries in the dataset mentioned. However, we can guarantee that the results of our algorithms for finding optimal tours are exact. We had to rely on custom generated data because the countries we tested were unable to exploit the limitations of the Nearest Neighbour algorithm. Finally, even though we examined three heuristic and two approximate algorithms in addition to Naive, there are other algorithms such as Tabu search, Simulated annealing, and Branch and Bound, Lin-Kernighan, Linear programming variations that we wanted to investigate.

As a result, in the future, we will need to implement and test more algorithms on the real-world dataset we used. We also want to create a user-friendly interface that can provide solutions to the applications mentioned above. However, Concorde TSP Solver will be the core focus of our future efforts. Concorde is a callable library written in the ANSI C programming language. It has over 700 functions that allow users to work on TSP problems in their own unique way. It is currently the best solver for TSP-like solutions. There is a lot of research scope. For example, by providing a good initial solution as input, we can improve the solution. So, we intend to conduct extensive research on Concorde. However, for the purposes of our project, we were able to investigate and test various heuristic and approximation-based algorithms on real-world data. So, for any given scenario, based on criteria such as time complexity, proximity to best-found solution, and input size. We can recommend a better algorithm for the given requirements. For example, if the number of nodes is less than 10, we can use the greedy algorithm; if a simple implementation and a fast algorithm are required, we can recommend the Nearest Neighbour algorithm. If the input data is large and we still want the best possible solution, we can recommend the Christofides solution. If there are no outliers or if the cities are not too far apart, we can suggest Nearest neighbour Heuristic.

Travelling salesman problem provides optimal cost for a tour for a single person to travel, but in real world scenario's one may have to redesign the problem to include multiple persons & thereby finding multiple tour routes that finally comes to same node. Vehicle routing problem solves this & applications include employee bus route, warehouse to customer delivery. Genetic algorithm chooses paths at random and improves them by generating children based on fitness function. It may take a lot of time to reach global optimal value. This process can be accelerated by including 3/2 opt path in one of its populations.

Sai Vineeth Kaza

From this project, I have learned about different algorithms to solve the Travelling Salesman Problem(TSP). I have implemented some of these algorithms which helped me a lot in understanding TSP and applying the algorithms learnt in the class. This project also showed that not all the complex algorithms are better. There might be basic algorithms which might work better than complex algorithms. This completely depends on the dataset we are dealing with. I am happy that, I went out of the course syllabus and learnt a lot of new algorithms to solve TSP. Coding some of these programs were really challenging and helped me improve my coding skills.

Rajesh Karumanchi

I was able to apply the algorithms I learned in this course to real-world data in this project. A more complex algorithm does not have to perform well. As demonstrated in our project, the Genetic algorithm is more complex than the Christofides and Nearest Neighbor algorithms. However, they provided the closest approximation. I was able to improve the code by using the mathematical analysis from the algorithm course. I also improved my research skills and understood how to better apply an algorithm based on specific requirements, as well as the benefits and drawbacks of selecting a specific algorithm. I am confident that this experience will help me in the future.

Chandra Teja Kommineni

In this project, I've learned various ways to find solutions to Travelling salesman problem. TSP being an NP-hard problem takes exponential time to solve using brute force methods. Hence, we trade off on accuracy of solution to implement the algorithm in polynomial time. There are broadly two classes of approaches to this – 1) Heuristics (inspired by nature such as genetic algorithms, ant colonization, simulated annealing), 2) Approximation algorithms such as 2-opt & christofides, these algorithms guarantee output to be bounded by a factor times optimal solution. To implement the algorithm, I had to use various other algorithms such as Union Find, Kruskal, finding degree, and Roulette wheel method for performing crossover on genetic algorithms. Hence, it has been a great learning experience for me.

LINKS

Git-hub Link: https://github.com/Rajesh-karumanchi/Algorithm-project
Presentation Link: https://www.youtube.com/watch?v=j2SJ7QKgadE

REFERENCES

[1]https://stemlounge.com/animated-algorithms-for-the-traveling-salesman-problem/#:~:text=The%20time%20complexity%20of%20the,grow%20faster%20than%20n%5E2.
[2]https://www.ijser.org/researchpaper/Solving-TSP-using-Genetic-Algorithm-and-Nearest-Neighbour-Algorithm-and-their-Comparison.pdf
[3] https://www.geeksforgeeks.org/travelling-salesman-problem-set-2-approximate-using-mst/

[4] https://courses.engr.illinois.edu/cs598csc/sp2011/lectures/lecture_2.pdf
[5] https://medium.com/opex-analytics/heuristic-algorithms-for-the-traveling-salesman-problem-6a53d8143584
[6] https://www.math.uwaterloo.ca/tsp/gallery/igraphics/index.html
[7] https://www.ijeat.org/wp-content/uploads/papers/v4i6/F4173084615.pdf
[8] http://cs.indstate.edu/~zeeshan/aman.pdf
[9] Corman H. Thomas, Leiserson E. Charles, Rivest L. Ronald, Stein Clifford "Introduction to Algorithms"
[10]https://medium.com/ivymobility-developers/algorithm-a168afcd3611#:~:text=Greedy%20Algorithm%20for%20TSP,that%20no%20loops%20are%20formed.
[11] https://www.geeksforgeeks.org/traveling-salesman-problem-using-genetic-algorithm/
[12] http://www.cs.cornell.edu/courses/cs681/2007fa/Handouts/christofides.pdf
[13] https://www.wired.com/2013/01/traveling-salesman-problem/