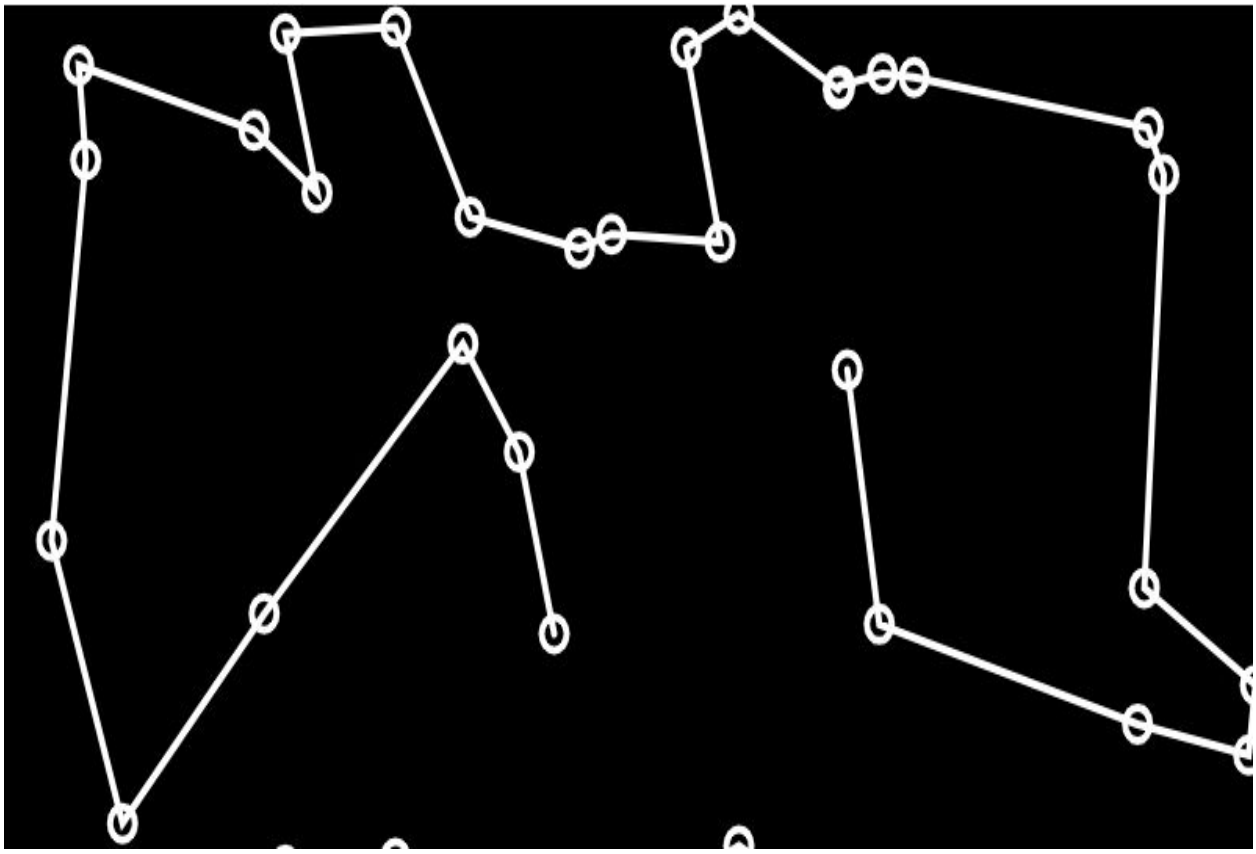
The Travelling Salesperson Problem and its solution using a genetic algorithm

Stafan Santhosh
191IT151

Pranav Surendran
191IT239

Shivam Isarka
191IT148



Introduction:

The traveling salesman problem is a problem that has a significant area of interest for computer science researchers. It is one of the most intensely studied optimization problems. We use it as a benchmark for many optimization methods. Euler is the first to develop an instance of the traveling salesman problem in 1759, whose trial moved a knight to every chessboard position exactly once.

The traveling salesman first gained fame in a book written by German salesman BF Voigt in 1832 to be a successful traveling salesman. He mentions the TSP, although not by that name, by suggesting that to cover as many locations as possible without visiting any site twice is the most critical aspect of scheduling a tour. The origins of the TSP in mathematics are dubious, but we know it happened around 1931.

What is the Travelling Salesperson Problem?

Let G = (V, A) be a graph.
V is a set of n vertices.
A is a set of arcs or edges.
C: be a distance matrix associated with A. The TSP consists of determining a minimum distance circuit passing through each vertex once and only once.
TSP can be modeled as an undirected weighted graph, such that cities are the graph's vertices, paths are the graph's edges, and a path's distance is the edge's length. It is a minimization problem starting and finishing at a specified vertex after having visited each other vertex exactly once. The model is often a complete graph (an edge connects, i.e., each pair of vertices). If no path exists between two cities, adding an arbitrarily long edge will complete the graph without affecting the optimal tour.

We look at asymmetric TSP in this paper. In the asymmetric TSP, paths may not exist in both directions, or the distances might differ.

Commonly Used Methods of solving TSP

Brute Force:
The procedure consists of generating all possible tours and evaluating their corresponding tour length. We select the tour with the smallest length as the best, which is guaranteed to be optimal.

Greedy Algorithm:
The algorithm creates a list of all edges in the graph and then orders them from the smallest cost to the highest cost. It then chooses the edges with the smallest cost first, providing they do not create a cycle. The greedy algorithm gives feasible solutions; however, they are not always right.

Nearest Neighbour:
It is similar to the greedy algorithm in its straightforward approach. We arbitrarily choose a starting city and then travel to the town closest to it that does not create a cycle. We continue to do this until all cities are on tour.

Minimum Spanning tree:

It is a set of n − 1 edge that connects all cities so that the sum of all the advantages used is minimized. Once we have found a minimum spanning tree for our graph, we can create a tour by treating the edges in our spanning tree as bidirectional edges. We then start from a city connected to one other city (this is known as a ’leaf’ city) and continue following untraversed edges to new cities. If there is no untraversed edge, we go back along the previous edge. We continue to do this until we return to the starting city. This will give us an upper bound for the optimal traveling salesman tour.

Genetic Algorithm:

The genetic algorithms are more appropriately said to be an optimization technique based on natural evolution. They include the survival of the fittest idea algorithm. The idea is first to guess the solutions and then combining the most fitting solution to create a new generation of solutions that should be better.

Work Done:

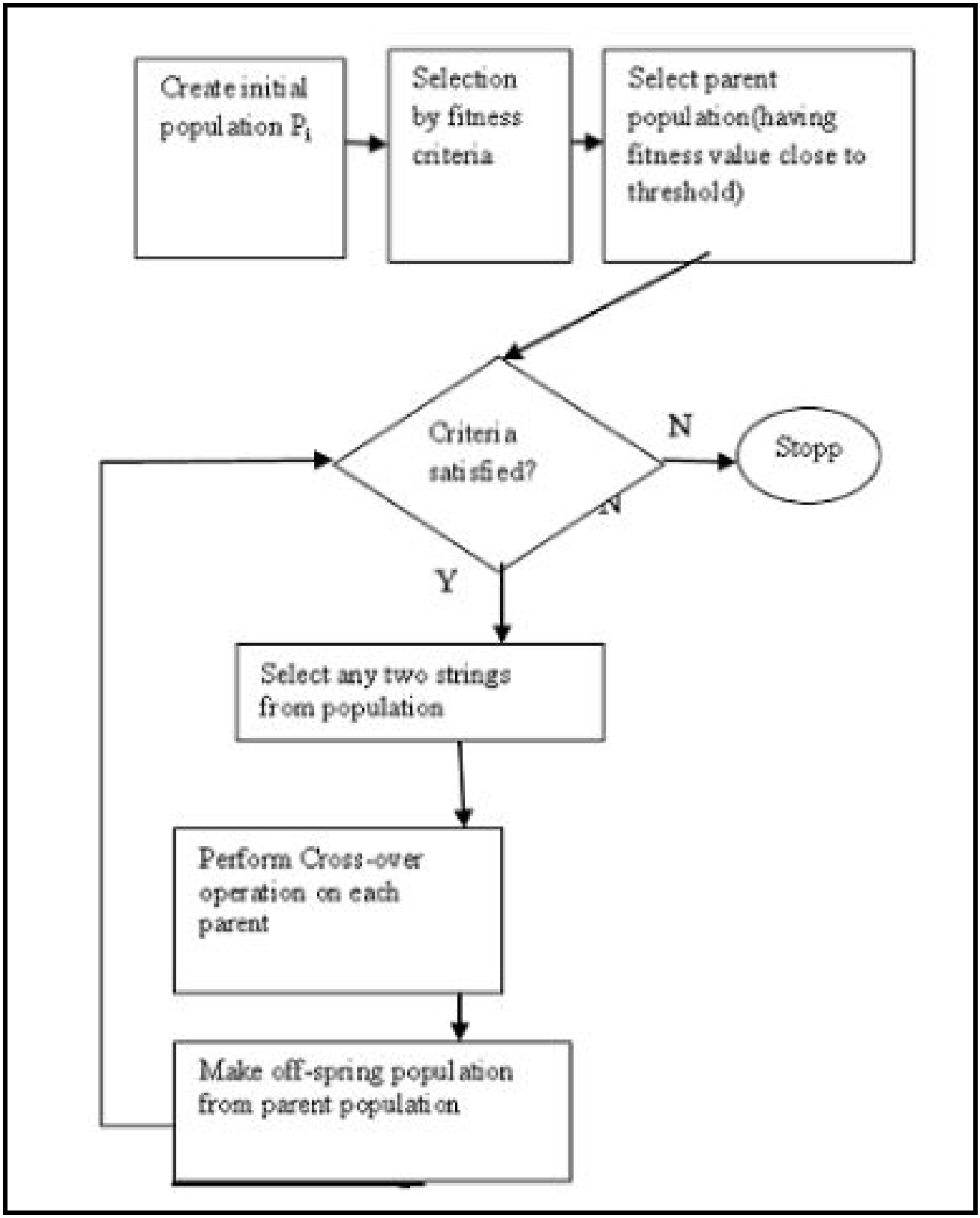
We implemented the brute force algorithm and the genetic algorithm in comparison.

The brute force algorithm was relatively simple and easy to implement. We run through all possible combinations in routes and choose the best one.

Implementation of the Genetic algorithm:

1. [Start] Generate a random population of n chromosomes (suitable solutions for the problem)
2. [Fitness] Evaluate the fitness f(x) of each chromosome x in the population
3. [New population] Create a new population by repeating the following steps until the new population is complete.
4. [Selection] Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)

5. [Crossover] is performed with a probability known as crossover probability that crossover the selected parents to produce a new offspring (children). If crossover probability is 0%, children are an exact copy of the parents.
6. [Mutation] is performed with a probability known as mutation probability that mutates new offspring at each locus (position in chromosome).
7. [Accepting] Place new offspring in a new population.
8. [Replace] Use the newly generated population for running the algorithm.
9. [Test] If the termination condition is satisfied, then stop the algorithm, and return the best solution in current population.
10. [Loop] Go to step 2


Create initial population P_i
Selection by fitness criteria
Select parent population(having fitness value close to threshold)
Criteria satisfied?
N
Stopp
Y
Select any two strings from population
Perform Cross-over operation on each parent
Make off-spring population from parent population

Control Parameters:

(a) Population size: - It determines how many chromosomes and, after that, how much genetic material is available for use during the search. If it is too little, the search has no chance to cover the space adequately. If there is too much, the GA wastes time evaluating chromosomes.
(b) Crossover probability: - It specifies the probability of crossover occurring between two chromosomes.
(c) Mutation probability: - It specifies the probability of doing a bit-wise mutation.
(d) Number of cities: - Determines how long the whole process is going to take.

Structure of genetic algorithms:

GAs may be summarized as follows:

```
GA( )
{ Initialize random population;
Evaluate the population;
Generation = 0;
While the termination criterion is not satisfied
{ Generation = Generation + 1;
Select good chromosomes by reproduction
procedure;
Perform crossover with a probability of crossover
(Pc);
Select fitter chromosomes by survivor selection
procedure;
Perform mutation with a probability of mutation (Pm);
Evaluate the population;
}
}
```

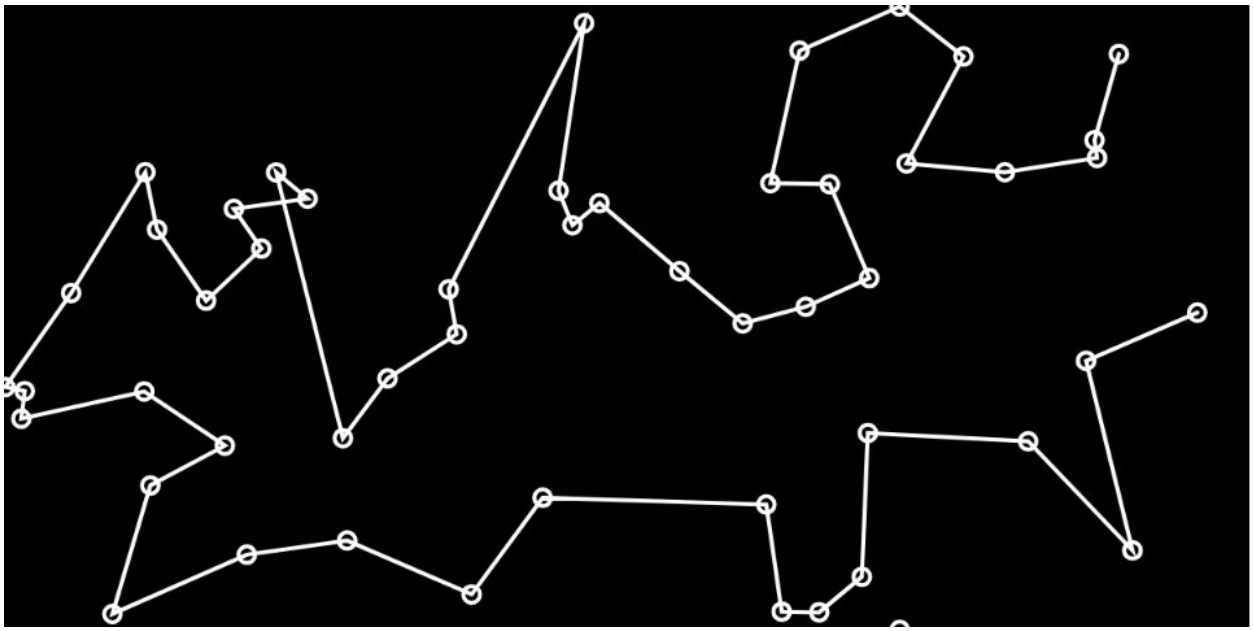
Results and discussion

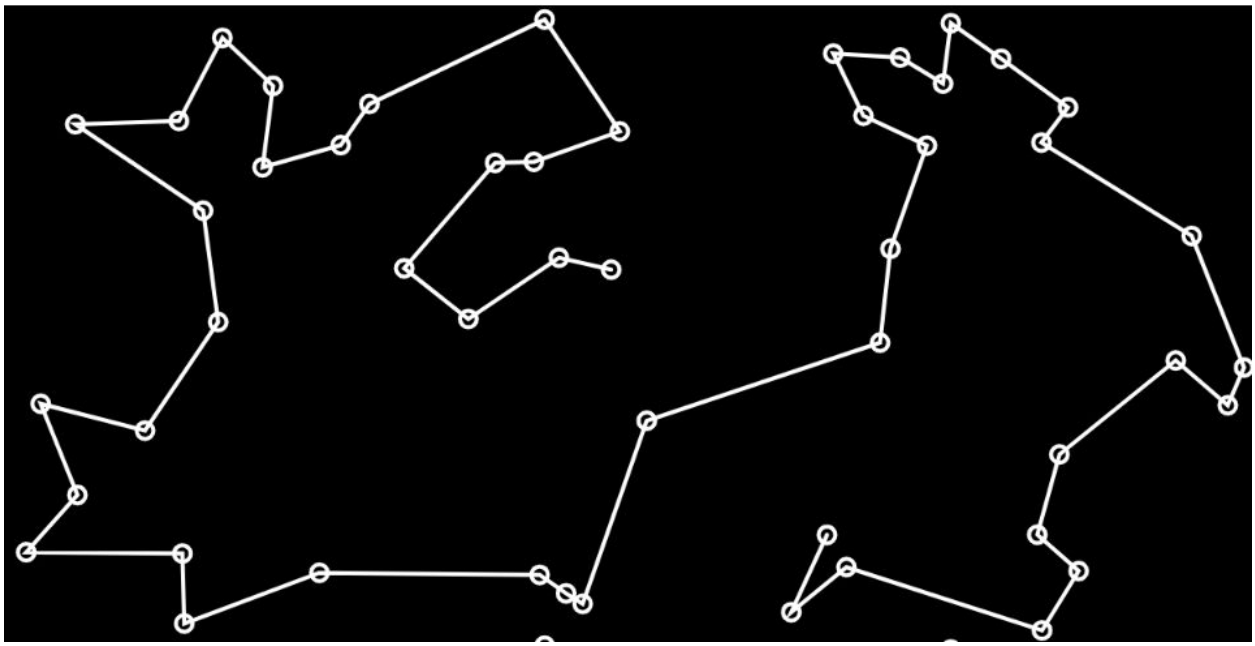
We found a massive difference in the amount of time taken by the much less efficient brute force method than the genetic algorithm on implementing the two. The solution's nature depends upon the implementation of both the crossover function and mutation functions. We have seen that the particular implementations that we have used lead to optimal results with some analysis.

Time Complexity Analysis:

Algorithm	Complexity	Advantage	Disadvantage
Genetic	O(Kmn)	Best solution using fitness criteria	No optimal solution
Greedy	O(logn)	Fastest	Solution accuracy
Dynamic Programming	O(n^2^n)	Global optimal solution	Expensive for both memory and time.

Results:





Conclusion

Genetic algorithms have contributed to some of the best known traveling salesman problem solutions to date. With our future work, particularly the analysis of our survivor selector, we can aim to produce further improvements. We have ideas on how to influence the mutation factor based on fitness and also different crossover mechanisms.

Overall, it seems that genetic algorithms have proved suitable for solving the traveling salesman problem.

As yet, genetic algorithms have not found a better solution to the traveling salesman problem than is already known, but many of the already known best solutions have been found by some genetic algorithm method also.