

# **PROGRAM STRUCTURE AND ALGORITHMS**

## **FINAL PROJECT - TRAVELLING SALESMAN PROBLEM**

Anushka Mandloi - 002775247

Dattvi Bhatt - 002744359

Vipul Shridhar - 002700991

---

### **AIM**

Our aim is to find the shortest path between the given cities using the Christofides algorithm and its different approaches

### **APPROACH**

We are solving the traveling salesman problem using Christofides Algorithm.

We are using two Tactical approaches and 2 Strategic optimal approaches to solve the problem and find their shortest path.

Tactical approaches used:

Random Swapping

Two-opt optimization

Three-opt optimization

Strategic approaches used:

Simulated Annealing

Ant colony optimization

Genetic Algorithm

# PROGRAM

## DATA STRUCTURES AND ALGORITHMS

We have used various data structures in our project

### 1. ArrayList

ArrayList implements the List interface and is a resizable array. Although it is comparable to an array, it can expand dynamically as more elements are added. It enables random access and is advantageous in situations when it is unknown how long the list will be in advance.

### 2. Stack

Last-in, first-out (LIFO) stack implementation is supported by the subclass of Vector known as Stack. It offers push and pop operations so that pieces can be added to or removed from the top of the stack, accordingly. In addition, there are ways to determine whether the stack is empty and to peek at the top piece without deleting it.

### 3. PriorityQueue

A priority queue that implements the Queue interface is called PriorityQueue. The queue can be expanded, and elements are stored so that the highest priority element is always at the front of the queue. The elements' natural ordering or a Comparator can be used to assess their priority. It is helpful in situations when it is necessary to process elements in the order of priority.

### 4. LinkedList

A linked list that implements the List interface is called LinkedList. It is helpful in situations when it is necessary to add and remove elements often because it can do so more quickly than an ArrayList. It does not, however, permit random access.

### 5. Graphs

A group of nodes (vertices) and the links (edges) connecting them are represented using graph data structures. The graph data structure can be implemented in Java by using different classes including Graph, Digraph, and WeightedGraph.

### 6. Union Find

A data structure called union find is employed to handle disjoint collections. Union and find are its two primary functions. The find operation returns the set representation for a given element, whereas the union operation combines two sets into a single set.

### 7. HashMaps

Key-value pairs are stored in data structures called hashmaps. For fundamental operations like adding, removing, and element finding, it offers constant time complexity. The HashMap class in Java is used to implement hash maps.

### 8. Sets

Data structures called sets are used to store collections of unique elements. Java offers numerous Set interface implementations, including HashSet, LinkedHashSet, and TreeSet. While TreeSet is an ordered set, HashSet and LinkedHashSet are unordered sets.

## ALGORITHM

Christofides Algorithm:

The traveling salesman problem (TSP), a well-known issue in operations research and computer science, can be roughly solved using the Christofides method, a heuristic technique. The algorithm was

created by Nicos Christofides in 1976 and is commonly used in practice because of its effectiveness and simplicity.

The algorithm works as follows:

1. Find the minimum spanning tree (MST) of the given graph.
2. Create a minimum-weight perfect matching for the vertices with an odd degree in the MST.
3. Combine the MST and the perfect matching to form a Eulerian tour.
4. Find a Eulerian tour on the Eulerian graph.
5. Convert the Eulerian tour into a Hamilton tour by shortcutting the repeated vertices.

Advantages of Christofides Algorithm:

- **Simplicity:** A variety of people can utilize the algorithm because it is reasonably simple to comprehend and apply.
- **Low computational complexity:** The approach is relatively effective for small to medium-sized instances of the TSP, with a computational complexity of  $O(n^2 \log n)$ .
- **Performance:** Performance is ensured because of the algorithm's worst-case approximation ratio of  $3/2$ , which limits the length of the TSP tour it finds to no more than 1.5 times the length of the ideal TSP tour.
- **Widely used:** Frequently used as a starting point for more complex algorithms, the Christofides algorithm is a well-liked and commonly used approach for approximating the TSP.
- **Good performance in practice:** A successful practice performance: Despite the algorithm's worst-case approximation ratio of  $3/2$

Uses of Christofides Algorithm:

Numerous practical uses for the Christofides algorithm exist, such as in logistics, transportation, and circuit design.

1. For instance, the TSP can be used in circuit design to reduce the wiring length on a printed circuit board

2. It is used in the logistics industry to optimize the delivery routes for a fleet of trucks

Even for large-scale instances, the Christofides method can be utilized to solve these issues effectively, which can increase productivity and lower costs.

Different Approaches to Optimize Christofides Algorithm:

Tactical Optimization Scheme:

1. Two-opt Optimization

The quality of the TSP tour produced by the Christofides algorithm can be further enhanced by the use of the two-opt local search optimization technique. The goal of this method is to find pairs of edges in the tour that can be swapped out for two other edges to produce a shorter tour.

The steps followed in two-opt optimization are:

1. Implement Christofides algorithm to obtain the initial tour
2. Identify two edges in the tour that form a crossing, i.e., two edges that intersect
3. Remove the two edges from the tour, and replace them with two new edges that connect the endpoints of the removed edges in a way that results in a shorter tour
4. Repeat steps 2-3 for all possible pairs of edges until no further improvement is possible

5. If the final tour is shorter than the initial tour, accept the final tour as the current best tour. If not, keep the initial tour as the best tour

The number of iterations completed and the size of the TSP instance determine how time-consuming the two-opt optimization strategy for TSP is.

The temporal complexity of the two-opt optimization strategy is  $O(k \cdot n^3)$ , where  $n$  is the number of cities in the TSP instance, and we assume that we do a fixed number of iterations, say  $k$ .

## 2. Three-opt optimization

The quality of the TSP tour produced by the Christofides algorithm can be further enhanced by using the three-opt local search optimization strategy. The goal of this strategy is to find groups of three edges in the tour that can be swapped out for three other edges to shorten the trip.

The steps followed in three-opt optimization are:

1. Implement Christofides algorithm to obtain the initial tour
2. Identify three edges in the tour that form a crossing, i.e., three edges that intersect
3. Remove the three edges from the tour, and replace them with three new edges that connect the endpoints of the removed edges in a way that results in a shorter tour
4. Repeat steps 2-3 for all possible pairs of edges until no further improvement is possible
5. If the final tour is shorter than the initial tour, accept the final tour as the current best tour. If not, keep the initial tour as the best tour

The temporal complexity of the three-opt optimization strategy is  $O(k \cdot n^4)$ , where  $n$  is the number of cities in the TSP instance, and we assume that we conduct a fixed number of iterations, say  $k$ , and we evaluate all possible sets of three edges in each iteration.

## Strategic Approaches

### 1. Simulated Annealing

Simulated annealing is a metaheuristic optimization approach that can be used to enhance the Christofides algorithm's TSP tour further. Simulated annealing is intended to mimic the physical annealing process, in which a material is heated and then gradually cooled to achieve a low-energy state. The annealing method is used in optimization to bypass local optima and broaden the scope of the search area.

The steps followed in the Simulated Annealing approach are:

1. Implement Christofides algorithm to obtain the initial tour
2. Set an initial temperature  $T$ , and a cooling schedule that determines how the temperature is reduced over time
3. Perform a series of following steps in various iterations
  - Make a random adjustment to the current TSP trip, such as switching two cities or going backward on a section of the route
  - Calculate the length of the new tour
  - If the new tour is shorter than the old one then accept the new tour
  - If the new tour is longer than the old tour, accept it with a probability determined by the Metropolis criteria
4. Repeat step 3 for a fixed number of iterations, or while the temperature reaches a minimum value. If the final tour is shorter than the old one then accept it otherwise keep the old one

Comparing simulated annealing to local search methods like two-opt and three-opt, its main benefit is its ability to circumvent local optima and explore a larger portion of the search space. However,

the choice of the beginning temperature and the rate of cooling determines the quality of the solutions produced by simulated annealing.

The time complexity of simulated annealing for TSP is often moderate and depends on the size of the TSP instance and the desired level of optimization

## 2. Genetic Algorithm

The Christofides approach for the traveling salesman problem (TSP) is frequently optimized using the genetic algorithm optimization method. To identify the ideal TSP tour, genetic algorithm optimization essentially imitates the processes of natural selection and evolution.

The steps followed in the Genetic Algorithm approach are:

1. Implement Christofides algorithm to obtain initial tour
2. Create a population of random TSP tours, where each is represented by chromosome
3. Evaluate the fitness of each chromosome in the population by calculating the length of the corresponding TSP tour
4. Select a chromosome from the population based on its fitness
5. Create new chromosomes by combining the genes of two parent chromosomes from a selected population
6. Mutate a chromosome by swapping two genes
7. Create a new population by selecting parents, performing crossover and mutation
8. Obtain a new tour, if it is shorter than the old one then accept it otherwise keep the old tour

## 3. Ant colony Optimization

It is a metaheuristic algorithm used to optimize the solution obtained from the Christofides algorithm

The steps followed in the ant colony optimization approach are:

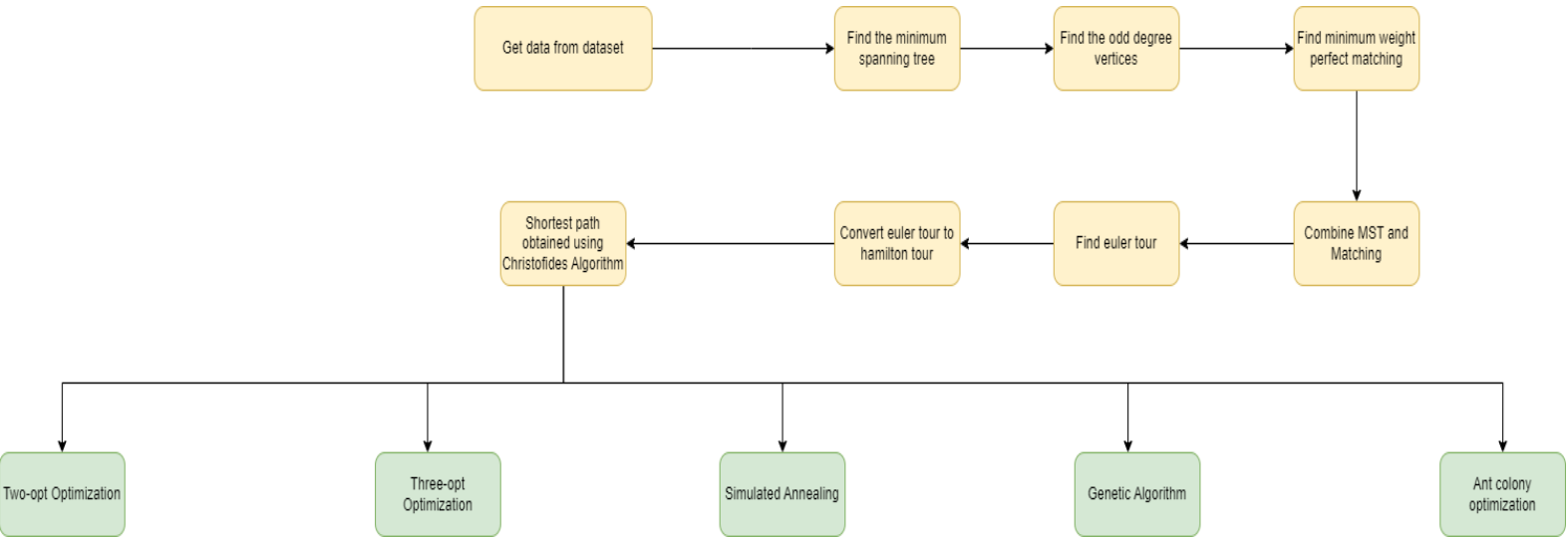
1. Implement the Christofides algorithm to obtain initial tour
2. Initialize the ant colony by creating a set of artificial ants, each representing a possible tour
3. Find a tour using pheromone trails - at each step each ant chooses the next city to visit based on the pheromone trails left by the ants. The probability of choosing a city depends on the amount of pheromone on the edge connecting the current city to the candidate city
4. After all, the ants have completed their tours, the pheromone trails on each edge are updated based on the quality of the tours. Edges that are part of the shorter tours receive more pheromones, whereas edges that are part of longer tours receive fewer pheromones.
5. The ant colony repeats the process of selecting tours and updating pheromone trails for a fixed number of iterations or until a termination criterion is met
6. Finally, a local search procedure such as two-opt or three-opt can be applied to the best tour founded by the ant colony in order to further improve its quality

## INVARIANTS

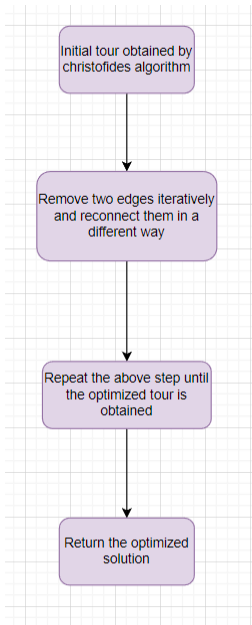
1. The distance between the two nodes is the same in both directions. For example, the distance from point A to B is the same as the distance from point B to A
2. The shortest distance between two points is the direct edge between them. For example, the shortest distance between points A to B will be the direct distance between them rather than other paths

3. The degree of each node in the MST is either 1 or 2 i.e. each node has only one or two edges.
4. (Minimum weight perfect matching) The approach creates a minimum-weight perfect match for each of the nodes in the MST with an odd degree. This group of edges has the smallest overall weight and covers all nodes with odd degrees.

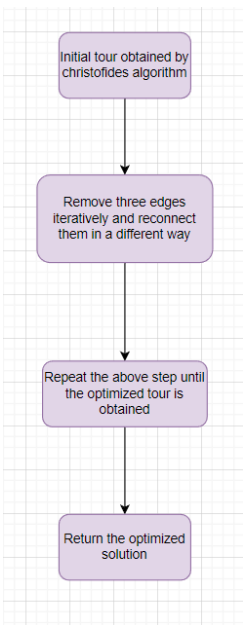
# FLOW CHARTS



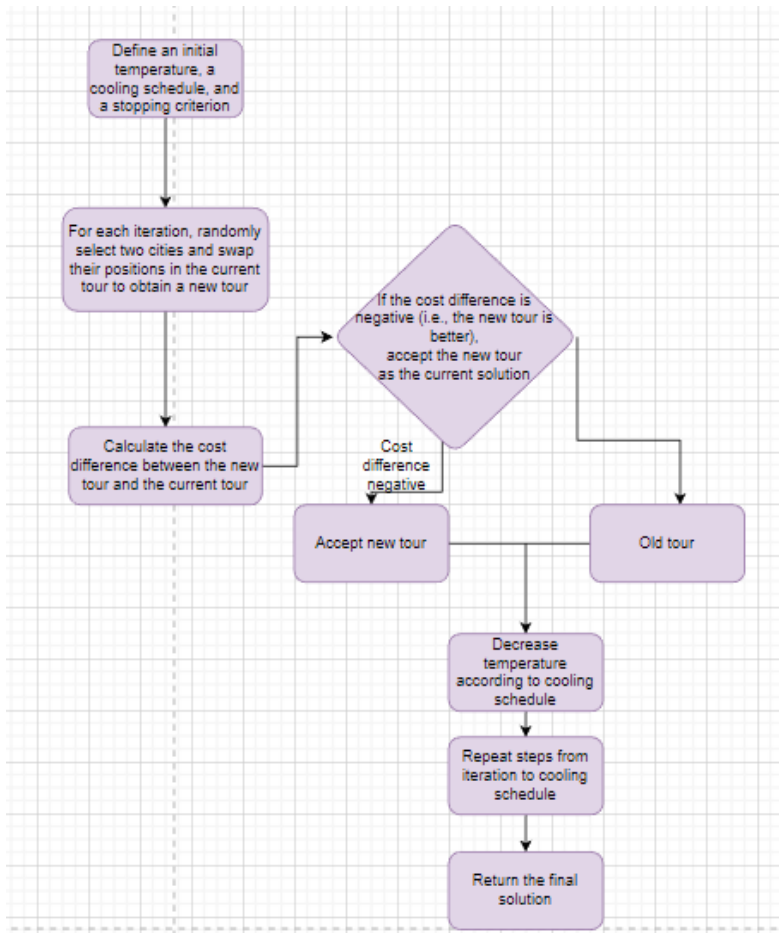
## Two-opt optimization



## Three-opt optimization

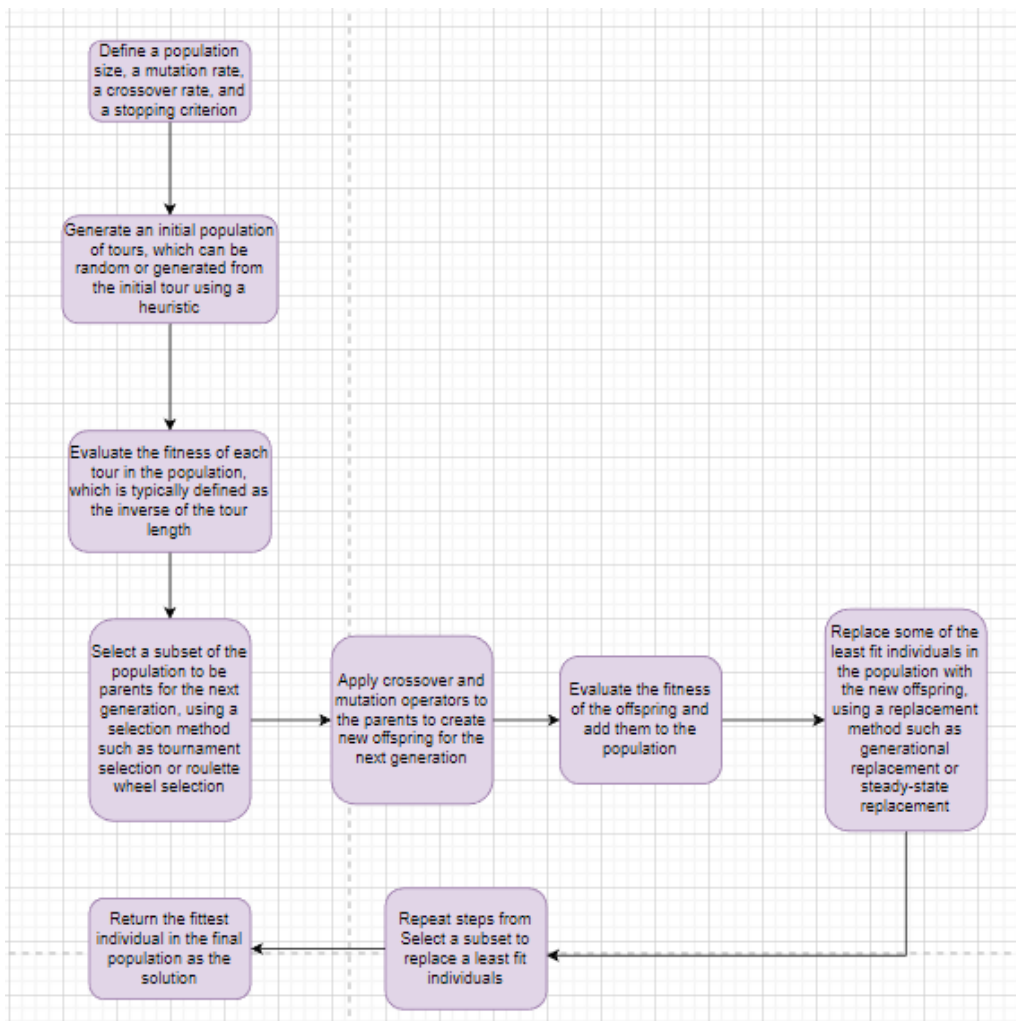


## Simulated Annealing

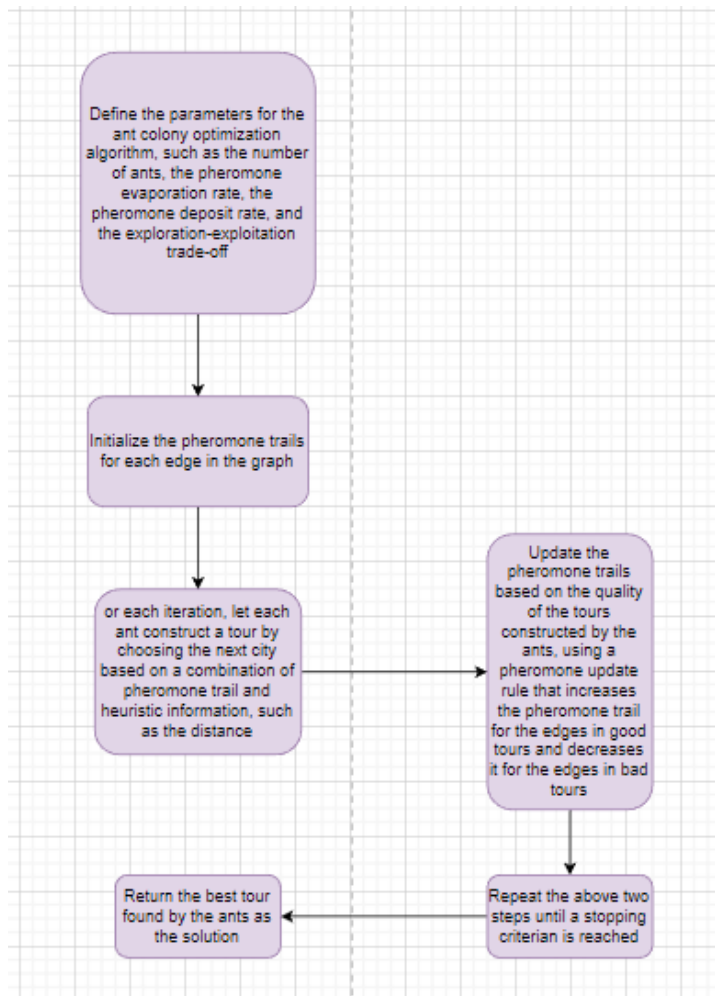




# Genetic Algorithm



## Ant colony optimization



# OBSERVATIONS AND GRAPHICAL ANALYSIS

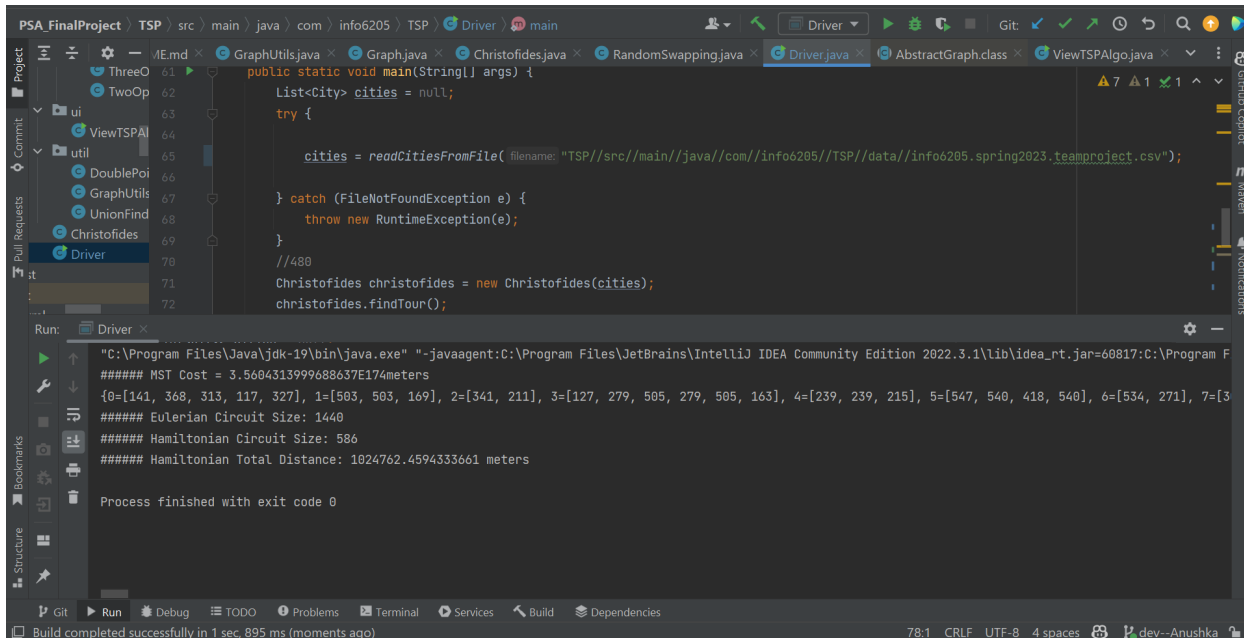
It is difficult to determine which optimization technique is the best for finding the shortest path for the traveling salesman problem.

Different optimization techniques have different approaches and their executions vary on the basis of input size, the structure of the data, the number of iterations, and specific implementation details.

For small to medium-sized data sets, two-opt and three-opt optimizations are extremely effective. They operate by gradually eliminating two edges (two opt) and three edges (three opt) from the path and rejoining it in a way that reduces the overall distance.

Simulated Annealing can be effective for small to medium datasets, as it works by swapping two cities iteratively, and accepting swaps if there is an improvement in the distance or path length.

Ant colony optimization and Genetic algorithms are effective for larger datasets, as they create and evaluate multiple solutions simultaneously and use their feedback to refine the search over time. They can be computationally expensive and require more time to converge to an optimal solution. The effectiveness of these optimization strategies can vary depending on the particular problem instance, thus it may be essential to test out a few different ones to see which one performs the best for a given data collection.

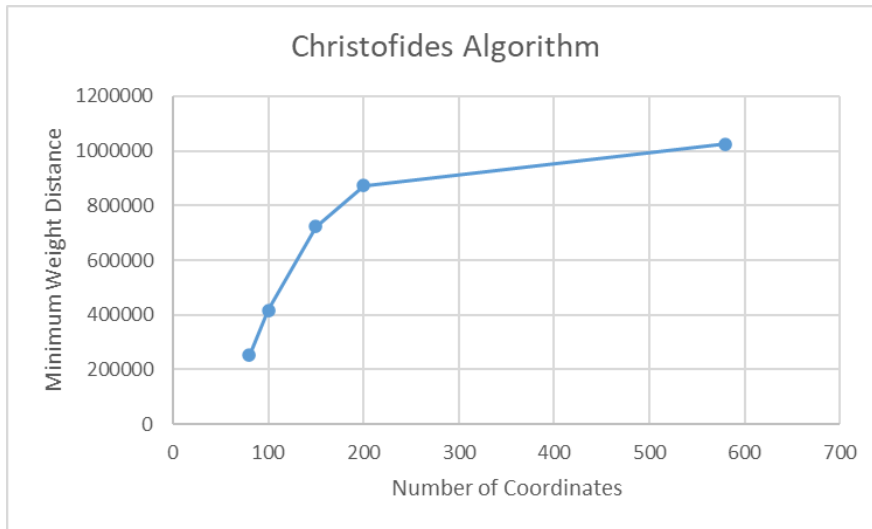


```
public static void main(String[] args) {
    List<City> cities = null;
    try {
        cities = readCitiesFromFile("TSP/src/main/java/com/info6205/TSP/data/info6205.spring2023.teamproject.csv");
    } catch (FileNotFoundException e) {
        throw new RuntimeException(e);
    }
    //480
    Christofides christofides = new Christofides(cities);
    christofides.findTour();
}
```

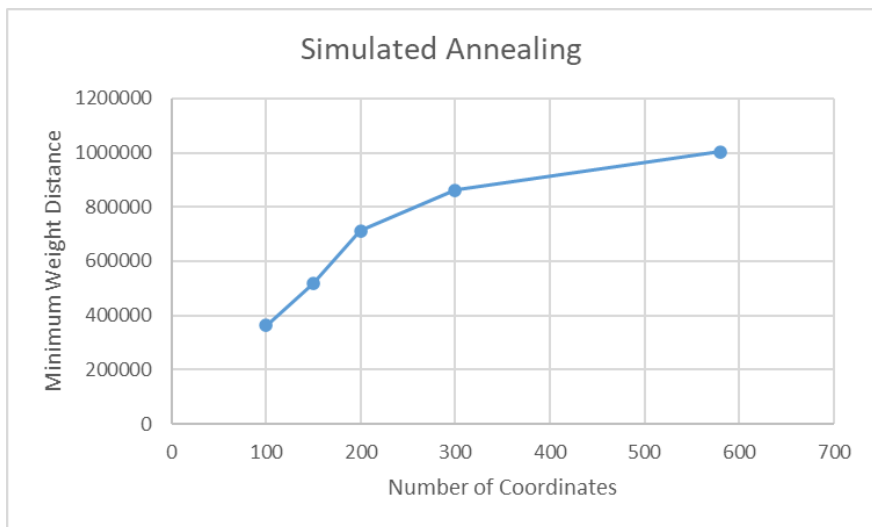
```
Run: Driver
"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2022.3.1\lib\idea_rt.jar=60817:C:\Program F
##### MST Cost = 3.5604313999688637E174meters
{0=[141, 368, 313, 117, 327], 1=[503, 503, 169], 2=[341, 211], 3=[127, 279, 505, 279, 505, 163], 4=[239, 239, 215], 5=[547, 540, 418, 540], 6=[534, 271], 7=[3
##### Eulerian Circuit Size: 1440
##### Hamiltonian Circuit Size: 586
##### Hamiltonian Total Distance: 1024762.4594333661 meters

Process finished with exit code 0
```

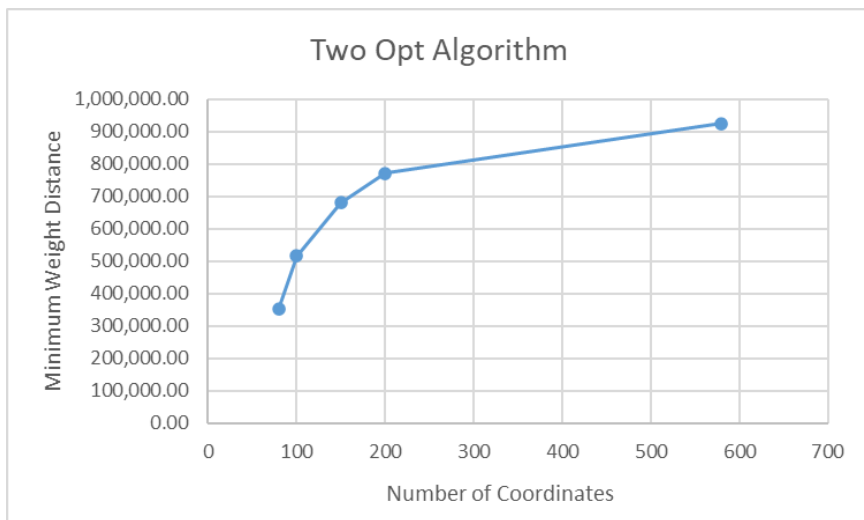
## Graphical analysis of Christofides algorithm



## Graphical analysis of Simulated Annealing



## Graphical analysis of Two opt optimization



## RESULTS AND MATHEMATICAL ANALYSIS

By implementing and executing various optimization techniques we have observed that to find the distance between two cities we use the Haversine formula. As we are looking for the distance between two cities on Earth, therefore we consider the radius and other aspects of the sphere to find out the values.

The formula is based on the haversine law, which asserts that the haversine of an angle's central portion equals the sum of its two side haversines. In the case of the Earth, the sides of the angle are the radii of the planet passing through each of the two spots, and the central angle is the angle formed by two points on the planet's surface.

The Haversine formula is given by:

$$\begin{aligned}a &= \sin^2(\Delta\text{lat}/2) + \cos(\text{lat1}) * \cos(\text{lat2}) * \sin^2(\Delta\text{lon}/2) \\c &= 2 * \text{atan2}(\sqrt{a}, \sqrt{1-a}) \\d &= R * c\end{aligned}$$

where  $\Delta\text{lat}$  and  $\Delta\text{lon}$  are the differences in latitude and longitude between the two points,  $R$  is the radius of the Earth, and  $d$  is the distance between the two points.

The Haversine formula is frequently used in geolocation applications and is particularly helpful for figuring out how far apart two cities are or other points on the surface of the Earth.

In our project we have used the `distance()` method to find the distance between two cities. Two City objects that each contain a city's latitude and longitude coordinates are provided as input parameters for the method. The Haversine formula, which takes into account the curvature of the Earth, is used to determine the distance between the two cities. The angular separation between the two cities is converted by the procedure to a distance in meters, which is then returned as the output, using the earth radius value.

The Haversine formula, which is the formula employed in the code, is frequently used to determine the separation between two places on the surface of the Earth. The output can be changed to kilometers or miles by modifying the earth radius parameter. The distance is determined in meters.

**Heuristic Solution:** Heuristic algorithms, such as the Nearest Neighbor or Greedy algorithms, provide approximate solutions by making locally optimal choices at each step without considering the global optimal solution. Heuristic solutions can be obtained quickly, but they may not always be optimal or near-optimal.

**Minimum Spanning Tree (MST) Solution:** The MST is a tree that spans all vertices of the TSP graph with the minimum total edge weight. It can be obtained using Kruskal's or Prim's algorithm. The MST solution is a suboptimal solution for TSP, as it does not consider the requirement of returning to the starting vertex, which is a key constraint.

**Christofides Solution:** The Christofides algorithm is a heuristic algorithm that guarantees a solution with a performance ratio of at most  $3/2$  times the optimal solution. It combines the construction of an MST with additional steps to obtain a valid tour that satisfies the requirement of returning to the starting

vertex. The Christofides solution is generally better than a simple MST solution, as it provides a guaranteed bound on the quality of the solution.

**Mathematical Analysis:** The performance ratio of Christofides algorithm, which guarantees a solution at most  $3/2$  times the optimal solution, has been proven mathematically. In contrast, heuristic solutions and MST solutions do not have such performance guarantees. Heuristic solutions may provide fast but suboptimal results, while MST solutions may not satisfy the requirement of returning to the starting vertex. Christofides solution, on the other hand, balances between optimality and computational efficiency, making it a popular choice for TSP.

# UNIT TESTS

## Christofides Algorithm

The screenshot shows an IDE with the `ChristofidesTest.java` file open. The code defines a `ChristofidesTest` class with a `testCalculateTourDistance` method. The test results pane shows that 4 out of 4 tests passed in 131 ms. The test output includes MST Cost, Eulerian Circuit Size, Hamiltonian Circuit Size, and Simulated Annealing Distance for two different city sets.

```
package com.info6205.TSP.algorithms;

import ...

public class ChristofidesTest {
    List<City> cities;
    private Christofides christofides;
    @Test
    public void testCalculateTourDistance() {
        // ...
    }
}
```

Run: ChristofidesTest

Tests passed: 4 of 4 tests - 131 ms

ChristofidesTest (com.info6205) 131 ms

- testCalculateTourDistance 77 ms
- findTour 39 ms
- findTour1 14 ms
- testCalculateTourDistance1 1 ms

##### MST Cost = 778296.742688982meters  
{0=[1, 3], 1=[0, 2], 2=[3, 1], 3=[2, 0]}

##### Eulerian Circuit Size: 8

##### Hamiltonian Circuit Size: 5

##### Hamiltonian Total Distance: 444762.77062250266 meters  
Simulated Annealing Distance: 444762.77062250266 meters

##### MST Cost = 3.6233179191440746E7meters  
{0=[1, 4], 1=[2, 0], 2=[3, 1], 3=[4, 2], 4=[3, 0]}

##### Eulerian Circuit Size: 10

##### Hamiltonian Circuit Size: 6

##### Hamiltonian Total Distance: 1.940178417227404E7 meters  
Simulated Annealing Distance: 1.940178417227404E7 meters

##### MST Cost = 563080.1789229582meters  
{0=[1, 2], 1=[2, 0], 2=[1, 0]}

##### Eulerian Circuit Size: 6

##### Hamiltonian Circuit Size: 4

## Graph Utils

The screenshot shows an IDE with the `GraphUtilsTest.java` file open. The code defines a `GraphUtilsTest` class with a `testGetMinimumSpanningTree` method. The test results pane shows that 6 out of 6 tests passed in 120 ms. The test output includes MST Cost, Eulerian Circuit Size, Hamiltonian Circuit Size, and Simulated Annealing Distance for two different city sets.

```
package com.info6205.TSP.algorithms;

import ...

public class GraphUtilsTest {
    List<City> cities;
    private GraphUtils graphUtils;
    @Test
    public void testGetMinimumSpanningTree() {
        // ...
    }
}
```

Run: GraphUtilsTest

Tests passed: 6 of 6 tests - 120 ms

GraphUtilsTest (com.info6205) 120 ms

- testGetMinimumSpanningTree 97 ms
- testGetMinimumWeightPerfect 13 ms
- testGetOddDegreeNodesWith 2 ms
- testGetOddDegreeNodes() 1 ms
- testGetMinimumWeightPerfect 1 ms
- testGraphConstructorWithValid 6 ms

## CONCLUSION

With a polynomial time complexity of  $O(n^3)$  and a good approximation solution for the issue, the Christofides algorithm for the TSP is concluded to be effective.  $n$  is the number of cities. Larger datasets may not have the best solution when using this approach, which performs well for small to medium-sized datasets. But in a fair amount of time, it can still offer a reliable approximation.

Additionally, we can enhance the result of the Christofides algorithm by using different optimization techniques such as two opt, three opt, simulated annealing, ant colony optimization, etc. These optimization methods offer an ideal or very close-to-optimal solution by drastically reducing the length of the journey.

As a result, we may solve the TSP using the Christofides method as a solid starting point and then use various optimization approaches to reach an optimal or nearly optimal solution. The NP-hardness of the problem may prevent the derived solution from always being the best, but the algorithms employed can still produce efficient and useful solutions for practical needs.

Below are the algorithms written from left to right in efficiency or optimization applied for solving TSP  
Left most is the most optimized and right most is the lest

Ant colony > Simulated annealing > Two opt optimization > Random swapping > Christofides algorithm

## REFERENCES

[https://en.wikipedia.org/wiki/Christofides\\_algorithm](https://en.wikipedia.org/wiki/Christofides_algorithm)

<http://www.cs.cornell.edu/courses/cs681/2007fa/Handouts/christofides.pdf>

<https://bochang.me/blog/posts/tsp/>

<http://matejgazda.com/christofides-algorithm-in-python/>

[https://en.wikipedia.org/wiki/Christofides\\_algorithm](https://en.wikipedia.org/wiki/Christofides_algorithm)

<https://www.youtube.com/watch?reload=9&v=GiDsJlBOVoA&feature=youtu.be>