# INFO 6205 Spring 2023 Project *Traveling Salesman*

Team Details:

Member 1: Dinesh Shekhawat (NUID - 002776484)

Member 2: Dhruv Patel (NUID - 002928881)

Member 3: Harshil Shah (NUID - 002780887)

GitHub Repo - <https://github.com/pateldhruvr/INFO6205-Final-Project/>

## Aim

Given a list of geographical points represented as latitudes & longitudes, generate the shortest possible route that visits each point exactly once and returns to the point of origin. Use Christofides to generate a candidate solution and apply 4 optimization strategies - two tactical and two strategic techniques to get the shortest possible tour.

## Approach

1. Get the lower bound estimate using MST Cost and further get more improved lower bound using max One-tree cost
2. Build a candidate tour using Christofides Algorithm
3. Use candidate tour obtained from Christofides and apply the following optimization strategies:-
4. 2-opt. Heuristic (Tactical)- Randomly swap two nodes for a specified time interval. If an improvement is found in the tour cost, update the tour and continue. Once the time is exhausted, switch the algorithm by looking for improvements by swapping two adjacent nodes in the tour. Update the tour if an improvement is found in the tour distance.
5. 3-opt. Heuristic (Tactical)- Same as 2-opt, but check all the combinations obtained from the selected three nodes and choose the one that gives the least tour distance during each swap.
6. Ant Colony Optimization (Strategic)- Simulate the behavior of ants by leaving pheromone trails over short paths, thus biasing ant movement towards areas of the problem space that have high pheromone concentrations, leading to a more optimized tour.
7. Simulated Annealing (Strategic)- Explore the solution space that gets ignored due to the local minimum trap using 2opt/3opt, which may lead to a more optimized solution
8. Mix and Match (Strategic + Tactical)- Apply different optimization techniques on top of each other to get the best possible shortest tour

## Program

1. Christofides
2. Data Structures
3. Heap (Min Indexed Priority Queue)[2]
4. Stack
5. HashMap
6. LinkedList
7. Graph
8. Classes
9. HaversineDistanceUtil {}
10. Point(String id, double latitude, double longitude) {}
11. Edge(Point from, Point to) {}
12. PrimsMST(List<Point> nodes) {}[3]
13. OneTree(int n) {}
14. KolmogorovWeightedPerfectMatchingImpl implements PerfectMatchingSolverService {}[4][5]
15. FluerysAlgorithm(int numOfVertices){}[6]
16. Christofides(List<Point> points) {}
17. Algorithm[1]
18. Create a minimum spanning tree T of G
19. Let O be the set of vertices with odd degree in T. By the handshaking lemma, O has an even number of vertices.
20. Find a minimum-weight perfect matching M in the induced subgraph given by the vertices from O
21. Combine the edges of M and T to form a connected multigraph H in which each vertex has even degree
22. Form an Eulerian circuit in H
23. Make the circuit found in previous step into a Hamiltonian circuit by skipping repeated vertices (shortcutting)
24. Invariants
25. Complete graph - All the vertices in the graph should be connected to each other
26. The triangle inequality - the distance between any two nodes is always less than or equal to the sum of the distances between those nodes and any other node
27. The graph must have an even number of odd-degree nodes, or else a perfect matching cannot be found

2. Ant Colony Optimization

1. Data Structures
2. 2D Array
3. Classes
4. AntColonyOptimization(

double[][] graph,

int[] christofidesTour,

int numberOfAnts,

double phermoneExponent,

double heuristicExponent,

double phermoneEvaporationRate,

double phermoneDepositFactor,

int numberOfIterations,

int maxImprovementIterations) {}

1. Algorithm
2. Initialize the parameters:

* N: number of ants
* α: trail factor
* β: heuristic factor
* ρ: evaporation rate
* Q: pheromone deposit factor
* t0: initial pheromone trail
* tau: pheromone trail matrix
* h: heuristic information matrix
* s: current solution (Christofides’s solution)

1. Initialize the pheromone trail matrix tau and the heuristic information matrix h.
2. Repeat for each iteration:

* Generate solutions using the following steps:
* Place each ant at a random starting point
* Move the ant to the next city based on the probabilities calculated using the pheromone trail matrix and the heuristic information matrix
* Update the pheromone trail matrix using the Q value and the length of the tour
* Update the pheromone trail matrix using the following steps:
* Evaporate the pheromone trail using the evaporation rate
* Add the pheromone deposit to the edges visited by the ants
* Update the best solution found so far
* Terminate if the stopping criterion is met

1. Return the best solution found.

1. Invariants
2. Parameter settings: ACO has several parameters that need to be set before the algorithm can be run. These include the number of ants, the pheromone update rate, and the importance of the heuristic information.
3. Pheromone trail: ACO uses a pheromone trail to communicate information between the ants. The pheromone trail is updated by the ants based on the quality of the solutions they find.
4. Ant behavior: The behavior of the ants is a key aspect of ACO. Each ant follows a set of rules to decide which path to take in the search space. These rules are often based on the pheromone trail and the heuristic information.

3. Simulated Annealing

1. Data Structures
2. 2D Array
3. Classes
4. SimulatedAnnealingOptimization(

int[] christofidesTour,

double[][] distanceMatrix,

int maxIteration,

double startingTemperature,

double finalTemperature,

double coolingRate) {}

1. Algorithm[7]
2. current ← problem.INITIAL-STATE
3. for t = 1 to MAX\_ITERATIONS do

* T ← schedule(t)
* if T = 1 then return current
* next ← a randomly selected successor of current
* ΔE ← VALUE(next) - VALUE(current)
* if ΔE > 0 then current ← next
* else current ← next only with probability eΔE/T

1. Invariants
2. Temperature Schedule: The temperature schedule used in the Simulated Annealing algorithm must satisfy certain properties, such as decreasing over time and converging to zero. The schedule must be carefully chosen to balance exploration and exploitation of the search space.
3. Probability Distribution: The probability distribution used to accept or reject candidate solutions must be valid, meaning that it must sum to 1 and be non-negative for all inputs.

Flowchart (inc. UI Flow)

## Observations & Graphical Analysis

## Result and Mathematical Analysis

## Unit Tests

## Conclusion

## References

[1] "Christofides algorithm," Wikipedia. [Online]. Available: <https://en.wikipedia.org/wiki/Christofides_algorithm>.

[2] W. Fiset, "MinIndexedDHeap.java," GitHub. [Online]. Available: <https://github.com/williamfiset/Algorithms/blob/master/src/main/java/com/williamfiset/algorithms/datastructures/priorityqueue/MinIndexedDHeap.java>.

[3] W. Fiset, "EagerPrimsAdjacencyList.java," Github repository file, 2020. [Online]. Available: <https://github.com/williamfiset/Algorithms/blob/master/src/main/java/com/williamfiset/algorithms/graphtheory/EagerPrimsAdjacencyList.java>.

[4] KolmogorovWeightedMatching.java, GitHub repository, Jgrapht. [Online]. Available:<https://github.com/jgrapht/jgrapht/blob/master/jgrapht-core/src/main/java/org/jgrapht/alg/matching/blossom/v5/KolmogorovWeightedMatching.java>.

[5] V. Kolmogorov, "Blossom V: a new implementation of a minimum cost perfect matching algorithm," Mathematical Programming Computation, vol. 1, no. 1, pp. 43-67, 2009.

[6] GeeksforGeeks, "Fleury’s Algorithm for printing Eulerian Path," [Online]. Available:<https://www.geeksforgeeks.org/fleurys-algorithm-for-printing-eulerian-path/>.

[7] "AIMA Pseudocode," GitHub repository, 2023. [Online]. Available: <https://github.com/aimacode/aima-pseudocode/blob/master/md/Simulated-Annealing.md>.