

INFO 6205 Spring 2023 Project

Project-Traveling Salesman

Submitted by:

Name: Mulpuri Uday Kiran Reddy **NUID:**002781063
Nandre Anurag **NUID:**002785735

Aim:

The aim of the project is to solve the Travelling Salesman Problem, which is an NP-hard problem, using the Christofides algorithm and optimizing further with tactical and strategic optimizations

Approach:

- The first step of the project involved processing the given dataset (Data\tspdatapoints.xlsx) into the necessary format (points.xlsx). Subsequently, we computed the distance between each vertex in terms of edges and recorded the results in a separate file (routes.xlsx).
- Next, we utilized Kruskal's Union Find Method to identify the minimum spanning tree (MST) of the graph. This was achieved by implementing the "minimumSpanningTree()" function within the MinWeightService Class.
- After obtaining the MST, we proceeded to extract the list of odd vertices from the MST Tour.
- The next step involved identifying the perfect matching for the list of odd. This was accomplished by utilizing the method "findMinimumWeightMatching()" present within the MinWeightServiceClass.
- Subsequently, we combined the Perfect Matching and MST to ensure that every vertex in the graph had an even number of edges. This allowed us to generate an Eulerian tour of the graph.
- Using the Eulerian tour obtained in the previous step, we aimed to create a TSP tour that visits each vertex in the graph and returns to the starting vertex. It was necessary to skip over any vertex that had already been encountered during the tour.
- In parallel with the previous step, we calculated the distance/weight of the TSP tour.
- Next, we performed two tactical optimizations, namely the 2-opt and 3-opt techniques, on the TSP tour obtained in the previous step.
- Furthermore, we applied two strategic optimizations, simulated annealing and ant colony optimization, to further enhance the TSP tour.

Algorithm:

Kruskal's Algorithm: The Minimum Spanning Tree (MST) algorithm is a commonly used approach to approximate a solution for the Traveling Salesman Problem (TSP). The MST algorithm begins by constructing a complete graph of the cities, and then computes the MST of the graph using Prim's or Kruskal's algorithm. The resulting MST is then traversed to obtain a sequence of cities, which is then converted into a Hamiltonian circuit by adding the starting city to the end. While the MST algorithm does not guarantee an optimal solution for the TSP, it is often used in practice because it can provide a good approximation. By using the MST to connect the cities with the minimum possible total weight, we can ensure that the resulting Hamiltonian circuit is relatively efficient and avoids unnecessary detours. Overall, the MST algorithm can be a useful tool for finding approximate solutions to the TSP in a variety of applications.

Tactical Optimizations:

2 opt:

The 2-opt algorithm is a heuristic optimization algorithm for the Traveling Salesman Problem (TSP) that improves an initial TSP tour generated from the Christofides algorithm. The algorithm starts by choosing two edges i and j in the tour such that the vertices do not share a common edge. It then generates a new tour by adding the elements from index 0 to i , followed by the elements of the original tour from j to i in reverse order, and finally the remaining elements from $j+1$. If the resulting tour is shorter than the original tour, it is accepted as the new tour. Otherwise, the algorithm repeats steps 2-4 for all possible pairs of edges until no improvement is possible. By iteratively applying this algorithm, the TSP tour can be gradually improved until no further improvements can be made. The 2-opt algorithm is a simple yet effective way to optimize TSP tours and is often used as a building block in more advanced optimization algorithms.

3 Opt:

To improve an initial TSP tour, the 3-opt algorithm selects three non-adjacent edges x , y , and z in the tour and generates a new tour by adding the elements from index 0 to x , followed by the elements of the original tour from y to x in reverse order, then the elements from z to y in reverse order, and finally the remaining elements from $z+1$. If the resulting tour is shorter than the original tour, it is accepted as the new tour. If not, the algorithm repeats steps 1 and 2 for all possible combinations of three non-adjacent edges until no improvement is possible. By iteratively applying this algorithm, the TSP tour can be gradually improved until no further improvements can be made. The 3-opt algorithm is a more advanced version of the 2-opt algorithm and is often used as a building block in more sophisticated optimization algorithms.

Strategic Optimizations:

Simulated Annealing:

Simulated annealing is a heuristic optimization algorithm for the Traveling Salesman Problem (TSP). It works by iteratively generating candidate solutions through 2-opt optimization, starting from an initial tour. If a new candidate solution is better than the current one, it is always accepted as the new solution. However, if the new solution is worse, it may still be accepted with a probability that decreases over time. This probability is determined by a temperature parameter, which controls the degree of randomness in the search process. As the algorithm progresses, the temperature parameter gradually decreases over iterations proportional to a cooling rate. This enables the algorithm to converge towards a better solution, while still allowing it to explore less promising solutions early on. By doing so, simulated annealing can discover TSP tours that are better than the tour generated by the Christofides algorithm, which is commonly used as a benchmark for TSP solvers.

Ant Colony Optimization

Ant colony optimization is a metaheuristic algorithm that uses a population of artificial ants to search for good solutions to a problem. For the Traveling Salesman Problem (TSP), ants construct solutions by iteratively selecting cities based on a pheromone trail that reflects the quality of previously constructed solutions. The pheromone trail is updated based on the quality of the constructed solutions. Ants are more likely to visit cities with a higher pheromone trail, as this indicates a better quality solution. However, to avoid getting trapped in local optima, ants also explore new paths randomly. Over time, the trail is reinforced on the best paths and evaporates on the less attractive paths. This helps the algorithm converge towards a good quality solution by focusing on the most promising paths while avoiding less optimal paths. By doing so, ant colony optimization can discover TSP tours that are better than the tour generated by other metaheuristic algorithms.

Data Structures:

1. Graph

- Adjacency Matrix
- Adjacency List

Used In: MinWeightService, ThreeOptService, TwoOptService, SimulatedAnnealingService, LinKernighan2Opt.

2. Disjoint Set

- Union Find
- Path Compression

Used In: MinWeightService, UnionFind

3. List

Used In: MinWeightService, ThreeOptService, TwoOptService, SimulatedAnnealingService, LinKernighan2Opt, ExcelFetchPointService, ExcelFetchRouteService, CsvUtilService, RouteWeightService

4. Set

Used In: MinWeightService, CsvUtilService

5. Map

Used In: MinWeightService, CsvUtilService

6. Priority Queue

Used In: MinWeightService

7. Arrays

8. Stack

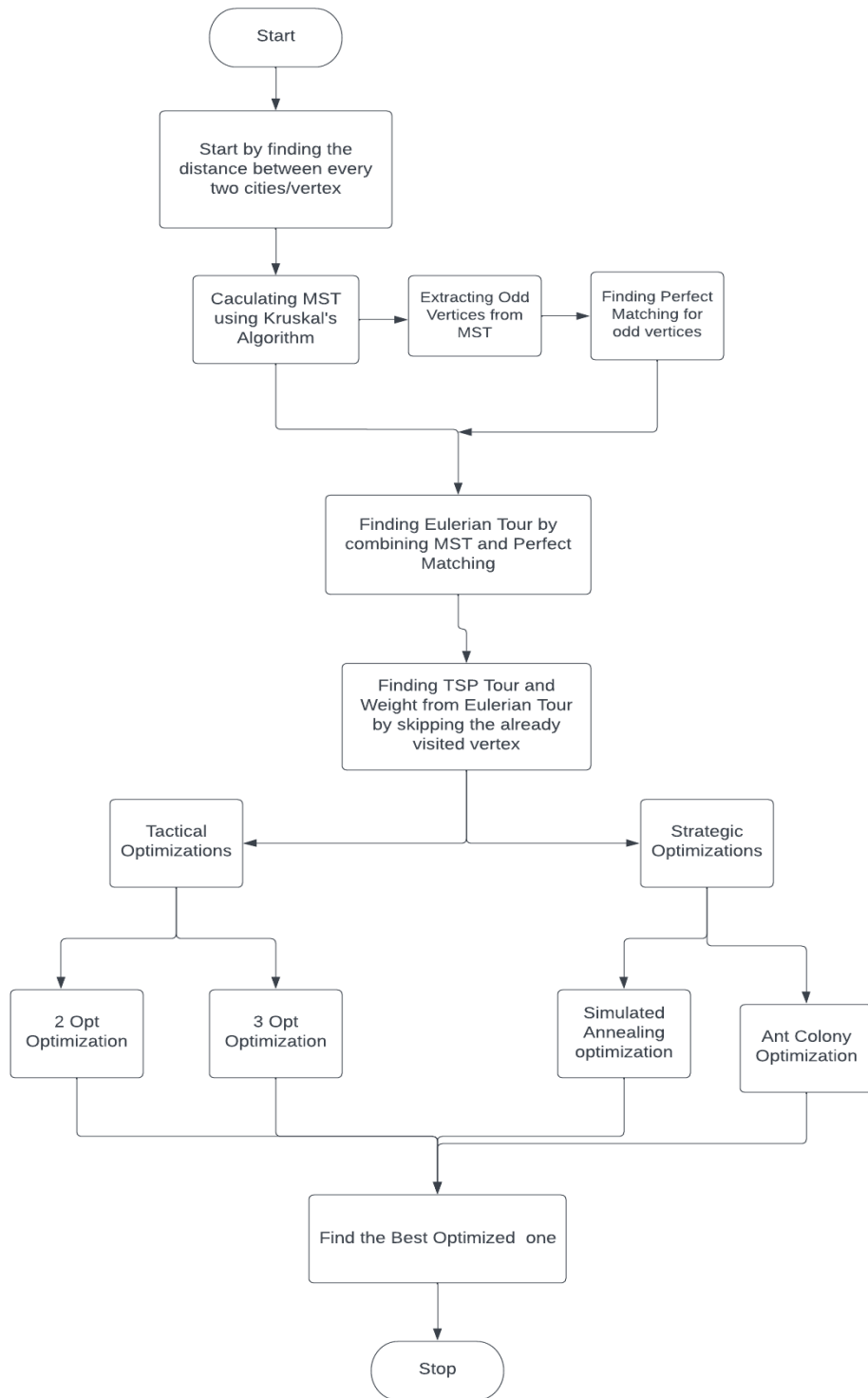
Used In: MinWeightService

Invariants:

- Temperature
- Rate of Cooling

- Iterations

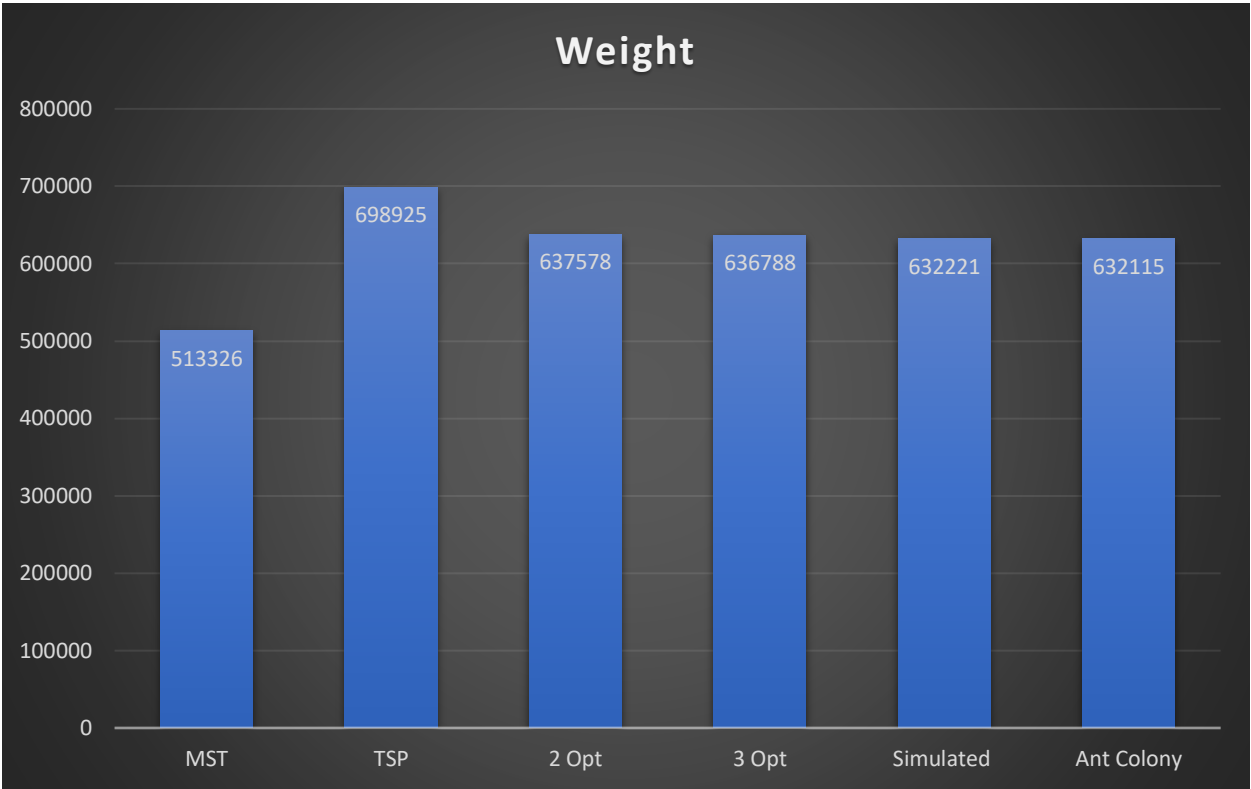
Flowchart:



Observations and Graphical Analysis:

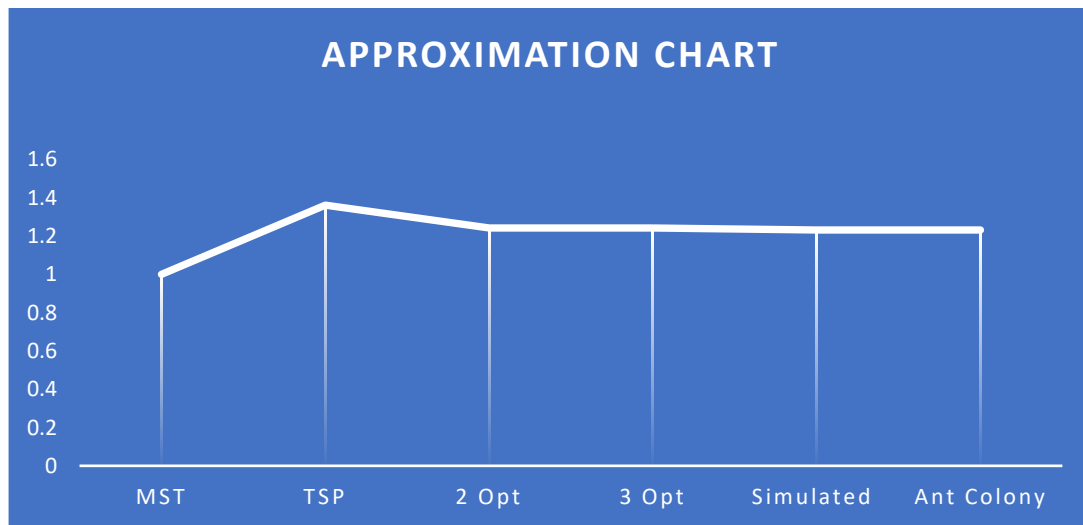
Graph for Distances of MST,TSP and Optimizations:

MST	TSP	2 Opt	3 Opt	Simulated	Ant Colony
513326	698925	637578	636788	632221	632115



Below are the values and Graph for Approximation calculation between MST and optimizations:

MST	TSP	2 Opt	3 Opt	Simulated	Ant Colony
1	1.36156166	1.24205281	1.24051383	1.23161695	1.23141045



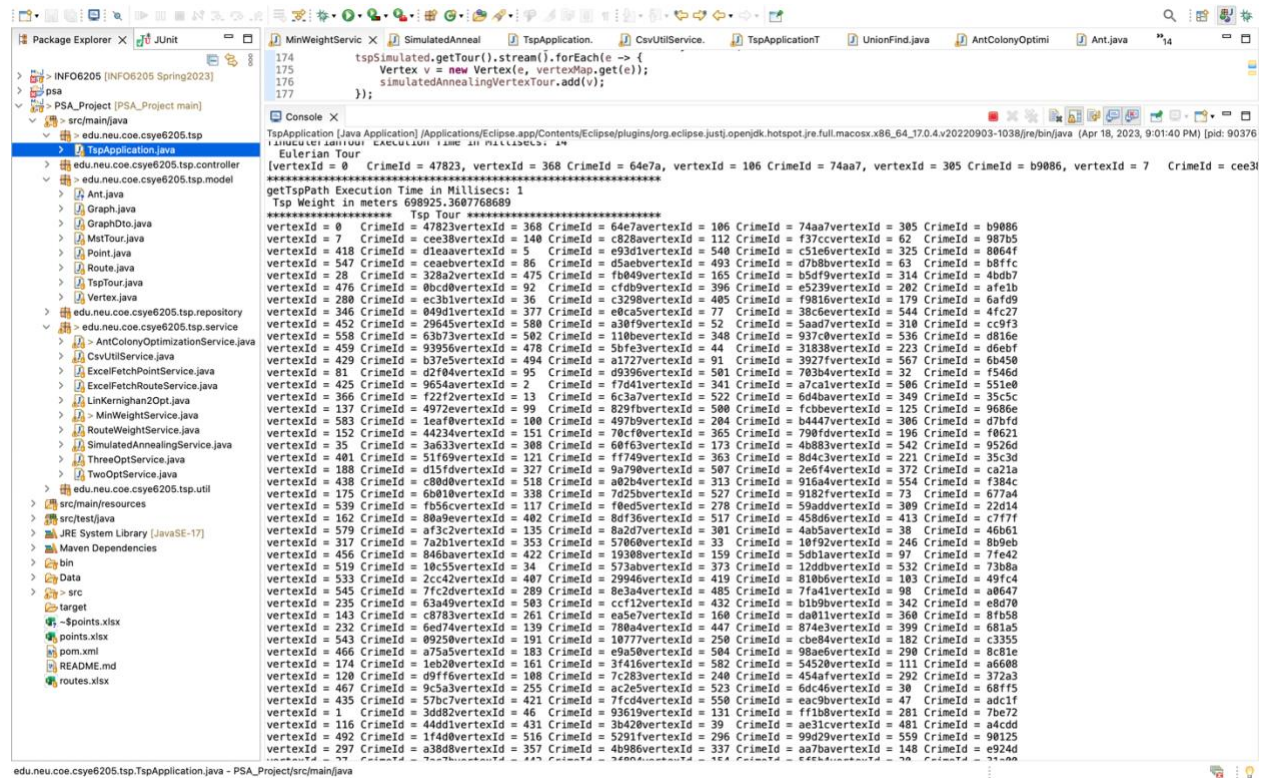
Results and Mathematical Analysis:

MST Tour Weight(meters) = 513326.09539907286

TSP Tour Weight(Meters) = 698925.3607768689

Approx Ratio= TSP/MST=698925.3607768689/513326.09539907286

Approximation Ratio=1.36



```
Package Explorer X JUnit
> INFO6205 [INFO6205 Spring2023]
> psa
> PSA_Project [PSA_Project main]
  > src/main/java
    > edu.neu.coe.csye6205.tsp
      > TspApplication.java
    > edu.neu.coe.csye6205.tsp.controller
    > edu.neu.coe.csye6205.tsp.model
      > Ant.java
      > Graph.java
      > GraphDto.java
      > Point.java
      > Route.java
      > TspTour.java
      > Vertex.java
    > edu.neu.coe.csye6205.tsp.repository
    > edu.neu.coe.csye6205.tsp.service
      > AntColonyOptimizationService.java
      > CsvUtilService.java
      > ExcelFetchPointService.java
      > ExcelFetchRouteService.java
      > LinkKernighan2Opt.java
      > MinWeightService.java
      > SimulatedAnnealingService.java
      > ThreeOptService.java
      > TwoOptService.java
    > edu.neu.coe.csye6205.tsp.util
  > src/main/resources
  > src/test/java
  > JRE System Library [JavaSE-17]
  > Maven Dependencies
  > bin
  > Data
  > src
  > target
    > $points.xlsx
    > points.xlsx
    > pom.xml
    > README.md
    > routes.xlsx

TspApplication [Java Application] [Applications/Eclipse.app/Contents/Eclipse/plugins/org.eclipse.jst.openjdk.hotspot.jre.full.macosx.x86_64.17.0.4.v20220903-1038/jre/bin/java (Apr 18, 2023, 9:01:40 PM) [pid: 90376]
174 tspSimulated.getTour().stream().forEach(e -> {
175     Vertex v = new Vertex(e, vertexMap.get(e));
176     simulatedAnnealingVertexTour.add(v);
177 });
Console X
TspApplication [Java Application] [Applications/Eclipse.app/Contents/Eclipse/plugins/org.eclipse.jst.openjdk.hotspot.jre.full.macosx.x86_64.17.0.4.v20220903-1038/jre/bin/java (Apr 18, 2023, 9:01:40 PM) [pid: 90376]
Eulerian Tour
[vertexId = 0 CrimeId = 47823, vertexId = 368 CrimeId = 64e7a, vertexId = 186 CrimeId = 74aa7, vertexId = 385 CrimeId = b9086, vertexId = 7 CrimeId = cee3f]
getTspPath Execution Time in Milliseconds: 1
Tsp Weight in meters 698925.3607768689
***** Tsp Tour *****
vertexId = 0 CrimeId = 47823vertexId = 368 CrimeId = 64e7avertexId = 186 CrimeId = 74aa7vertexId = 385 CrimeId = b9086
vertexId = 7 CrimeId = cee3fvertexId = 148 CrimeId = c828avertexId = 112 CrimeId = f37ccvertexId = 92 CrimeId = 987b5
vertexId = 418 CrimeId = d1eaavertexId = 5 CrimeId = e93d1vertexId = 540 CrimeId = c51e6vertexId = 325 CrimeId = 8064f
vertexId = 547 CrimeId = ceaebvertexId = 86 CrimeId = d5aebvertexId = 493 CrimeId = d7b8bvertexId = 63 CrimeId = b8ffc
vertexId = 28 CrimeId = 328a2vertexId = 475 CrimeId = f0499vertexId = 165 CrimeId = b5df9vertexId = 314 CrimeId = 4b0b7
vertexId = 476 CrimeId = 0bc0dvertexId = 92 CrimeId = cfd9bvertexId = 396 CrimeId = e5239vertexId = 282 CrimeId = afe1b
vertexId = 280 CrimeId = ec3blvertexId = 36 CrimeId = c3298vertexId = 405 CrimeId = f9816vertexId = 179 CrimeId = 6afdc
vertexId = 346 CrimeId = 049d1vertexId = 377 CrimeId = e0ca5vertexId = 77 CrimeId = 38c6evertexId = 544 CrimeId = 4fc27
vertexId = 452 CrimeId = 29645vertexId = 580 CrimeId = a30f9vertexId = 52 CrimeId = 5aad7vertexId = 310 CrimeId = cc9f3
vertexId = 558 CrimeId = 63073vertexId = 502 CrimeId = 1180evertexId = 348 CrimeId = 937cbvertexId = 536 CrimeId = 0810e
vertexId = 459 CrimeId = 93956vertexId = 478 CrimeId = 5bf63vertexId = 44 CrimeId = 31838vertexId = 223 CrimeId = d6ebf
vertexId = 429 CrimeId = b37e5vertexId = 494 CrimeId = a1727vertexId = 91 CrimeId = 3927fvertexId = 567 CrimeId = 6b450
vertexId = 81 CrimeId = d2f04vertexId = 95 CrimeId = d9396vertexId = 501 CrimeId = 703b4vertexId = 32 CrimeId = f546d
vertexId = 425 CrimeId = 9654avertexId = 2 CrimeId = f7d41vertexId = 341 CrimeId = a7calvertexId = 586 CrimeId = 551e0
vertexId = 366 CrimeId = 72272vertexId = 13 CrimeId = 6c3a7vertexId = 522 CrimeId = 6d4bavertexId = 349 CrimeId = 35c5c
vertexId = 137 CrimeId = 4972evertexId = 99 CrimeId = 829fbvertexId = 500 CrimeId = fcbbevertexId = 125 CrimeId = 9686e
vertexId = 583 CrimeId = 1ea10vertexId = 100 CrimeId = 497b9vertexId = 284 CrimeId = b4447vertexId = 306 CrimeId = d7bfd
vertexId = 152 CrimeId = 44234vertexId = 151 CrimeId = 78cf0vertexId = 365 CrimeId = 798fdvertexId = 196 CrimeId = f0e21
vertexId = 35 CrimeId = 3a633vertexId = 308 CrimeId = 68f63vertexId = 173 CrimeId = 4b883vertexId = 542 CrimeId = 9526d
vertexId = 481 CrimeId = 51f69vertexId = 121 CrimeId = ff749vertexId = 363 CrimeId = 8d4c3vertexId = 221 CrimeId = 35c3d
vertexId = 188 CrimeId = d15fdvertexId = 327 CrimeId = 9a790vertexId = 507 CrimeId = 2e6f4vertexId = 372 CrimeId = ca21a
vertexId = 438 CrimeId = c88d0vertexId = 518 CrimeId = a02b4vertexId = 313 CrimeId = 916a4vertexId = 554 CrimeId = f394c
vertexId = 175 CrimeId = 60010vertexId = 338 CrimeId = 7d25bvertexId = 527 CrimeId = 9182fvertexId = 73 CrimeId = 677a4
vertexId = 539 CrimeId = fb56cvertexId = 117 CrimeId = f0ed5vertexId = 278 CrimeId = 59addvertexId = 309 CrimeId = 22d14
vertexId = 162 CrimeId = 80a9evertexId = 402 CrimeId = 8df36vertexId = 517 CrimeId = 458d6vertexId = 413 CrimeId = c717f
vertexId = 579 CrimeId = af3c2vertexId = 135 CrimeId = 8a207vertexId = 301 CrimeId = 4ab5avertexId = 38 CrimeId = 40b61
vertexId = 317 CrimeId = 7a2b1vertexId = 353 CrimeId = 57860vertexId = 33 CrimeId = 10792vertexId = 246 CrimeId = 809eb
vertexId = 456 CrimeId = 846bavertexId = 422 CrimeId = 19388vertexId = 159 CrimeId = 5db1avertexId = 97 CrimeId = 7fe42
vertexId = 519 CrimeId = 18c55vertexId = 34 CrimeId = 573abvertexId = 373 CrimeId = 12ddbvertexId = 532 CrimeId = 73b8a
vertexId = 533 CrimeId = 2cc42vertexId = 407 CrimeId = 29946vertexId = 419 CrimeId = 81806vertexId = 183 CrimeId = 49f4c
vertexId = 545 CrimeId = 7fc2dvertexId = 289 CrimeId = 8e3a4vertexId = 405 CrimeId = 7f414vertexId = 90 CrimeId = a0647
vertexId = 235 CrimeId = 63a49vertexId = 503 CrimeId = ccf12vertexId = 432 CrimeId = b1b9bvertexId = 342 CrimeId = e8d70
vertexId = 143 CrimeId = c8783vertexId = 261 CrimeId = ea5e7vertexId = 160 CrimeId = da011vertexId = 360 CrimeId = 8fb58
vertexId = 232 CrimeId = 6ed74vertexId = 139 CrimeId = 780a4vertexId = 447 CrimeId = 874e3vertexId = 399 CrimeId = 681a5
vertexId = 543 CrimeId = 09259vertexId = 191 CrimeId = 10777vertexId = 250 CrimeId = cb084vertexId = 182 CrimeId = c3355
vertexId = 466 CrimeId = a75a5vertexId = 183 CrimeId = e9a50vertexId = 504 CrimeId = 98a66vertexId = 290 CrimeId = 8c81e
vertexId = 174 CrimeId = 1eb20vertexId = 161 CrimeId = 3f416vertexId = 582 CrimeId = 54520vertexId = 111 CrimeId = a6608
vertexId = 120 CrimeId = 09f16vertexId = 108 CrimeId = 7c283vertexId = 248 CrimeId = 454afvertexId = 292 CrimeId = 372a3
vertexId = 467 CrimeId = 9c5a3vertexId = 255 CrimeId = ac2e5vertexId = 523 CrimeId = 6d40evertexId = 30 CrimeId = 68f15
vertexId = 435 CrimeId = 57bc7vertexId = 421 CrimeId = 7fcd4vertexId = 550 CrimeId = eac9bvertexId = 47 CrimeId = adc1f
vertexId = 1 CrimeId = 3dd82vertexId = 46 CrimeId = 93619vertexId = 131 CrimeId = f1fb8vertexId = 281 CrimeId = 7be72
vertexId = 116 CrimeId = 44dd1vertexId = 431 CrimeId = 3b420vertexId = 39 CrimeId = ae31cvertexId = 481 CrimeId = a4cdc
vertexId = 492 CrimeId = 1f440vertexId = 516 CrimeId = 5291fvertexId = 296 CrimeId = 99d29vertexId = 559 CrimeId = 80125
vertexId = 297 CrimeId = a38d8vertexId = 357 CrimeId = 4b986vertexId = 337 CrimeId = aa7bavertexId = 148 CrimeId = e924d
vertexId = 77 CrimeId = 7a7b3vertexId = 417 CrimeId = 3a001vertexId = 154 CrimeId = e4764vertexId = 20 CrimeId = 71200
```

MST Tour Weight(meters) = 513326.09539907286

After 2 Opt Weight = 637578.4578100945

Approx Ratio= 2 opt/MST=637578.4578100945/513326.09539907286

Approximation Ratio=1.24

MST Tour Weight(meters) = 513326.09539907286

After 3 Opt Weight = 636788.2026585272

Approx Ratio= 3 opt/MST= 636788.2026585272/513326.09539907286

Approximation Ratio=1.24

```

174 tspSimulated.getTour().stream().forEach(e -> {
175     Vertex v = new Vertex(e, vertexMap.get(e));
176     simulatedAnnealingVertexTour.add(v);
177 });
178 System.out.println(" Simulated Annealing Tsp Weight in meters " + ((tspSimulated.getLength()-4,567)) * 1000);
179 System.out.println(" Simulated Annealing Tsp Tour " + simulatedAnnealingVertexTour);
180 System.out.println("***** Simulated Annealing Tsp Tour *****");
181 System.out.println(printUtil(simulatedAnnealingVertexTour));
182
183
184
185 // System.out.println("*****");
186 // startTime = new Timestamp(System.currentTimeMillis());
187 // AntColonyOptimizationService antColonyService=new AntColonyOptimizationService(70, 100, graph, tsp.getTour());
188 // Ant antTour = antColonyService.runAntColony(graph);
189 // System.out.println("Ant Colony Opt Execution Time in Milliseconds: "
190 // + (new Timestamp(System.currentTimeMillis()).getTime() - startTime.getTime()));
191 // List<Vertex> antColonyVertexTour = new ArrayList<>();
192 // antTour.antTour().stream().forEach(e -> {

```

Console X

TspApplication [Java Application] /Applications/Eclipse.app/Contents/Eclipse/plugins/org.eclipse.jdt.openjdk.hotspot.jre.full.macosx.x86_64.17.0.4.v20220903-1038/jre/bin/java (Apr 18, 2023, 9:01:40 PM) [pid: 90376]

```

*****
30pt Execution Time in Milliseconds: 659078
3 Opt Tsp Weight in meters 636788.2026585272
*****
3 Opt Tsp Tour *****
vertexId = 0 CrimeId = 47823vertexId = 368 CrimeId = 64e7avertexId = 106 CrimeId = 74aa7vertexId = 305 CrimeId = b9086
vertexId = 7 CrimeId = cee38vertexId = 548 CrimeId = c51e6vertexId = 5 CrimeId = e93d1vertexId = 418 CrimeId = d10aa
vertexId = 62 CrimeId = 98795vertexId = 112 CrimeId = f37ccvertexId = 148 CrimeId = c828avertexId = 382 CrimeId = 7773f
vertexId = 414 CrimeId = d4615vertexId = 22 CrimeId = 480a2vertexId = 569 CrimeId = b46afvertexId = 287 CrimeId = 85357
vertexId = 311 CrimeId = 1c7e4vertexId = 70 CrimeId = 4dffevertexId = 541 CrimeId = 5439avertexId = 60 CrimeId = 19cb3
vertexId = 238 CrimeId = b97cdvertexId = 376 CrimeId = 2c19bvertexId = 426 CrimeId = b86c7vertexId = 241 CrimeId = f68b8
vertexId = 334 CrimeId = c5c16vertexId = 61 CrimeId = bb3a6vertexId = 427 CrimeId = aae6avertexId = 528 CrimeId = 81c6c
vertexId = 343 CrimeId = 0a7a9vertexId = 515 CrimeId = 8e185vertexId = 158 CrimeId = f98e7vertexId = 187 CrimeId = 0428d
vertexId = 445 CrimeId = ba063vertexId = 455 CrimeId = b7681vertexId = 553 CrimeId = b85d9vertexId = 446 CrimeId = 53135
vertexId = 480 CrimeId = 6c267vertexId = 577 CrimeId = 7283dvertexId = 54 CrimeId = faeaavertexId = 217 CrimeId = b71c9
vertexId = 324 CrimeId = 7048fvertexId = 186 CrimeId = af4cevertexId = 382 CrimeId = f1507vertexId = 9 CrimeId = e04c1
vertexId = 336 CrimeId = 3a074vertexId = 513 CrimeId = 42ba6vertexId = 321 CrimeId = d4299vertexId = 379 CrimeId = db067
vertexId = 122 CrimeId = 1fc21vertexId = 498 CrimeId = 9e912vertexId = 387 CrimeId = a661bvertexId = 109 CrimeId = f1a4e
vertexId = 72 CrimeId = ffb25vertexId = 537 CrimeId = 1980bvertexId = 451 CrimeId = 57aa7vertexId = 21 CrimeId = bff1e
vertexId = 288 CrimeId = 84393vertexId = 124 CrimeId = 647afvertexId = 237 CrimeId = afcc4vertexId = 511 CrimeId = e8c5b
vertexId = 458 CrimeId = 0e210vertexId = 238 CrimeId = 46e26vertexId = 190 CrimeId = 36acavertexId = 364 CrimeId = 29047
vertexId = 185 CrimeId = 3361bvertexId = 181 CrimeId = 96dd4vertexId = 114 CrimeId = bd42evertexId = 19 CrimeId = 88b5c
vertexId = 84 CrimeId = f4751vertexId = 514 CrimeId = 78909vertexId = 49 CrimeId = 4d4ddvertexId = 45 CrimeId = 6cb16
vertexId = 282 CrimeId = c80c3vertexId = 272 CrimeId = 1e779vertexId = 153 CrimeId = c726bvertexId = 167 CrimeId = 97aa5
vertexId = 449 CrimeId = b1b19vertexId = 115 CrimeId = 58622vertexId = 453 CrimeId = 62a5fvertexId = 326 CrimeId = 66ceb
vertexId = 286 CrimeId = f2fb7vertexId = 85 CrimeId = 7cd8bvertexId = 59 CrimeId = 74a15vertexId = 264 CrimeId = 1512a
vertexId = 417 CrimeId = ceab2vertexId = 434 CrimeId = d9536vertexId = 147 CrimeId = e71b5vertexId = 118 CrimeId = 36c1d
vertexId = 488 CrimeId = c9b4dvertexId = 58 CrimeId = c3fafvertexId = 249 CrimeId = cce32vertexId = 194 CrimeId = cfc63
vertexId = 29 CrimeId = ba74evertexId = 347 CrimeId = ba542vertexId = 395 CrimeId = 30704vertexId = 146 CrimeId = 2b083
vertexId = 102 CrimeId = af637vertexId = 386 CrimeId = 61e92vertexId = 68 CrimeId = 58ed1vertexId = 267 CrimeId = 662e9
vertexId = 362 CrimeId = fe7aevertexId = 454 CrimeId = 8d2d7vertexId = 450 CrimeId = 881c9vertexId = 295 CrimeId = d1ba6
vertexId = 208 CrimeId = 1408bvertexId = 75 CrimeId = d2c1dvertexId = 437 CrimeId = 9bdeevertexId = 263 CrimeId = 62998
vertexId = 315 CrimeId = 43aa0vertexId = 257 CrimeId = 0e956vertexId = 556 CrimeId = 3a496vertexId = 424 CrimeId = 76697
vertexId = 11 CrimeId = a08efvertexId = 409 CrimeId = f3ee1vertexId = 24 CrimeId = ca598vertexId = 561 CrimeId = 4d2c0
vertexId = 287 CrimeId = dded9vertexId = 490 CrimeId = ef0b1vertexId = 392 CrimeId = e1b9bvertexId = 89 CrimeId = 48c20
vertexId = 361 CrimeId = 6b397vertexId = 518 CrimeId = b82a2vertexId = 473 CrimeId = 09e90vertexId = 149 CrimeId = 18474
vertexId = 571 CrimeId = ffe9evertexId = 488 CrimeId = 26349vertexId = 546 CrimeId = 8b0f7vertexId = 329 CrimeId = c5f90

```

MST Tour Weight(meters) = 513326.09539907286

After Simulated Annealing Weight = 632221.2026585272

Approx Ratio= simulated/MST= 632221.2026585272

/513326.09539907286

Approximation Ratio=1.23

Package Explorer | JUnit | MinWeightService | SimulatedAnneal | TspApplication | TspApplicationT | UnionFind.java | AntColonyOptimi | Ant.java | README.md

INFO6205 [INFO6205 Spring2023]

PSA_Project [PSA_Project main]

edu.neu.coe.csye6205.tsp

edu.neu.coe.csye6205.tsp.controller

psa

edu.neu.coe.csye6205.tsp.model

Ant.java

Graph.java

GraphDto.java

MatTour.java

Point.java

Route.java

TspTour.java

Vertex.java

edu.neu.coe.csye6205.tsp.repository

edu.neu.coe.csye6205.tsp.service

AntColonyOptimizationService.java

CsvUtilService.java

ExcelFetchPointService.java

ExcelFetchRouteService.java

Linkernighan2Opt.java

RouteWeightService.java

SimulatedAnnealingService.java

ThreeOptService.java

TwoOptService.java

edu.neu.coe.csye6205.tsp.util

src/main/resources

src/test/java

JRE System Library [JavaSE-17]

Maven Dependencies

bin

Data

src

target

points.xlsx

points.xlsx

pom.xml

README.md

routes.xlsx

```
189 // System.out.println("Ant Colony Opt Execution Time in Millisecs:");
190 // + (new Timestamp(System.currentTimeMillis())).getTime() - startTime.getTime());
191 // List<Vertex> AntColonyVertexTour = new ArrayList<>();
192 // antTour.getTour().stream().forEach(e -> {
```

Console

TspApplication [Java Application] [Applications\Eclipse app\Contents\Eclipse\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.macosx.x86_64_17.0.4.v20220903-1038\re\bin\java (Apr 18, 2023, 9:01:40 PM) [pid: 90376]

***** Simulated Annealing Opt Execution Time in Millisecs: 665578 *****

Simulated Annealing Tsp Weight in meters 632221.2026585272

Simulated Annealing Tsp Tour [vertexId = 0 CrimeId = 47823, vertexId = 368 CrimeId = 64e7a, vertexId = 106 CrimeId = 74aa7, vertexId = 305 CrimeId = b9086, vertexId = 7 CrimeId = cee38, vertexId = 540 CrimeId = c51e6, vertexId = 5 CrimeId = e93d1, vertexId = 418 CrimeId = d1eaa, vertexId = 62 CrimeId = 987b5, vertexId = 112 CrimeId = f37cc, vertexId = 140 CrimeId = c828a, vertexId = 382 CrimeId = 7773f, vertexId = 414 CrimeId = d4615, vertexId = 22 CrimeId = 400ba, vertexId = 569 CrimeId = b46af, vertexId = 207 CrimeId = 8535f, vertexId = 311 CrimeId = 1c7e4, vertexId = 70 CrimeId = 4dfef, vertexId = 541 CrimeId = 5439a, vertexId = 60 CrimeId = 19c3b, vertexId = 238 CrimeId = b97cd, vertexId = 376 CrimeId = 2cf9b, vertexId = 426 CrimeId = b06c7, vertexId = 241 CrimeId = f68b8, vertexId = 334 CrimeId = c5c16, vertexId = 61 CrimeId = bb3ab, vertexId = 427 CrimeId = aae64, vertexId = 528 CrimeId = 01c6c, vertexId = 343 CrimeId = 0a7a9, vertexId = 515 CrimeId = 8e185, vertexId = 158 CrimeId = f98c7, vertexId = 187 CrimeId = 0428d, vertexId = 445 CrimeId = ba063, vertexId = 455 CrimeId = b7681, vertexId = 553 CrimeId = b85d9, vertexId = 446 CrimeId = 53135, vertexId = 400 CrimeId = 6c267, vertexId = 577 CrimeId = 7283d, vertexId = 54 CrimeId = faeaa, vertexId = 217 CrimeId = b71c9, vertexId = 324 CrimeId = 7840f, vertexId = 186 CrimeId = af4ce, vertexId = 382 CrimeId = f1587, vertexId = 9 CrimeId = ed4c1, vertexId = 336 CrimeId = 3a07a, vertexId = 513 CrimeId = 420ba, vertexId = 321 CrimeId = d4299, vertexId = 379 CrimeId = ddb67, vertexId = 122 CrimeId = 1fc21, vertexId = 498 CrimeId = 9e912, vertexId = 387 CrimeId = a661b, vertexId = 109 CrimeId = f1a4e, vertexId = 72 CrimeId = ffb25, vertexId = 537 CrimeId = 198db, vertexId = 451 CrimeId = 57aa7, vertexId = 21 CrimeId = bff1e, vertexId = 288 CrimeId = 84393, vertexId = 124 CrimeId = 647af, vertexId = 237 CrimeId = afcc4, vertexId = 511 CrimeId = e8c5b, vertexId = 458 CrimeId = 0021b, vertexId = 230 CrimeId = 46e20, vertexId = 190 CrimeId = 36aca, vertexId = 364 CrimeId = 29847, vertexId = 185 CrimeId = 3361b, vertexId = 181 CrimeId = 96dd4, vertexId = 114 CrimeId = bd42e, vertexId = 19 CrimeId = 88b5c, vertexId = 84 CrimeId = f4751, vertexId = 514 CrimeId = 789d9, vertexId = 49 CrimeId = 4dddd, vertexId = 45 CrimeId = 6cb16, vertexId = 282 CrimeId = c80c3, vertexId = 272 CrimeId = 1e779, vertexId = 153 CrimeId = c726b, vertexId = 167 CrimeId = 97aa5, vertexId = 449 CrimeId = b1bf9, vertexId = 115 CrimeId = 58d22, vertexId = 453 CrimeId = 62a5f, vertexId = 328 CrimeId = 66ceb, vertexId = 286 CrimeId = f2fb7, vertexId = 85 CrimeId = 7cd8b, vertexId = 59 CrimeId = 74a15, vertexId = 264 CrimeId = 1512a, vertexId = 417 CrimeId = ceab2, vertexId = 434 CrimeId = d9536, vertexId = 147 CrimeId = e71b5, vertexId = 118 CrimeId = 36c1d, vertexId = 408 CrimeId = c9b4d, vertexId = 58 CrimeId = c3faf, vertexId = 249 CrimeId = cce32, vertexId = 194 CrimeId = cfec3, vertexId = 29 CrimeId = ba74e, vertexId = 347 CrimeId = ba542, vertexId = 395 CrimeId = 30704, vertexId = 146 CrimeId = 2b603, vertexId = 182 CrimeId = af637, vertexId = 386 CrimeId = 61e92, vertexId = 68 CrimeId = 50ed1, vertexId = 267 CrimeId = 66e29, vertexId = 362 CrimeId = fe7ae, vertexId = 454 CrimeId = 8d2d7, vertexId = 450 CrimeId = 801c9, vertexId = 295 CrimeId = d1ba6, vertexId = 200 CrimeId = 14b0b, vertexId = 75 CrimeId = d2c1d, vertexId = 437 CrimeId = 9bdee, vertexId = 263 CrimeId = d9298, vertexId = 315 CrimeId = 43aa9, vertexId = 257 CrimeId = 0c85e, vertexId = 556 CrimeId = 3a496, vertexId = 424 CrimeId = 76697, vertexId = 11 CrimeId = a0de4, vertexId = 400 CrimeId = f3e1e, vertexId = 24 CrimeId = ca598, vertexId = 561 CrimeId = 442c0, vertexId = 287 CrimeId = dded9, vertexId = 490 CrimeId = ef0b1, vertexId = 392 CrimeId = e1b9b, vertexId = 89 CrimeId = d8c20, vertexId = 361 CrimeId = 6b397, vertexId = 510 CrimeId = b82a2, vertexId = 473 CrimeId = 09e90, vertexId = 149 CrimeId = 18474, vertexId = 571 CrimeId = ff9e9, vertexId = 488 CrimeId = 2d349, vertexId = 546 CrimeId = 8bdfb, vertexId = 329 CrimeId = c5f90, vertexId = 584 CrimeId = e701b, vertexId = 134 CrimeId = cdfcf, vertexId = 299 CrimeId = 2ccf2, vertexId = 356 CrimeId = 421bf, vertexId = 423 CrimeId = c15c0, vertexId = 15 CrimeId = cec4c, vertexId = 403 CrimeId = 595f8, vertexId = 293 CrimeId = 803cc, vertexId = 171 CrimeId = 3d168, vertexId = 318 CrimeId = cfcbe, vertexId = 176 CrimeId = fdcf4, vertexId = 142 CrimeId = f5b0f, vertexId = 578 CrimeId = 83178, vertexId = 566 CrimeId = 3eeel, vertexId = 406 CrimeId = bda8f, vertexId = 71 CrimeId = ba3ae, vertexId = 136 CrimeId = 53257, vertexId = 83 CrimeId = 5da87, vertexId = 285 CrimeId = a833d, vertexId = 88 CrimeId = 2d0ae, vertexId = 79 CrimeId = 34272, vertexId = 470 CrimeId = e8c89, vertexId = 385 CrimeId = 2a25c, vertexId = 351 CrimeId = d8b01, vertexId = 199 CrimeId = 9e51c, vertexId = 236 CrimeId = 4dcfa, vertexId = 487 CrimeId = a7b0f, vertexId = 65 CrimeId = e85ee, vertexId = 244 CrimeId = 44fb2, vertexId = 53 CrimeId = 36cae, vertexId = 548 CrimeId = c085f, vertexId = 497 CrimeId = fa4e7, vertexId = 43 CrimeId = e01c, vertexId = 9 CrimeId = 4bcd0, vertexId = 155 CrimeId = 42640, vertexId = 525 CrimeId = 6a705, vertexId = 36 CrimeId = c3298, vertexId = 280 CrimeId = ec3b1, vertexId = 405 CrimeId = f9816, vertexId = 170 CrimeId = 6af09, vertexId = 52 CrimeId = 5aad7, vertexId = 580 CrimeId = a30f9, vertexId = 452 CrimeId = 29645, vertexId = 377 CrimeId = e0ca5, vertexId = 544 CrimeId = 4fc27, vertexId = 77 CrimeId = 38c6e, vertexId = 346 CrimeId = 849d1, vertexId = 310 CrimeId = cc9f3, vertexId = 558 CrimeId = 63b73, vertexId = 502 CrimeId = 110be, vertexId = 340 CrimeId = 937c0, vertexId = 429 CrimeId = b37e5, vertexId = 223 CrimeId = d6ebf, vertexId = 478 CrimeId = 5bfe3, vertexId = 44 CrimeId = 31838, vertexId = 459 CrimeId = 93956

MST Tour Weight(meters) = 513326.09539907286

After Ant Colony opt Weight = 632221.2026585272

Approx Ratio= Ant/MST= 632115.7917132177/513326.09539907286

Approximation Ratio=1.232

The screenshot displays the Eclipse IDE interface. On the left, the Package Explorer shows the project structure for 'edu.neu.coe.csye6205.tsp'. The main editor window shows the 'TspApplication.java' file, which contains the main logic of the Ant Colony Optimization algorithm. The code includes imports for 'java.util.*', 'java.io.*', and 'edu.neu.coe.csye6205.tsp.*'. It defines a 'TspApplication' class with a 'main' method that initializes the algorithm and prints the results. The console window at the bottom shows the output of the program, including the execution time in milliseconds and the final weight of the tour. The output is as follows:

```
*****
Ant Colony Opt Execution Time in Milliseconds: 1093569
Ant Colony opt Tsp Weight in meters 632115.7917132177
Ant Colony opt Tsp Tour (vertexId = 0 CrimeId = 47823, vertexId = 141 CrimeId = ea746, vertexId = 313 CrimeId = 916a4, vertexId = 554 CrimeId = f384c, vertexId = 175 CrimeId = 6b810, vertexId = 338 CrimeId = 7a25b, vertexId = 527 CrimeId = 9182f, vertexId = 73 CrimeId = 677a4, vertexId = 539 CrimeId = f055c, vertexId = 278 CrimeId = 59add, vertexId = 117 CrimeId = f8ed5, vertexId = 389 CrimeId = 22d14, vertexId = 269 CrimeId = 5b793, vertexId = 31 CrimeId = 9db58, vertexId = 55 CrimeId = 02057, vertexId = 227 CrimeId = 3ce3f, vertexId = 465 CrimeId = bb937, vertexId = 531 CrimeId = 15ede, vertexId = 214 CrimeId = 71eab, vertexId = 472 CrimeId = 0951a, vertexId = 345 CrimeId = 29943, vertexId = 172 CrimeId = ae1e1, vertexId = 521 CrimeId = 0ad98, vertexId = 156 CrimeId = b5b64, vertexId = 288 CrimeId = 84939, vertexId = 124 CrimeId = 647af, vertexId = 237 CrimeId = afcc4, vertexId = 458 CrimeId = 0d210, vertexId = 511 CrimeId = e8c5b, vertexId = 230 CrimeId = 46e26, vertexId = 190 CrimeId = 36acav, vertexId = 550 CrimeId = eac9b, vertexId = 47 CrimeId = adcl1, vertexId = 160 CrimeId = da011, vertexId = 360 CrimeId = 8fb58, vertexId = 139 CrimeId = 788a4, vertexId = 232 CrimeId = 6ed74, vertexId = 466 CrimeId = a75a5, vertexId = 182 CrimeId = c3355, vertexId = 183 CrimeId = e9a50, vertexId = 584 CrimeId = 90a8e, vertexId = 290 CrimeId = 8c81e, vertexId = 174 CrimeId = 1eb20, vertexId = 161 CrimeId = 3f416, vertexId = 250 CrimeId = cbe84, vertexId = 191 CrimeId = 10777, vertexId = 543 CrimeId = 09250, vertexId = 399 CrimeId = 681a5, vertexId = 447 CrimeId = 874e3, vertexId = 261 CrimeId = ea5e7, vertexId = 143 CrimeId = c8783, vertexId = 342 CrimeId = e8d70, vertexId = 432 CrimeId = b1b9b, vertexId = 1 CrimeId = 3d882, vertexId = 583 CrimeId = cc12f, vertexId = 235 CrimeId = 63a49, vertexId = 46 CrimeId = 93619, vertexId = 131 CrimeId = f1b8b, vertexId = 98 CrimeId = a0647, vertexId = 485 CrimeId = 7fa41, vertexId = 289 CrimeId = 8e3a4, vertexId = 103 CrimeId = 49f4c, vertexId = 545 CrimeId = 7fc2d, vertexId = 159 CrimeId = 5db1a, vertexId = 97 CrimeId = 7fe42, vertexId = 519 CrimeId = 10c55, vertexId = 34 CrimeId = 573ab, vertexId = 373 CrimeId = 12ddb, vertexId = 532 CrimeId = 73b8a, vertexId = 533 CrimeId = 2cc42, vertexId = 407 CrimeId = 29946, vertexId = 419 CrimeId = 818b6, vertexId = 184 CrimeId = 50735, vertexId = 247 CrimeId = f653c, vertexId = 90 CrimeId = 7dfe2, vertexId = 460 CrimeId = 0c987, vertexId = 168 CrimeId = 2bf09, vertexId = 82 CrimeId = 61a84, vertexId = 559 CrimeId = 98125, vertexId = 234 CrimeId = 486aa, vertexId = 41 CrimeId = 69f18, vertexId = 474 CrimeId = 394fd, vertexId = 296 CrimeId = 99d29, vertexId = 441 CrimeId = 7d2d7, vertexId = 291 CrimeId = d24e7, vertexId = 206 CrimeId = 3f4fb, vertexId = 285 CrimeId = 116db, vertexId = 468 CrimeId = 67e12, vertexId = 93 CrimeId = 3f884, vertexId = 433 CrimeId = 5f5bc, vertexId = 398 CrimeId = 95ba8, vertexId = 216 CrimeId = 1a5df, vertexId = 388 CrimeId = c269e, vertexId = 37 CrimeId = d182d, vertexId = 378 CrimeId = 2c8b0, vertexId = 469 CrimeId = 4fa0b, vertexId = 284 CrimeId = 03c99, vertexId = 69 CrimeId = 0868c, vertexId = 471 CrimeId = 4a714, vertexId = 312 CrimeId = c0b73, vertexId = 50 CrimeId = 436ec, vertexId = 253 CrimeId = d972b, vertexId = 564 CrimeId = 0637b, vertexId = 276 CrimeId = 1f724, vertexId = 319 CrimeId = 33982, vertexId = 415 CrimeId = 0db30, vertexId = 166 CrimeId = 5f1eb, vertexId = 457 CrimeId = e1cd5, vertexId = 209 CrimeId = 0fe9a, vertexId = 384 CrimeId = a1279, vertexId = 242 CrimeId = 4ee9b, vertexId = 367 CrimeId = c0a40, vertexId = 480 CrimeId = 55f0d, vertexId = 163 CrimeId = f00de, vertexId = 304 CrimeId = bf103, vertexId = 197 CrimeId = bad43, vertexId = 585 CrimeId = 1d259, vertexId = 279 CrimeId = fe864, vertexId = 3 CrimeId = fdde0, vertexId = 127 CrimeId = 4709f, vertexId = 332 CrimeId = f4456, vertexId = 489 CrimeId = 54e6b, vertexId = 328 CrimeId = d0993, vertexId = 370 CrimeId = b8962, vertexId = 320 CrimeId = 4af3b, vertexId = 259 CrimeId = 0f199, vertexId = 14 CrimeId = 97f15, vertexId = 277 CrimeId = 7b357, vertexId = 316 CrimeId = 4cf86, vertexId = 333 CrimeId = bc3a7, vertexId = 412 CrimeId = f5094, vertexId = 16 CrimeId = 264aa, vertexId = 198 CrimeId = 09a87, vertexId = 20 CrimeId = 31a00, vertexId = 154 CrimeId = 5f5b4, vertexId = 27 CrimeId = 7ac7b, vertexId = 442 CrimeId = 3f894, vertexId = 148 CrimeId = e924d, vertexId = 337 CrimeId = aa7ba, vertexId = 357 CrimeId = 4b986, vertexId = 297 CrimeId = a30a8, vertexId = 420 CrimeId = ccae0, vertexId = 563 CrimeId = 15c89, vertexId = 225 CrimeId = 4d065, vertexId = 66 CrimeId = cea0e, vertexId = 110 CrimeId = 7a813, vertexId = 369 CrimeId = 1a8db, vertexId = 10 CrimeId = f5d8a, vertexId = 555 CrimeId = 395c9, vertexId = 180 CrimeId = 22e22, vertexId = 170 CrimeId = 22e48, vertexId = 76 CrimeId = c232d, vertexId = 113 CrimeId = dc716, vertexId = 464 CrimeId = 9418e, vertexId = 67 CrimeId = c2856, vertexId = 239 CrimeId = 5042a, vertexId = 4 CrimeId = 9e1a6, vertexId = 51 CrimeId = 98adb, vertexId = 177 CrimeId = 31fc8, vertexId = 430 CrimeId = f060b, vertexId = 524 CrimeId = 0c7b9, vertexId = 339 CrimeId = cad8c, vertexId = 496 CrimeId = 9be80, vertexId = 572 CrimeId = 97951, vertexId = 335 CrimeId = 6e85a, vertexId = 484 CrimeId = 144e9, vertexId = 222 CrimeId = 9e57f, vertexId = 530 CrimeId = 0b647
```

Unit Tests:

Below are the Unit Tests results along with Code ScreenShots :

The screenshot displays the Eclipse IDE interface with the JUnit test results and the corresponding source code.

JUnit Results:

- Package Explorer: JUnit
- Finished after 2.302 seconds
- Runs: 7/7 Errors: 0 Failures: 0
- Test Results:
 - TspApplicationTests (Runner: JUnit 5) (0.259 s)
 - Opt2Test() (0.000 s)
 - Opt3Test() (0.000 s)
 - findMinimumWeightMatchingTest() (0.217 s)
 - minimumSpanningTreeTest() (0.008 s)
 - getTstPathAndWeightTest() (0.008 s)
 - findEulerianTourTest() (0.003 s)
 - OddDegreeVertexTest() (0.002 s)

Source Code (TspApplicationTests.java):

```
43
44 @Test
45 public void minimumSpanningTreeTest1() {
46     GraphDto dto = InitializeGraph();
47     MstTour mstTour = minWeightService.minimumSpanningTree(dto.getEdges());
48     assertTrue(mstTour.getLength() == 4);
49 }
50
51 @Test
52 public void OddDegreeVertexTest() {
53     List<Integer> res = new ArrayList<>(Arrays.asList(0, 2, 3, 4));
54     List<Edge> edges = new ArrayList<>();
55     edges.add(new Edge(1, 0, 1.0));
56     edges.add(new Edge(2, 1, 1.0));
57     edges.add(new Edge(3, 0, 1.0));
58     edges.add(new Edge(4, 0, 1.0));
59     List<Integer> oddVertex = minWeightService.oddDegreeVertices(edges);
60     assertTrue(res.equals(oddVertex));
61 }
62
63 @Test
64 public void findMinimumWeightMatchingTest() {
65     List<Integer> res = new ArrayList<>(Arrays.asList(0, 2, 3, 4));
66     List<Edge> edges = new ArrayList<>();
67     edges.add(new Edge(1, 0, 1.0));
68     edges.add(new Edge(2, 1, 1.0));
69     edges.add(new Edge(3, 0, 1.0));
70     edges.add(new Edge(4, 0, 1.0));
71     GraphDto dto = InitializeGraph();
72     List<Edge> minWeightEdges = minWeightService.findMinimumWeightMatching(res, dto.getGraph());
73     List<Edge> combinedGraphEdges = minWeightService.combineGraphs(edges, minWeightEdges);
74     assertTrue(minWeightService.oddDegreeVertices(combinedGraphEdges).size() == 0);
75 }
76
77 @Test
78 public void findEulerianTourTest() {
79     List<Edge> combinedGraphEdges = new ArrayList<>();
80     combinedGraphEdges.add(new Edge(2, 1, 1.0));
81     combinedGraphEdges.add(new Edge(1, 0, 1.0));
82     combinedGraphEdges.add(new Edge(0, 3, 1.0));
83     combinedGraphEdges.add(new Edge(3, 0, 1.0));
84     combinedGraphEdges.add(new Edge(2, 4, 1.0));
85     combinedGraphEdges.add(new Edge(4, 0, 1.0));
86     Graph mstAndMinWeightCombinedGraph = new Graph(5);
87     combinedGraphEdges.forEach(e -> {
88         mstAndMinWeightCombinedGraph.addVertex(e.from, e.to, e.weight);
89     });
90     List<Integer> eulerianTour = minWeightService.findEulerianTour(mstAndMinWeightCombinedGraph, 0);
91     assertTrue(new HashSet<>(eulerianTour).size() == 5);
92 }
93
94
```

Console:

```
<terminated> TspApplicationTests [JUnit] /Applications/Eclipse.app/Contents/Eclipse/plugins/org.eclipse.justi.openjdk.hotspot.jre.full.macosx.x86_64.17.0.4.v20220903-1038/re/bin/java (Apr 18, 2023, 5:44:15)
17:44:16.789 [main] DEBUG org.springframework.test.context.support.AbstractContextLoader -- Could not detect default resource locations for test class [TspApplicationTests]: no resource found for [TspApplicationTests.class], since no resource found for [TspApplicationTests.class]
17:44:16.831 [main] DEBUG org.springframework.boot.test.context.SpringBootTestContextBootstrapper -- Using ContextCustomizers for test class [TspApplicationTests]
17:44:16.936 [main] DEBUG org.springframework.test.context.util.TestContextSpringFactoriesUtils -- Skipping candidate TestExecutionListener [org.springframework.test.context.support.AnnotationMethodTestExecutionListener] because it is not a Spring Framework class
```

```

93 }
94
95 @Test
96 public void getTspPathAndWeightTest() {
97     GraphDto dto = InitializeGraph();
98     List<Edge> combinedGraphEdges = new ArrayList<>();
99     combinedGraphEdges.add(new Edge(2, 1, 1.0));
100     combinedGraphEdges.add(new Edge(1, 0, 1.0));
101     combinedGraphEdges.add(new Edge(0, 3, 1.0));
102     combinedGraphEdges.add(new Edge(3, 0, 1.0));
103     combinedGraphEdges.add(new Edge(2, 4, 1.0));
104     combinedGraphEdges.add(new Edge(4, 0, 1.0));
105     Graph mstAndMinWeightCombinedGraph = new Graph(5);
106     combinedGraphEdges.forEach(e -> {
107         mstAndMinWeightCombinedGraph.addVertex(e.from, e.to, e.weight);
108     });
109     List<Integer> eulerianTour = minWeightService.findEulerianTour(mstAndMinWeightCombinedGraph, 0);
110     TspTour tspTour = minWeightService.getTspPath(0, dto.getGraph(), eulerianTour);
111     System.out.println(tspTour.getTour());
112     assertTrue(tspTour.getLength() == 6 && new HashSet<>(tspTour.getTour().size() == 5
113         && tspTour.getTour().get(0) == tspTour.getTour().get(tspTour.getTour().size() - 1));
114 }
115
116 @Test
117 public void Opt2Test() {
118     GraphDto dto = InitializeGraph();
119     List<Integer> tspTour = new ArrayList<>(Arrays.asList(0, 3, 1, 2, 4, 0));
120     TspTour tsp = new TspTour();
121     tsp.setTour(tspTour);
122     tsp.setLength(6);
123     TspTour tsp2optTour = opt2Service.twoOptTour(tsp, dto.getGraph());
124     assertTrue(tsp2optTour.getLength() <= 6 && new HashSet<>(tsp2optTour.getTour().size() == 5
125         && tsp2optTour.getTour().get(0) == tsp2optTour.getTour().get(tsp2optTour.getTour().size() - 1));
126 }
127
128 @Test
129 public void Opt3Test() {
130     GraphDto dto = InitializeGraph();
131     List<Integer> tspTour = new ArrayList<>(Arrays.asList(0, 3, 1, 2, 4, 0));
132     TspTour tsp = new TspTour();
133     tsp.setTour(tspTour);
134     tsp.setLength(6);
135     TspTour tsp3optTour = opt3Service.threeOptTour(tsp, dto.getGraph());
136     assertTrue(tsp3optTour.getLength() <= 6 && new HashSet<>(tsp3optTour.getTour().size() == 5
137         && tsp3optTour.getTour().get(0) == tsp3optTour.getTour().get(tsp3optTour.getTour().size() - 1));
138 }
139
140 public GraphDto InitializeGraph() {
141     Graph graph = new Graph(5);
142     graph.addVertex(0, 1, 1);
143     graph.addVertex(0, 2, 2);
144 }

```

Conclusion:

In conclusion, our project utilized various algorithms and techniques to solve the Traveling Salesman Problem (TSP) for a given dataset of vertices. We first processed the dataset and computed the distances between each vertex. Next, we implemented Kruskal's Union Find Method to identify the minimum spanning tree (MST) of the graph. We then extracted the list of odd vertices from the MST Tour and identified the perfect matching for them using the minimum weight matching method. We combined the Perfect Matching and MST to generate an Eulerian tour of the graph, which we used to create a TSP tour that visited each vertex and returned to the starting vertex. The resulting TSP tour was optimized using the 2-opt and 3-opt techniques, as well as strategic optimizations like simulated annealing and ant colony optimization.

Our project achieved a minimum spanning tree value of 513326 and a TSP tour distance of 698925. After applying the 2-opt and 3-opt techniques, the TSP tour distance was further reduced to 637578 and 636788, respectively. The strategic optimization techniques of simulated annealing and ant colony optimization also contributed to reducing the TSP tour distance, resulting in values of 632221 and 632115, respectively.

Overall, our project demonstrated the effectiveness of various algorithms and techniques in solving the Traveling Salesman Problem, and highlighted the importance of optimization in achieving a more efficient and accurate solution.

References:

- <https://www.youtube.com/watch?v=GiDsjiBOVoA&t=396s>

- https://en.wikipedia.org/wiki/Christofides_algorithm
- https://en.wikipedia.org/wiki/Travelling_salesman_problem
- https://en.wikipedia.org/wiki/Simulated_annealing
- <https://slowandsteadybrain.medium.com/traveling-salesman-problem-ce78187cf1f3>