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**

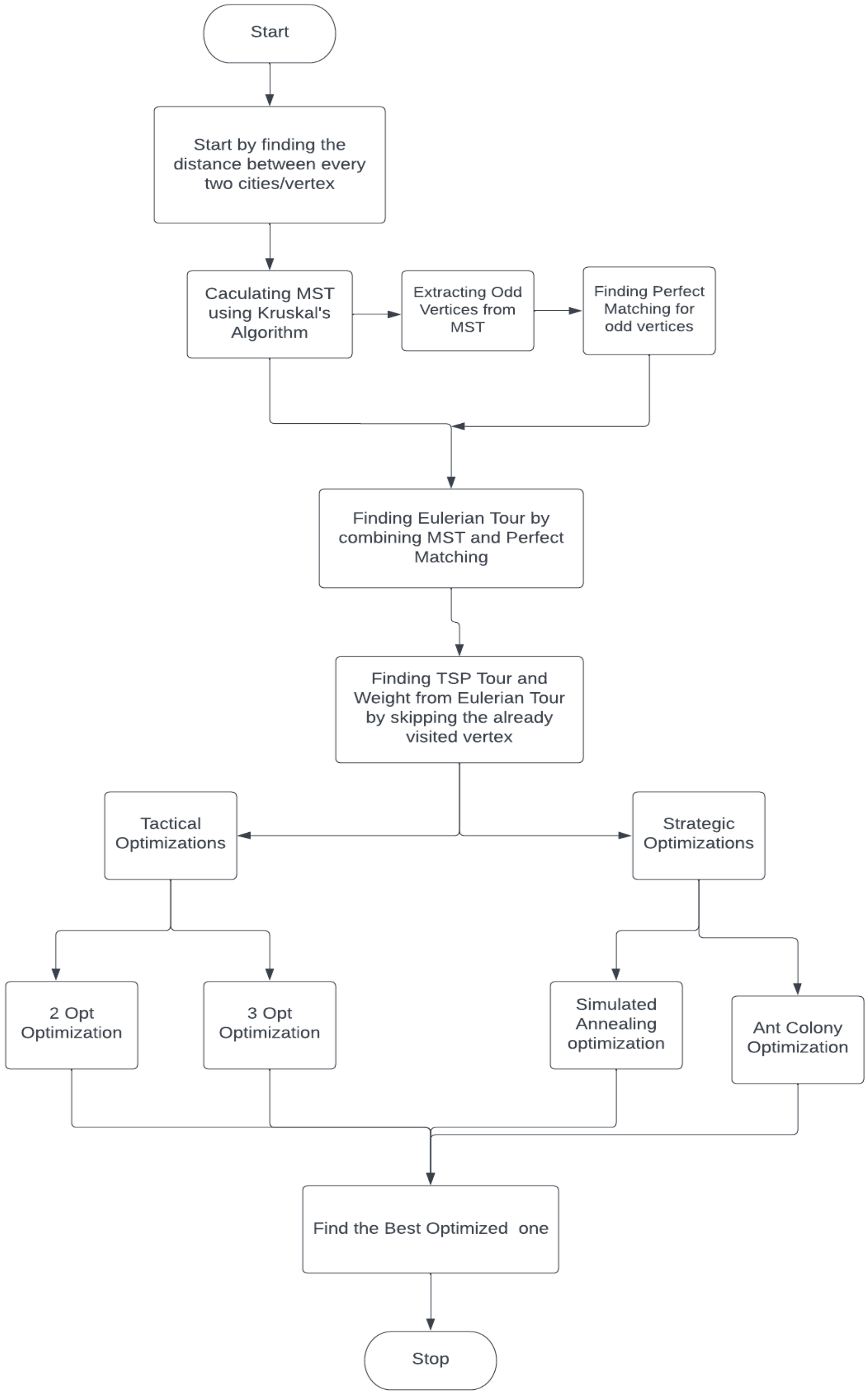
   Used In: MinWeightService

Invariants:

- Temperature
- Rate of Cooling

- Iterations

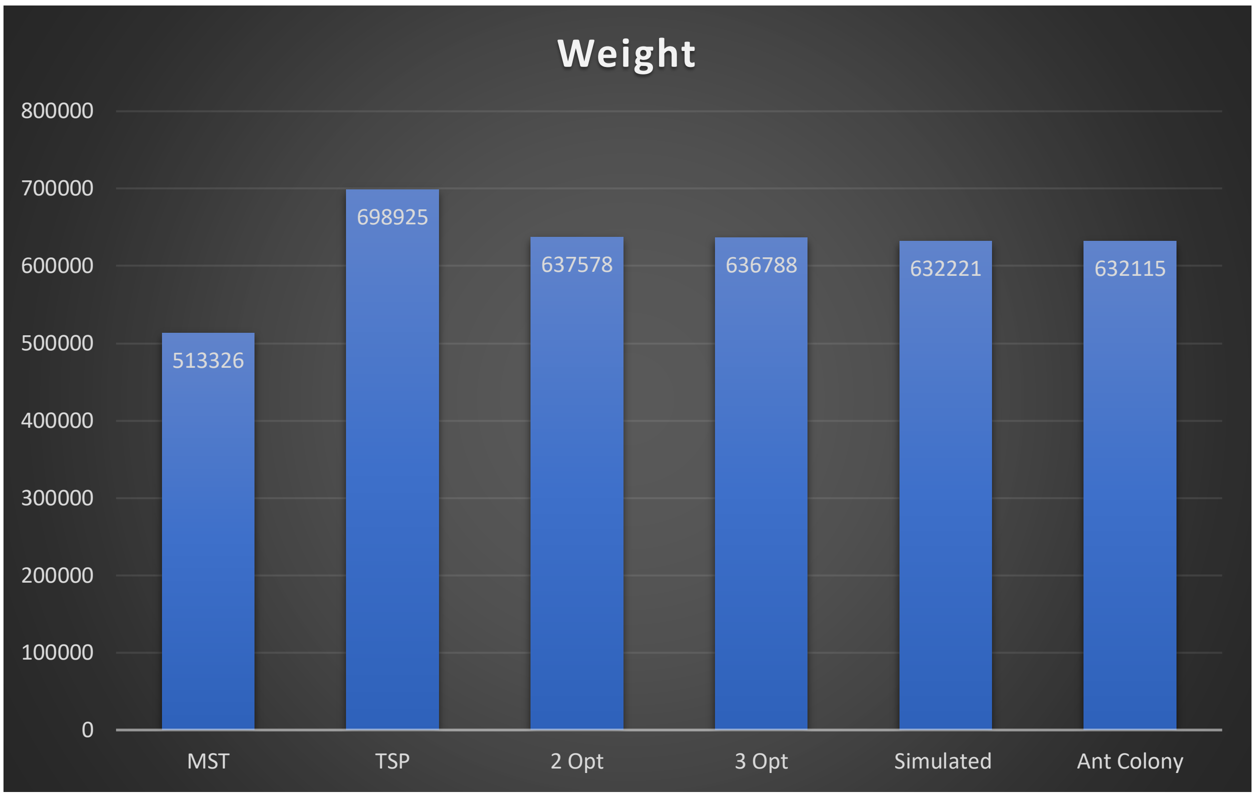
Flowchart:


Start
Start by finding the distance between every two cities/vertex
Caculating MST using Kruskal's Algorithm
Extracting Odd Vertices from MST
Finding Perfect Matching for odd vertices
Finding Eulerian Tour by combining MST and Perfect Matching
Finding TSP Tour and Weight from Eulerian Tour by skipping the already visited vertex
Tactical Optimizations
Strategic Optimizations
2 Opt Optimization
3 Opt Optimization
Simulated Annealing optimization
Ant Colony Optimization
Find the Best Optimized one
Stop

Observations and Graphical Analysis:

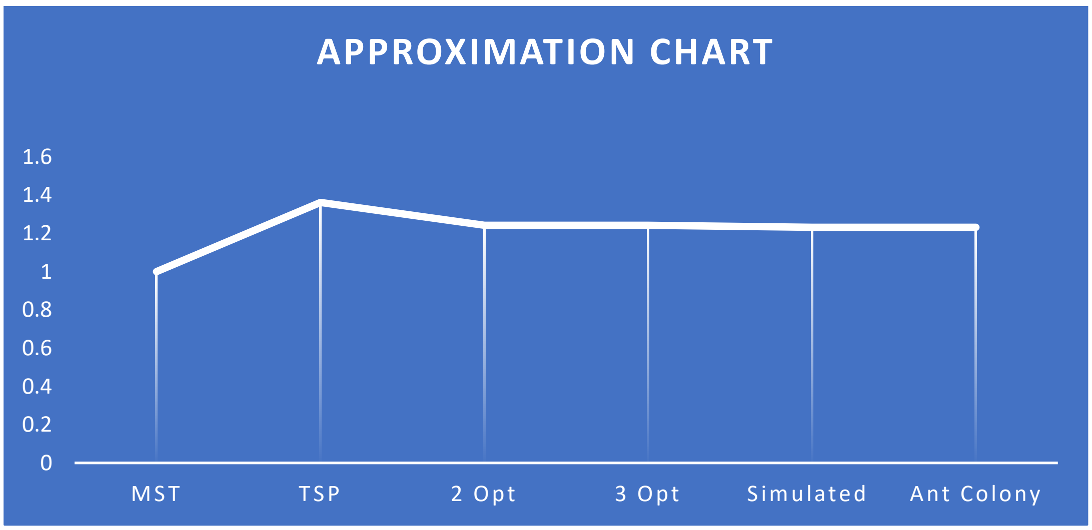
Graph for Distances of MST,TSP and Optimizations:

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



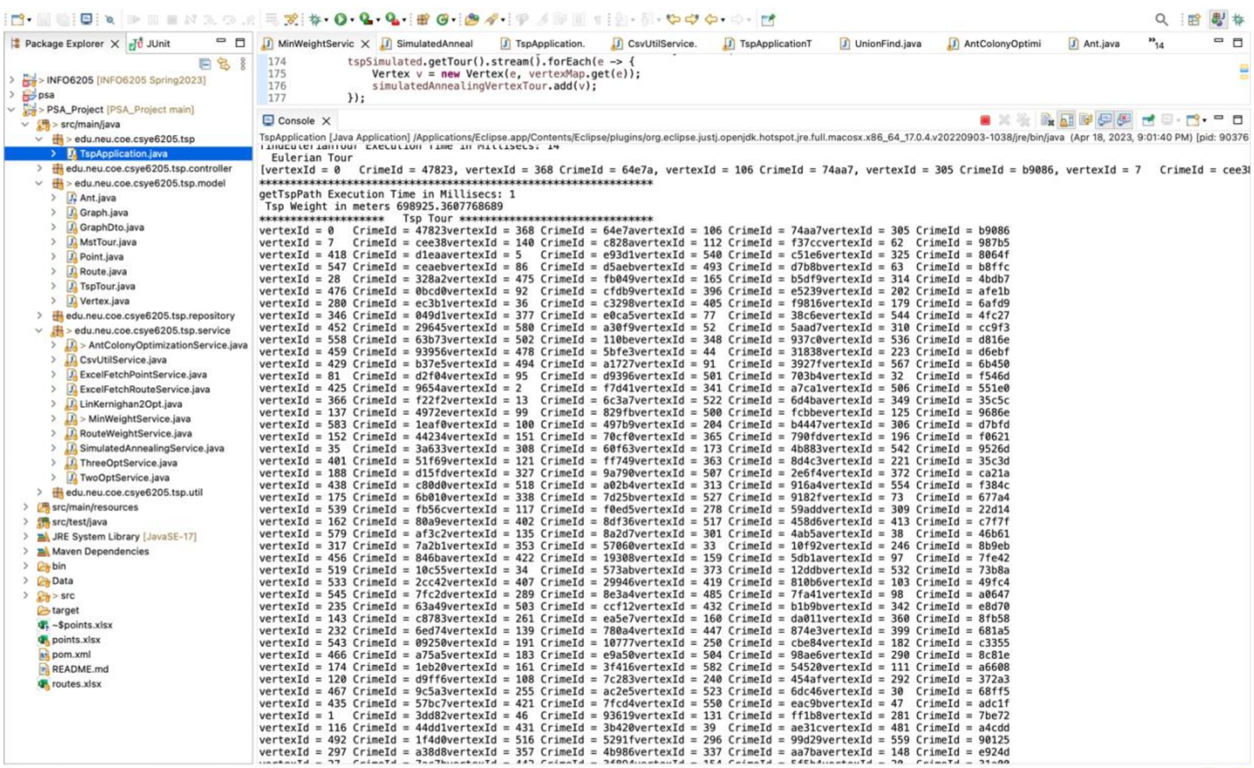
Results and Mathematical Analysis:

MST Tour Weight(meters) = 513326.09539907286

TSP Tour Weight(Meters) = 698925.3607768689

Approx Ratio= TSP/MST=698925.3607768689/513326.09539907286

Approximation Ratio=1.36

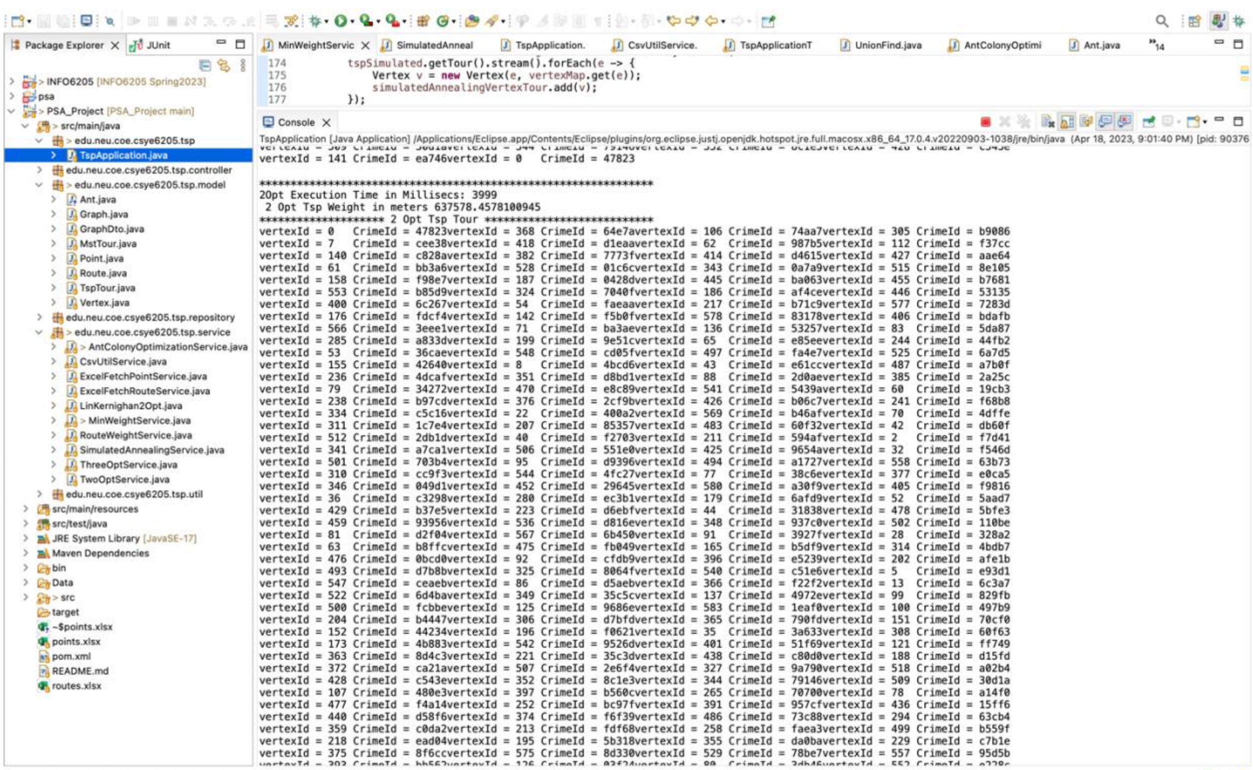


MST Tour Weight(meters) = 513326.09539907286

After 2 Opt Weight = 637578.4578100945

Approx Ratio= 2 opt/MST=637578.4578100945/513326.09539907286

Approximation Ratio=1.24

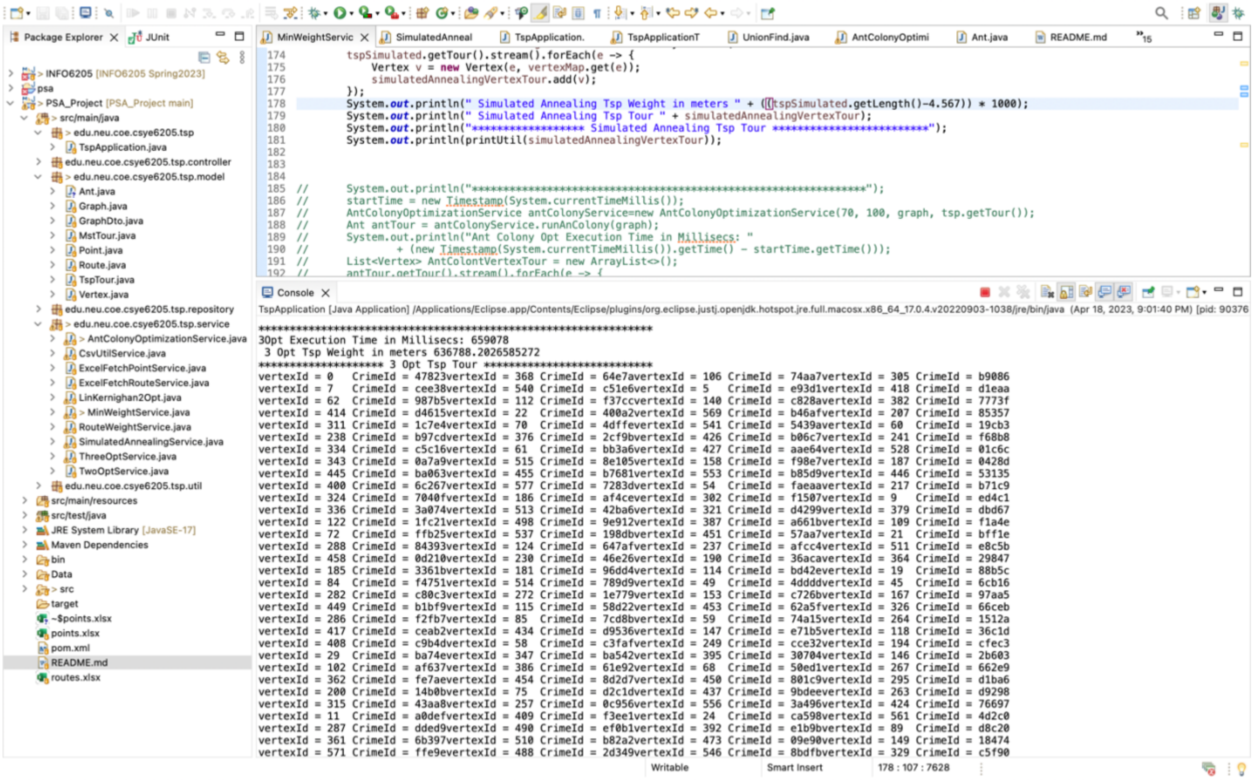


MST Tour Weight(meters) = 513326.09539907286

After 3 Opt Weight = 636788.2026585272

Approx Ratio= 3 opt/MST= 636788.2026585272/513326.09539907286

Approximation Ratio=1.24

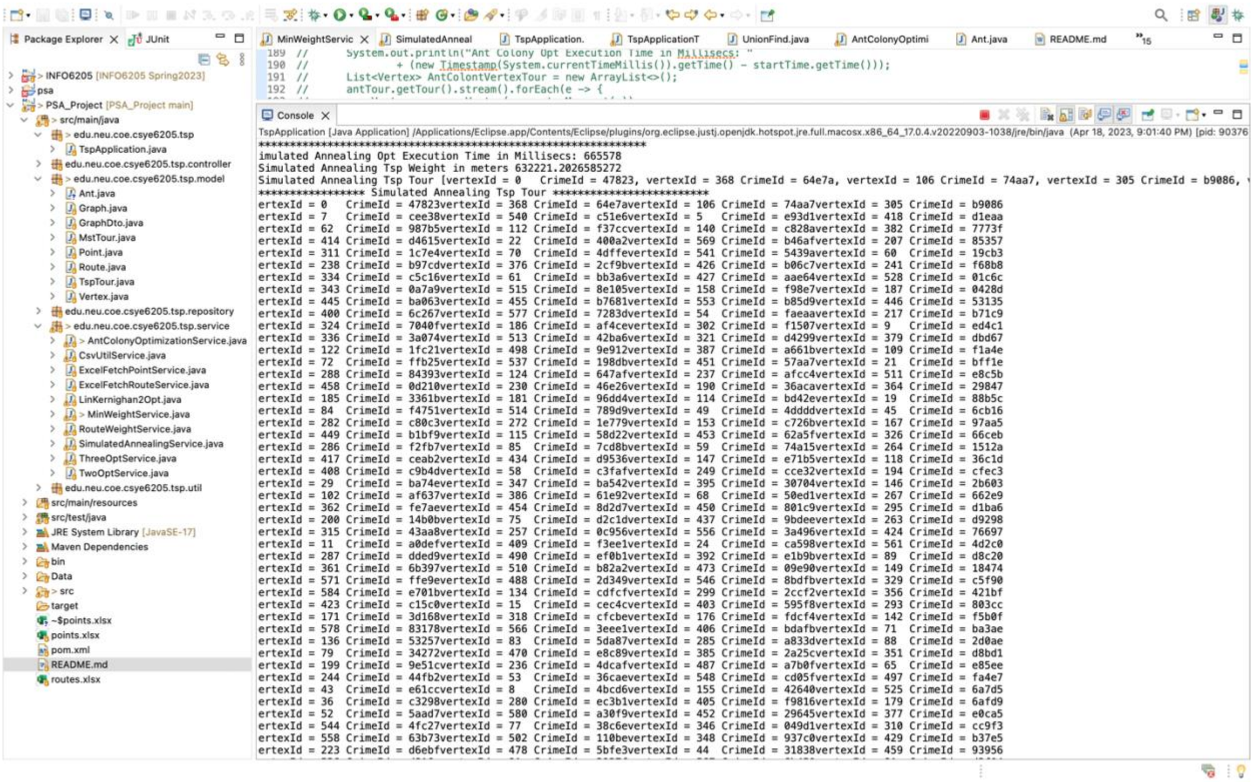


MST Tour Weight(meters) = 513326.09539907286

After Simulated Annealing Weight = 632221.2026585272

Approx Ratio= simulated/MST= 632221.2026585272 /513326.09539907286

Approximation Ratio=1.23

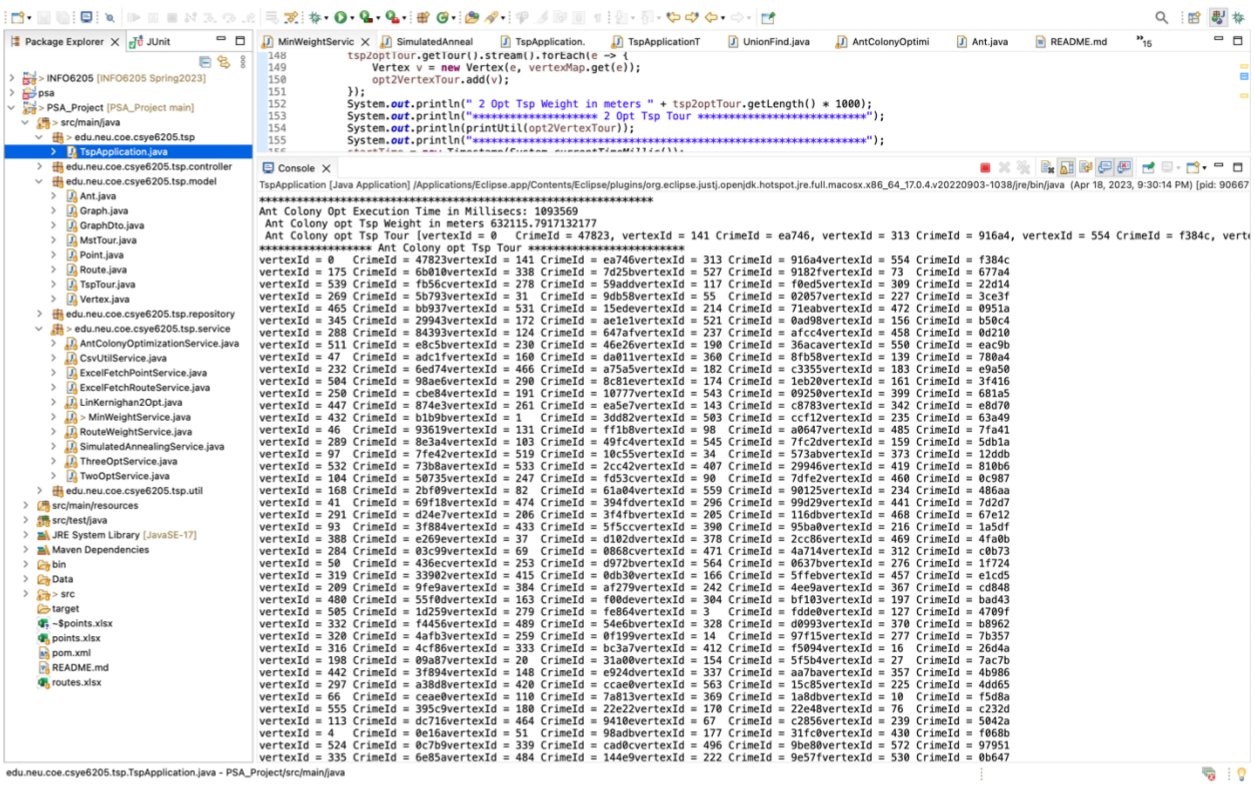

Package Explorer
JUnit
INFO6205 [INFO6205 Spring2023]
psa
PSA_Project [PSA_Project main]
src/main/java
edu.neu.coe.csye6205.tsp
TspApplication.java
edu.neu.coe.csye6205.tsp.controller
edu.neu.coe.csye6205.tsp.model
Ant.java
Graph.java
GraphDto.java
MstTour.java
Point.java
Route.java
TspTour.java
Vertex.java
edu.neu.coe.csye6205.tsp.repository
edu.neu.coe.csye6205.tsp.service
AntColonyOptimizationService.java
CsvUtilService.java
ExcelFetchPointService.java
ExcelFetchRouteService.java
LinKernighan2Opt.java
MinWeightService.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
MinWeightServic
SimulatedAnneal
TspApplication.
TspApplicationT
UnionFind.java
AntColonyOptimi
Ant.java
README.md
189 //   System.out.println("Ant Colony Opt Execution Time in Millisecs: "
190 //   + (new Timestamp(System.currentTimeMillis()).getTime() - startTime.getTime()));
191 //   List<Vertex> AntColontVertexTour = new ArrayList<>();
192 //   antTour.getTour().stream().forEach(e -> {
Console
TspApplication [Java Application] /Applications/Eclipse.app/Contents/Eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_17.0.4.v20220903-1038/jre/bin/java (Apr 18, 2023, 9:01:40 PM) [pid: 90376
****
imulated 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,
***************** Simulated Annealing Tsp Tour *************************
ertexId = 0   CrimeId = 47823vertexId = 368 CrimeId = 64e7avertexId = 106 CrimeId = 74aa7vertexId = 305 CrimeId = b9086
ertexId = 7   CrimeId = cee38vertexId = 540 CrimeId = c51e6vertexId = 5   CrimeId = e93d1vertexId = 418 CrimeId = d1eaa
ertexId = 62 CrimeId = 987b5vertexId = 112 CrimeId = f37ccvertexId = 140 CrimeId = c828avertexId = 382 CrimeId = 7773f
ertexId = 414 CrimeId = d4615vertexId = 22 CrimeId = 400a2vertexId = 569 CrimeId = b46afvertexId = 207 CrimeId = 85357
ertexId = 311 CrimeId = 1c7e4vertexId = 70 CrimeId = 4dffevertexId = 541 CrimeId = 5439avertexId = 60 CrimeId = 19cb3
ertexId = 238 CrimeId = b97cdvertexId = 376 CrimeId = 2cf9bvertexId = 426 CrimeId = b06c7vertexId = 241 CrimeId = f68b8
ertexId = 334 CrimeId = c5c16vertexId = 61 CrimeId = bb3a6vertexId = 427 CrimeId = aae64vertexId = 528 CrimeId = 01c6c
ertexId = 343 CrimeId = 0a7a9vertexId = 515 CrimeId = 8e105vertexId = 158 CrimeId = f98e7vertexId = 187 CrimeId = 0428d
ertexId = 445 CrimeId = ba063vertexId = 455 CrimeId = b7681vertexId = 553 CrimeId = b85d9vertexId = 446 CrimeId = 53135
ertexId = 400 CrimeId = 6c267vertexId = 577 CrimeId = 7283dvertexId = 54 CrimeId = faeaavertexId = 217 CrimeId = b71c9
ertexId = 324 CrimeId = 7040fvertexId = 186 CrimeId = af4cevertexId = 302 CrimeId = f1507vertexId = 9   CrimeId = ed4c1
ertexId = 336 CrimeId = 3a074vertexId = 513 CrimeId = 42ba6vertexId = 321 CrimeId = d4299vertexId = 379 CrimeId = dbd67
ertexId = 122 CrimeId = 1fc21vertexId = 498 CrimeId = 9e912vertexId = 387 CrimeId = a661bvertexId = 109 CrimeId = f1a4e
ertexId = 72 CrimeId = ffb25vertexId = 537 CrimeId = 198dbvertexId = 451 CrimeId = 57aa7vertexId = 21 CrimeId = bff1e
ertexId = 288 CrimeId = 84393vertexId = 124 CrimeId = 647afvertexId = 237 CrimeId = afcc4vertexId = 511 CrimeId = e8c5b
ertexId = 458 CrimeId = 0d210vertexId = 230 CrimeId = 46e26vertexId = 190 CrimeId = 36acavertexId = 364 CrimeId = 29847
ertexId = 185 CrimeId = 3361bvertexId = 181 CrimeId = 96dd4vertexId = 114 CrimeId = bd42evertexId = 19 CrimeId = 88b5c
ertexId = 84 CrimeId = f4751vertexId = 514 CrimeId = 789d9vertexId = 49 CrimeId = 4ddddvertexId = 45 CrimeId = 6cb16
ertexId = 282 CrimeId = c80c3vertexId = 272 CrimeId = 1e779vertexId = 153 CrimeId = c726bvertexId = 167 CrimeId = 97aa5
ertexId = 449 CrimeId = b1bf9vertexId = 115 CrimeId = 58d22vertexId = 453 CrimeId = 62a5fvertexId = 326 CrimeId = 66ceb
ertexId = 286 CrimeId = f2fb7vertexId = 85 CrimeId = 7cd8bvertexId = 59 CrimeId = 74a15vertexId = 264 CrimeId = 1512a
ertexId = 417 CrimeId = ceab2vertexId = 434 CrimeId = d9536vertexId = 147 CrimeId = e71b5vertexId = 118 CrimeId = 36c1d
ertexId = 408 CrimeId = c9b4dvertexId = 58 CrimeId = c3fafvertexId = 249 CrimeId = cce32vertexId = 194 CrimeId = cfec3
ertexId = 29 CrimeId = ba74evertexId = 347 CrimeId = ba542vertexId = 395 CrimeId = 30704vertexId = 146 CrimeId = 2b603
ertexId = 102 CrimeId = af637vertexId = 386 CrimeId = 61e92vertexId = 68 CrimeId = 50ed1vertexId = 267 CrimeId = 662e9
ertexId = 362 CrimeId = fe7aevertexId = 454 CrimeId = 8d2d7vertexId = 450 CrimeId = 801c9vertexId = 295 CrimeId = d1ba6
ertexId = 200 CrimeId = 14b0bvertexId = 75 CrimeId = d2c1dvertexId = 437 CrimeId = 9bdeevertexId = 263 CrimeId = d9298
ertexId = 315 CrimeId = 43aa8vertexId = 257 CrimeId = 0c956vertexId = 556 CrimeId = 3a496vertexId = 424 CrimeId = 76697
ertexId = 11 CrimeId = a0defvertexId = 409 CrimeId = f3ee1vertexId = 24 CrimeId = ca598vertexId = 561 CrimeId = 4d2c0
ertexId = 287 CrimeId = dded9vertexId = 490 CrimeId = ef0b1vertexId = 392 CrimeId = e1b9bvertexId = 89 CrimeId = d8c20
ertexId = 361 CrimeId = 6b397vertexId = 510 CrimeId = b82a2vertexId = 473 CrimeId = 09e90vertexId = 149 CrimeId = 18474
ertexId = 571 CrimeId = ffe9evertexId = 488 CrimeId = 2d349vertexId = 546 CrimeId = 8bdfbvertexId = 329 CrimeId = c5f90
ertexId = 584 CrimeId = e701bvertexId = 134 CrimeId = cdfcfvertexId = 299 CrimeId = 2ccf2vertexId = 356 CrimeId = 421bf
ertexId = 423 CrimeId = c15c0vertexId = 15 CrimeId = cec4cvertexId = 403 CrimeId = 595f8vertexId = 293 CrimeId = 803cc
ertexId = 171 CrimeId = 3d168vertexId = 318 CrimeId = cfcbevertexId = 176 CrimeId = fdcf4vertexId = 142 CrimeId = f5b0f
ertexId = 578 CrimeId = 83178vertexId = 566 CrimeId = 3eee1vertexId = 406 CrimeId = bdafbvertexId = 71 CrimeId = ba3ae
ertexId = 136 CrimeId = 53257vertexId = 83 CrimeId = 5da87vertexId = 285 CrimeId = a833dvertexId = 88 CrimeId = 2d0ae
ertexId = 79 CrimeId = 34272vertexId = 470 CrimeId = e8c89vertexId = 385 CrimeId = 2a25cvertexId = 351 CrimeId = d8bd1
ertexId = 199 CrimeId = 9e51cvertexId = 236 CrimeId = 4dcafvertexId = 487 CrimeId = a7b0fvertexId = 65 CrimeId = e85ee
ertexId = 244 CrimeId = 44fb2vertexId = 53 CrimeId = 36caevertexId = 548 CrimeId = cd05fvertexId = 497 CrimeId = fa4e7
ertexId = 43 CrimeId = e61ccvertexId = 8   CrimeId = 4bcd6vertexId = 155 CrimeId = 42640vertexId = 525 CrimeId = 6a7d5
ertexId = 36 CrimeId = c3298vertexId = 280 CrimeId = ec3b1vertexId = 405 CrimeId = f9816vertexId = 179 CrimeId = 6afd9
ertexId = 52 CrimeId = 5aad7vertexId = 580 CrimeId = a30f9vertexId = 452 CrimeId = 29645vertexId = 377 CrimeId = e0ca5
ertexId = 544 CrimeId = 4fc27vertexId = 77 CrimeId = 38c6evertexId = 346 CrimeId = 049d1vertexId = 310 CrimeId = cc9f3
ertexId = 558 CrimeId = 63b73vertexId = 502 CrimeId = 110bevertexId = 348 CrimeId = 937c0vertexId = 429 CrimeId = b37e5
ertexId = 223 CrimeId = d6ebfvertexId = 478 CrimeId = 5bfe3vertexId = 44 CrimeId = 31838vertexId = 459 CrimeId = 93956

MST Tour Weight(meters) = 513326.09539907286

After Ant Colony opt Weight = 632221.2026585272

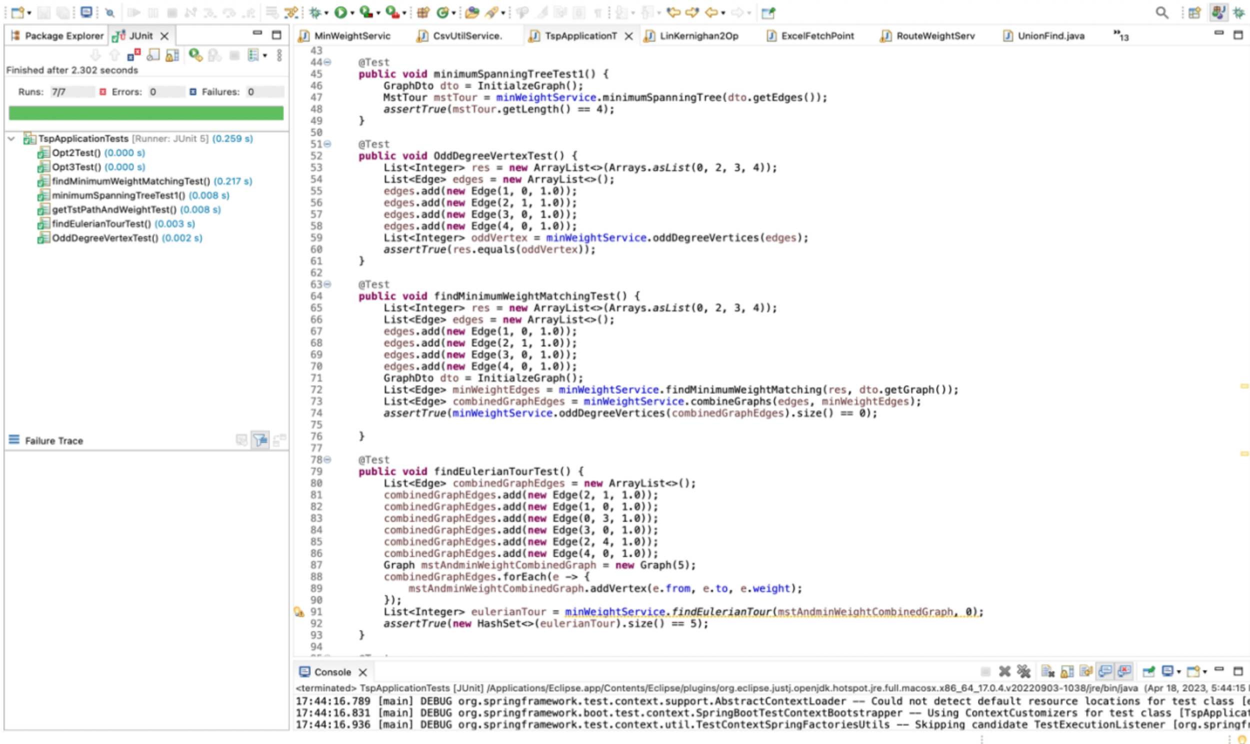
Approx Ratio= Ant/MST= 632115.7917132177/513326.09539907286

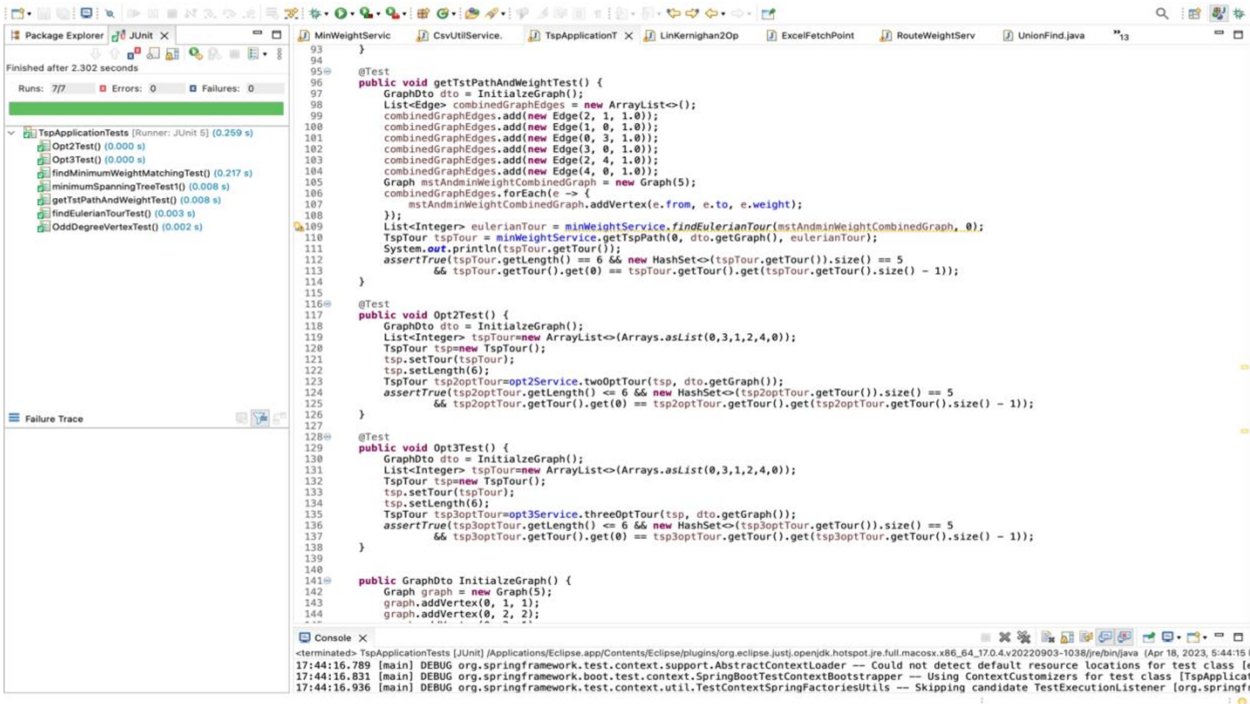
Approximation Ratio=1.232



Unit Tests:

Below are the Unit Tets results along with Code ScreenShots :





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