BIRZEIT UNIVERSITY

Faculty of Engineering and Technology

Electrical and Computer Engineering Department

ENCS3340 - Artificial Intelligence

Project #1 - Optimization Strategies for Local Package Delivery Operations

Prepared by:

Duha Imad 1220623

Leen Anas 1220380

Instructor's Name:

Dr. Yazan Abu Farha

Section: 4

4th May 2025

❖ About The Project

The project was structured as a series of steps to simulate and optimize a local package delivery system. It began with data representation through randomly generated delivery points, vehicle definitions, and assigning priority levels to each package. The objective was to minimize the total distance traveled by delivery vehicles while respecting vehicle capacity constraints and delivery priorities (as a soft constraint).

We implemented the solution using Python and developed an interface using Flask, HTML, ,CSS, and javaScript. The user can input delivery data either manually through the web form or by uploading a file. The backend parses the input, validates the data, and then triggers the optimization algorithm selected by the user — either Simulated Annealing or Genetic Algorithm.

The delivery routing task is framed as a constrained combinatorial optimization problem. Inputs include packages (with location, weight, and priority), vehicles with capacity limits, and the number of vehicles. The goal is to assign packages to vehicles and determine delivery sequences to minimize total travel distance, while prioritizing early delivery of high-priority packages. Constraints ensure vehicle capacities aren't exceeded, each package is delivered once if feasible, and all routes start and end at the depot.

Simulated Annealing

If the Simulated Annealing method is selected, the algorithm starts by generating an initial random solution, which assigns packages to vehicles and defines delivery routes arbitrarily. This state acts as the starting point for optimization. While the temperature remains above one, the algorithm repeatedly generates new solutions by swapping packages between vehicles, and changing the delivery order within a vehicle. Each new solution is evaluated by calculating the total travel distance, factoring in priority penalties. If it yields a shorter distance than the current best, it is accepted as the new best solution. Otherwise, it may still be accepted with a probability that decreases as the temperature drops which helps in escaping the local minima.

Constraint handling is managed by discarding and regenerating any infeasible neighbors, such as those where vehicles exceed their capacity. The algorithm only works with valid solutions during optimization. For parameter tuning, the initial temperature was set to 1000 to explore more options at the beginning. We tested different cooling rates from 0.90 to 0.99, and chose 0.95 because it gave good results without taking too long. Slower cooling helped explore better solutions but made the process slower. Once the temperature falls below the threshold, the algorithm stops, and the best solution found is displayed on the UI. The result includes the optimized delivery assignments and routes, illustrated using figures of test cases below for clarity.

Genetic Algorithm

If the user selects the Genetic Algorithm, the optimization begins by filtering out oversized packages that cannot be handled by any available vehicle. These are set aside and reported. Next, an initial population of valid delivery solutions is generated. Each individual represents an assignment of packages to vehicles and an order of delivery. The individuals are created using a heuristic that attempts to assign packages to vehicles without exceeding capacity constraints, ordering heavier packages first. Once the initial population is ready, the algorithm begins evolving the population over 500 generations where each population has 70 individuals in it.

In each generation, fitness evaluation is performed where each individual's fitness is calculated based on a greedy solution for the problem (to get the worst possible values) minus the total distance by all vehicles (greater number means better fitness). The distance is normalized to make sure it is a number between 0 and 1. a priority score favoring earlier delivery of high-priority packages, and penalties for unassigned packages which reduce overall fitness. This happens by calculating priority points for each individual better ordering is better score and after calculating that score,it is normalized using the distances part of the fitness, low distances will make the priority not that effected. Hence, very bad distance with good priority will not be chosen. For parent selection, two individuals are chosen using roulette-wheel selection (fitness-proportional), giving better individuals a higher chance of being selected. During crossover, a single-point crossover combines routes from both parents while ensuring feasibility by avoiding package duplication and respecting capacity limits. If the offspring is infeasible, a repair or regeneration step is applied. Mutation occurs with a low probability (0.05), where children may change by swapping packages between routes or reordering a vehicle's delivery list, which helps maintain diversity and prevents the algorithm from getting stuck in a bad solution too early. To make sure good solutions aren't lost, the two best ones from each generation are kept unchanged. The rest of the population is filled with new children created from crossover and mutation.

The constraint handling process ensures that all generated individuals are verified against predefined constraints, discarding or correcting infeasible individuals, such as those with overloaded vehicles. For parameter tuning, several factors were considered. The population size was tested within the range of 30 to 100, with **70** being selected as the optimal balance between solution quality and performance. A mutation rate of 0.05 was found to be enough to keep diversity without upsetting the population. The number of generations was set to 500, providing good results without taking too much time. Increasing the generations further did improve the results, but the improvements became smaller. After completing all generations, the algorithm picks the best solution, which covers all packages (if possible) with the shortest distance and highest priority (possible). Any unassigned packages are also noted. The best solution and total travel distance are then sent to the frontend for visualization.

User Interface

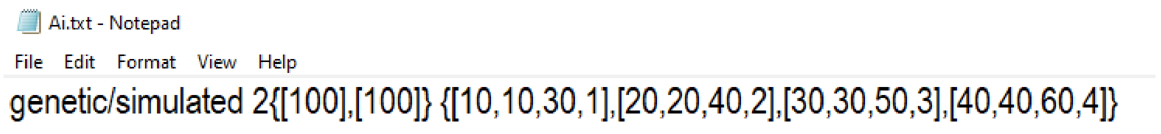
The UI shows the results of the optimization algorithm with text and a visual map. It starts with a "Results" title and a canvas for drawing vehicle paths. Below the canvas, there's a section that displays details like the algorithm used, total distance, vehicle assignments, and skipped packages if there are any.

The canvas shows the movement of vehicles, drawing lines between package destinations. The vehicle paths are animated, with each vehicle stopping briefly at each destination. There are axes on the canvas to help show the positions. If no result is found, a message is displayed instead of the results. The UI uses both text and animation to clearly show how the algorithm's solution works.

❖ Test Cases

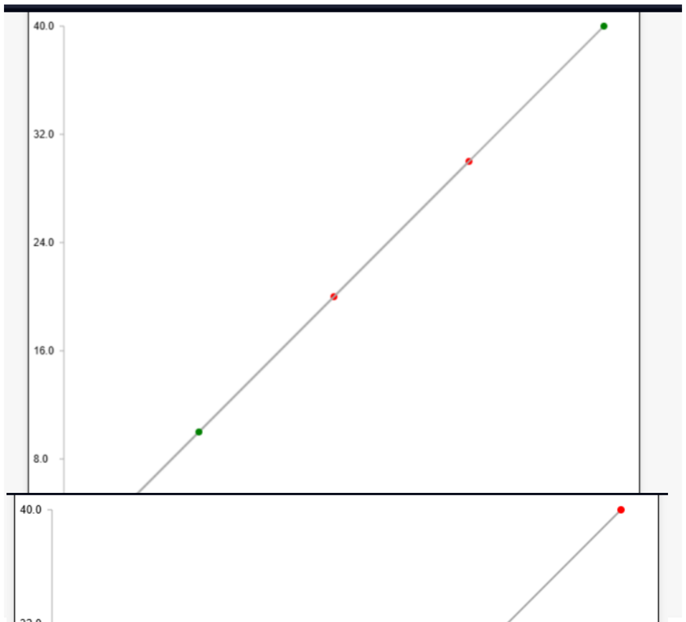
Test Case 1: Basic Feasibility Test

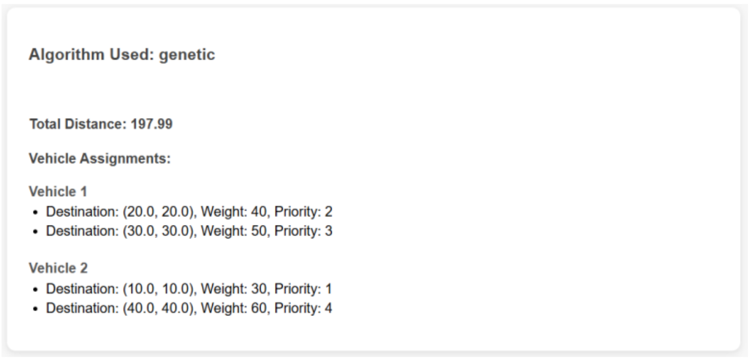
Input:



```
genetic/simulated 2{[100],[100]} {[10,10,30,1],[20,20,40,2],[30,30,50,3],[40,40,60,4]}
```

Outputs: The first two figures show the genetic results, the second shows the simu.





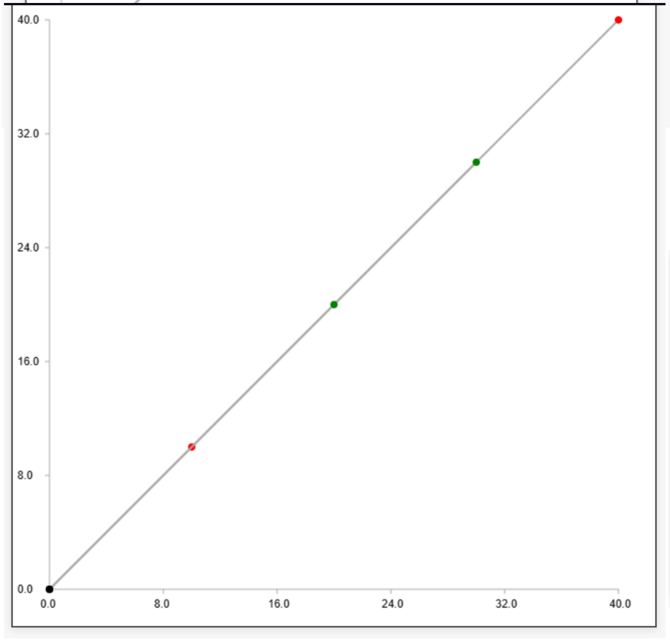
Algorithm Used: genetic

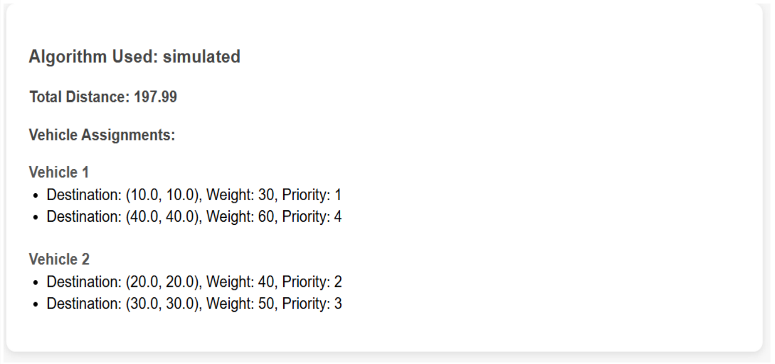
Total Distance: 197.99

Vehicle Assignments:

Vehicle 1
- Destination: (20.0, 20.0), Weight: 40, Priority: 2
- Destination: (30.0, 30.0), Weight: 50, Priority: 3

Vehicle 2
- Destination: (10.0, 10.0), Weight: 30, Priority: 1
- Destination: (40.0, 40.0), Weight: 60, Priority: 4





Algorithm Used: simulated

Total Distance: 197.99

Vehicle Assignments:

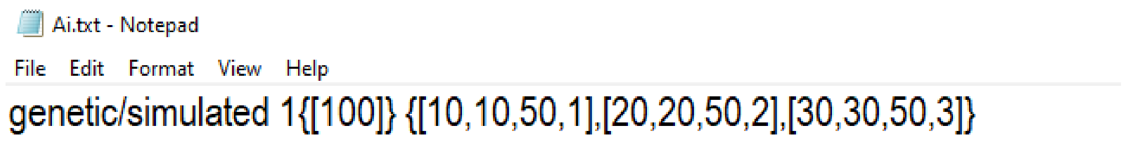
Vehicle 1
- Destination: (10.0, 10.0), Weight: 30, Priority: 1
- Destination: (40.0, 40.0), Weight: 60, Priority: 4

Vehicle 2
- Destination: (20.0, 20.0), Weight: 40, Priority: 2
- Destination: (30.0, 30.0), Weight: 50, Priority: 3

The outputs is as expected as shown in the figures above, the packages are distributed among vehicle and no vehicle carries more than 100kg and all the packages are assigned.

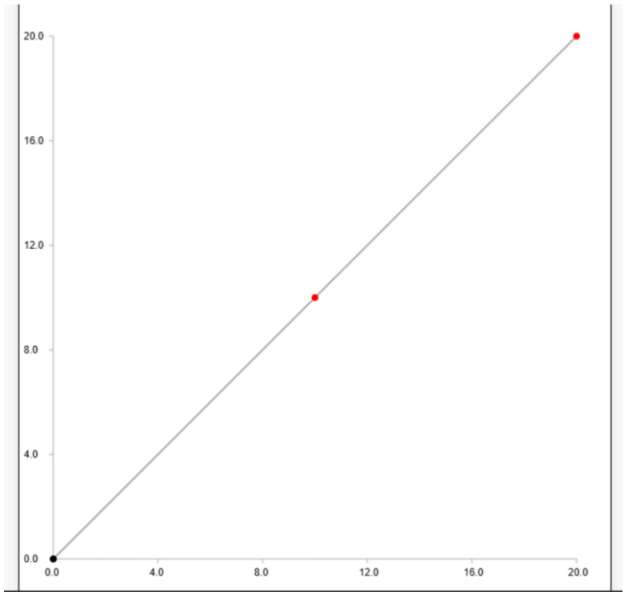
Test Case 2: Priority Handling Test

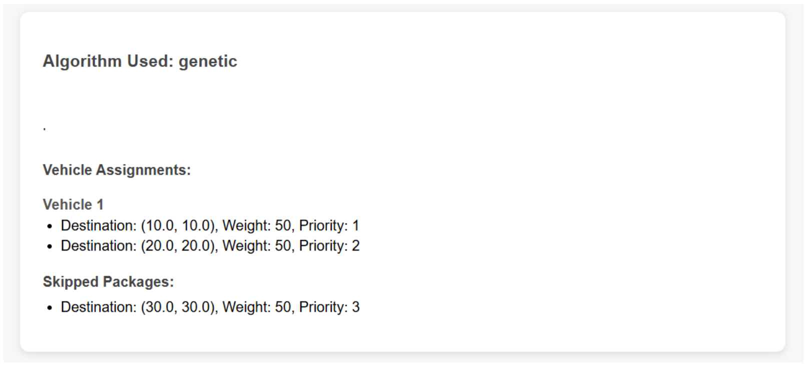
Input:



```
genetic/simulated 1{[100]} {[10,10,50,1],[20,20,50,2],[30,30,50,3]}
```

Output:





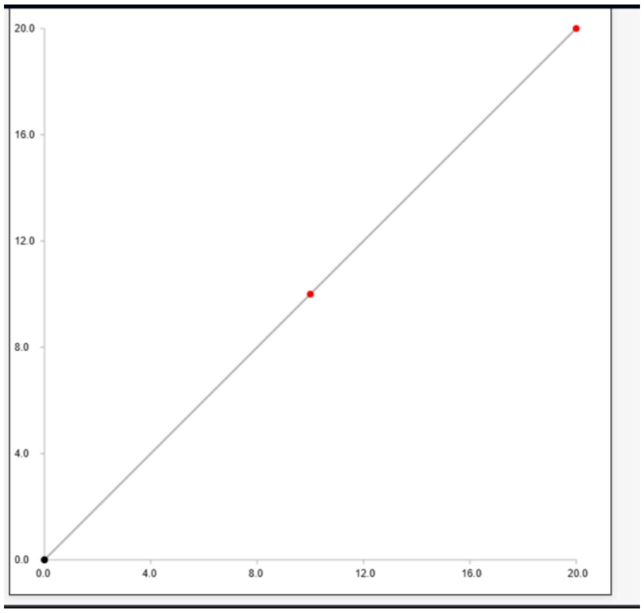
Algorithm Used: genetic

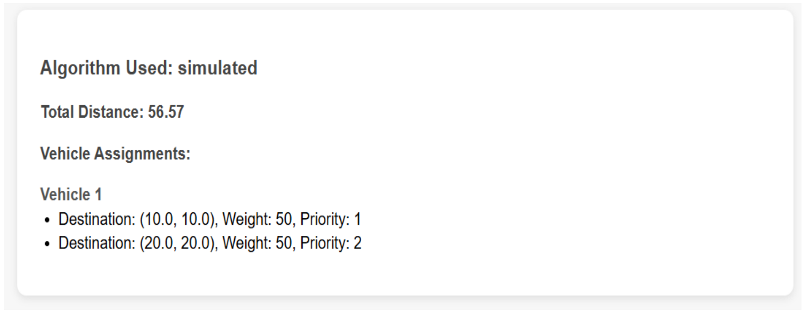
.

Vehicle Assignments:

Vehicle 1
- Destination: (10.0, 10.0), Weight: 50, Priority: 1
- Destination: (20.0, 20.0), Weight: 50, Priority: 2

Skipped Packages:
- Destination: (30.0, 30.0), Weight: 50, Priority: 3





Algorithm Used: simulated

Total Distance: 56.57

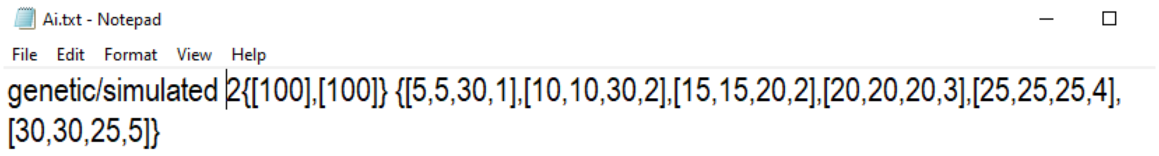
Vehicle Assignments:

Vehicle 1
- Destination: (10.0, 10.0), Weight: 50, Priority: 1
- Destination: (20.0, 20.0), Weight: 50, Priority: 2

As expected the packages are selected for delivery depending on the higher priority, and the third package is unassigned due to capacity constraints. This is shown in the figures above.

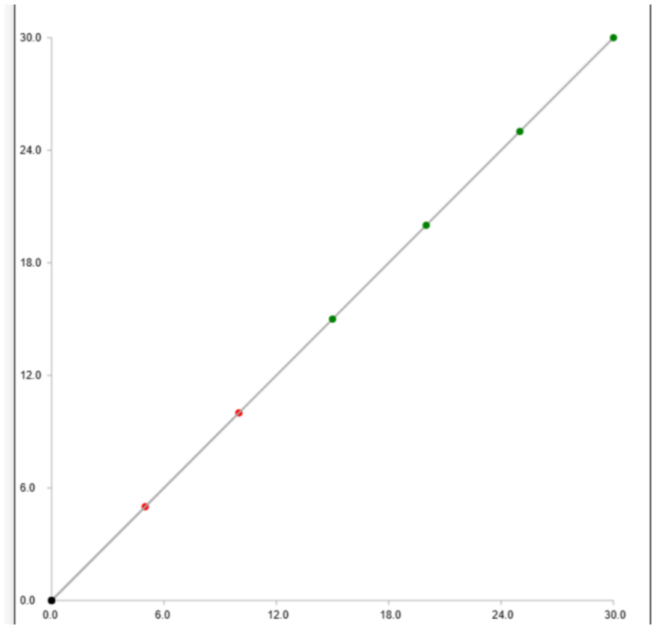
Test Case 3: Distance Optimization Test

Input:



```
genetic/simulated 2{[100],[100]} {[5,5,30,1],[10,10,30,2],[15,15,20,2],[20,20,20,3],[25,25,25,4],
[30,30,25,5]}
```

Output:



Algorithm Used: genetic

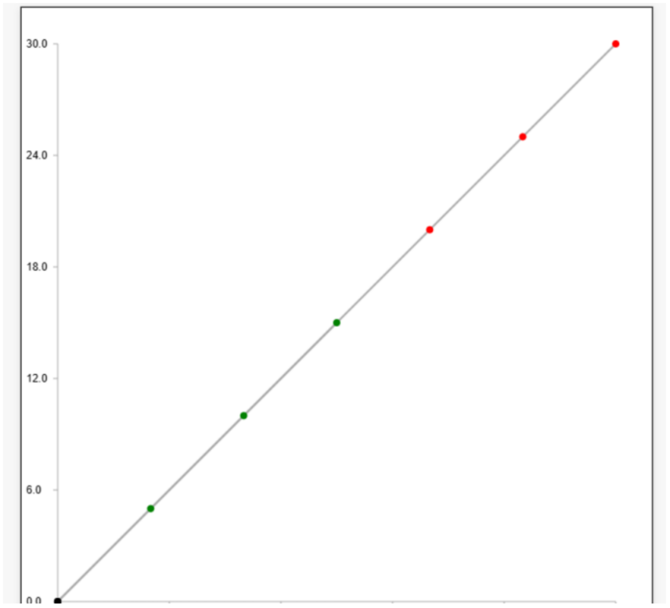
Best Fitness: 0.00

Total Distance: 113.14

Vehicle Assignments:

Vehicle 1
- Destination: (5.0, 5.0), Weight: 30, Priority: 1
- Destination: (10.0, 10.0), Weight: 30, Priority: 2

Vehicle 2
- Destination: (15.0, 15.0), Weight: 20, Priority: 2
- Destination: (20.0, 20.0), Weight: 20, Priority: 3
- Destination: (25.0, 25.0), Weight: 25, Priority: 4
- Destination: (30.0, 30.0), Weight: 25, Priority: 5



Algorithm Used: simulated

Total Distance: 127.28

Vehicle Assignments:

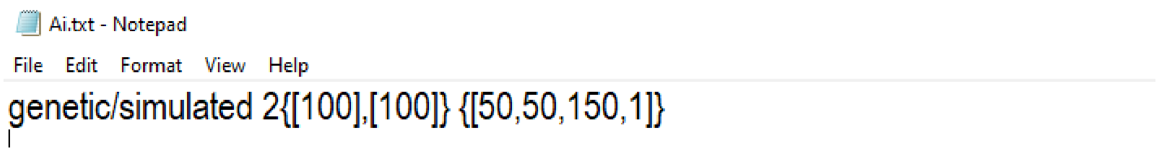
Vehicle 1
- Destination: (20.0, 20.0), Weight: 20, Priority: 3
- Destination: (25.0, 25.0), Weight: 25, Priority: 4
- Destination: (30.0, 30.0), Weight: 25, Priority: 5

Vehicle 2
- Destination: (5.0, 5.0), Weight: 30, Priority: 1
- Destination: (10.0, 10.0), Weight: 30, Priority: 2
- Destination: (15.0, 15.0), Weight: 20, Priority: 2

The Packages are assigned and routed to minimize the combined distance travelled by both vehicles.

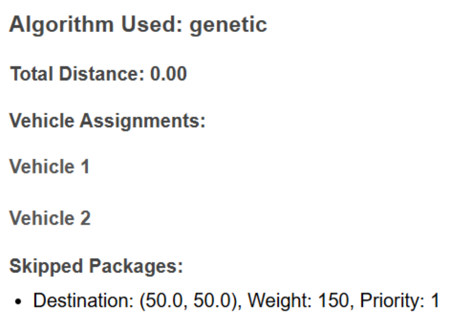
Test Case 4: Edge Case - Overcapacity Package

Input:

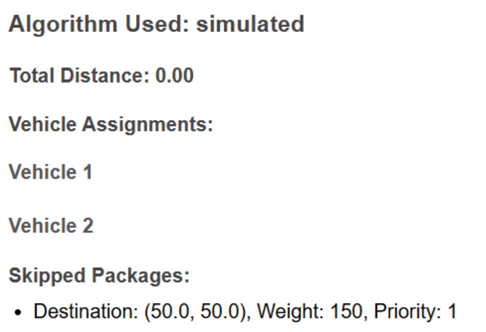


```
genetic/simulated 2{[100],[100]} {[50,50,150,1]}
```

Output:



```
Algorithm Used: genetic

Total Distance: 0.00

Vehicle Assignments:

Vehicle 1

Vehicle 2

Skipped Packages:

• Destination: (50.0, 50.0), Weight: 150, Priority: 1
```

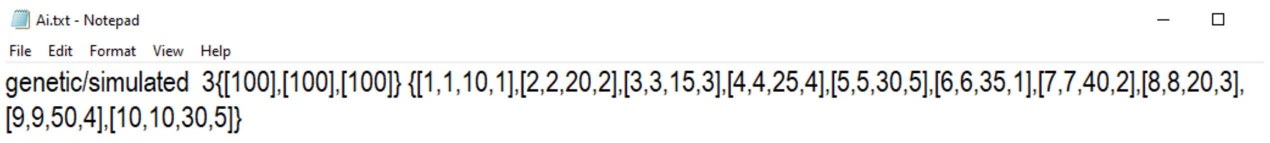


```
Algorithm Used: simulated

Total Distance: 0.00

Vehicle Assignments:

Vehicle 1

Vehicle 2

Skipped Packages:

• Destination: (50.0, 50.0), Weight: 150, Priority: 1
```

Since the package exceeds the vehicles capacity, then its not assigned to any of them, and the package is shown as a slipped package as illustrated in the figures.

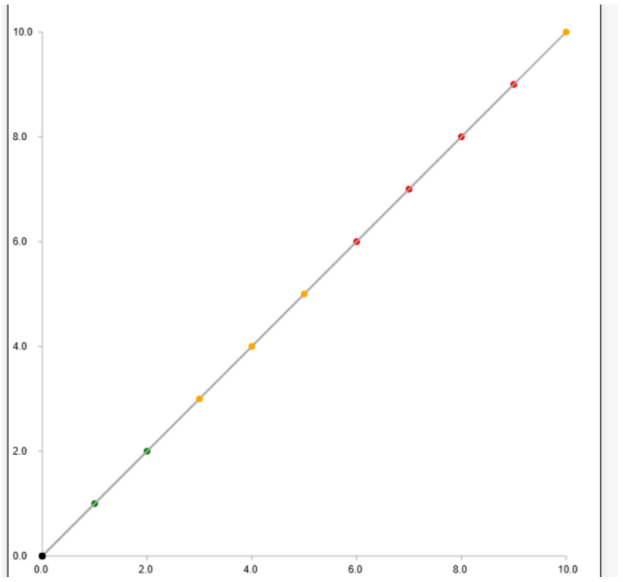
Test Case 5: Simulated Annealing vs. Genetic Algorithm Comparison

Input:



```
genetic/simulated  3{[100],[100],[100]} {[1,1,10,1],[2,2,20,2],[3,3,15,3],[4,4,25,4],[5,5,30,5],[6,6,35,1],[7,7,40,2],[8,8,20,3],
[9,9,50,4],[10,10,30,5]}
```

Output:



Algorithm Used: genetic

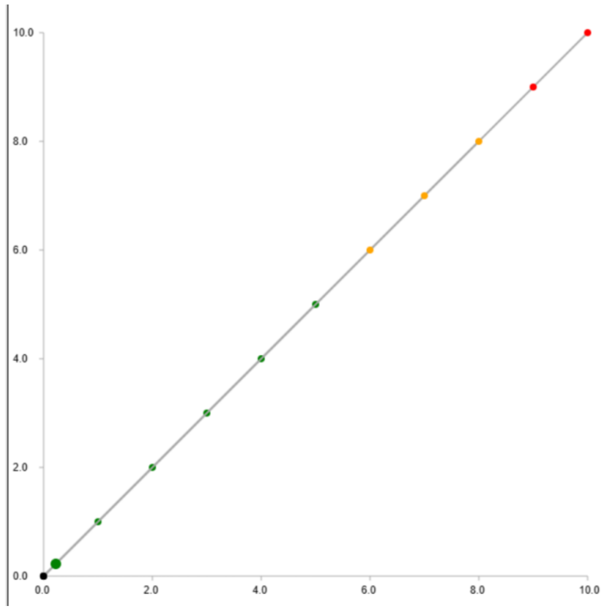
Total Distance: 59.40

Vehicle Assignments:

Vehicle 1
- Destination: (6.0, 6.0), Weight: 35, Priority: 1
- Destination: (7.0, 7.0), Weight: 40, Priority: 2
- Destination: (8.0, 8.0), Weight: 20, Priority: 3
- Destination: (9.0, 9.0), Weight: 50, Priority: 4

Vehicle 2
- Destination: (1.0, 1.0), Weight: 10, Priority: 1
- Destination: (2.0, 2.0), Weight: 20, Priority: 2

Vehicle 3
- Destination: (3.0, 3.0), Weight: 15, Priority: 3
- Destination: (4.0, 4.0), Weight: 25, Priority: 4
- Destination: (5.0, 5.0), Weight: 30, Priority: 5
- Destination: (10.0, 10.0), Weight: 30, Priority: 5



Algorithm Used: simulated

Total Distance: 65.05

Vehicle Assignments:

Vehicle 1
- Destination: (9.0, 9.0), Weight: 50, Priority: 4
- Destination: (10.0, 10.0), Weight: 30, Priority: 5

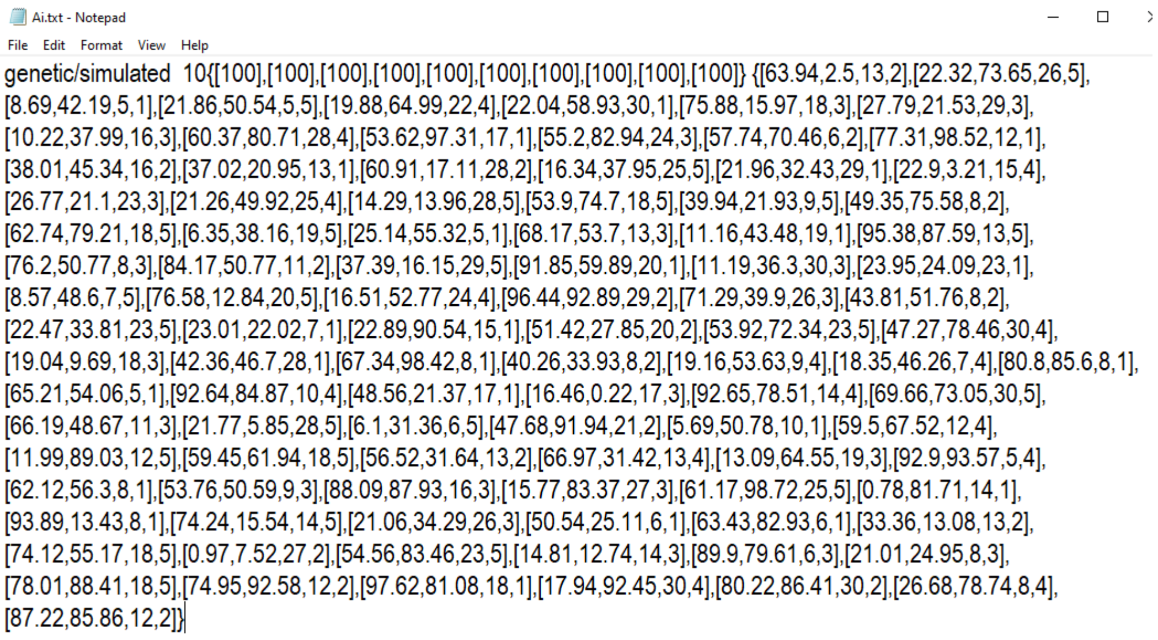
Vehicle 2
- Destination: (1.0, 1.0), Weight: 10, Priority: 1
- Destination: (2.0, 2.0), Weight: 20, Priority: 2
- Destination: (3.0, 3.0), Weight: 15, Priority: 3
- Destination: (4.0, 4.0), Weight: 25, Priority: 4
- Destination: (5.0, 5.0), Weight: 30, Priority: 5

Vehicle 3
- Destination: (6.0, 6.0), Weight: 35, Priority: 1
- Destination: (7.0, 7.0), Weight: 40, Priority: 2
- Destination: (8.0, 8.0), Weight: 20, Priority: 3

As we can see in the results, both of the solutions gave a valid output the difference in the distance is low indicating that both algorithms give a valid result.

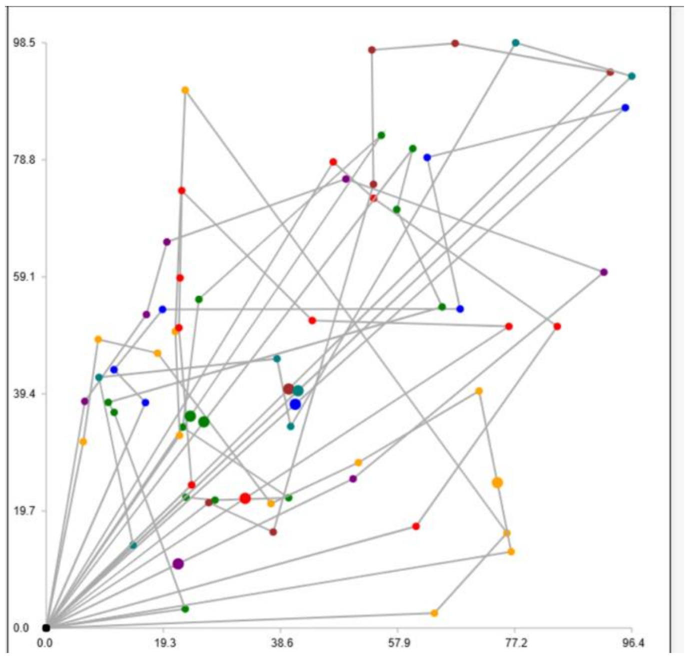
Test Case 6: Scalability Test

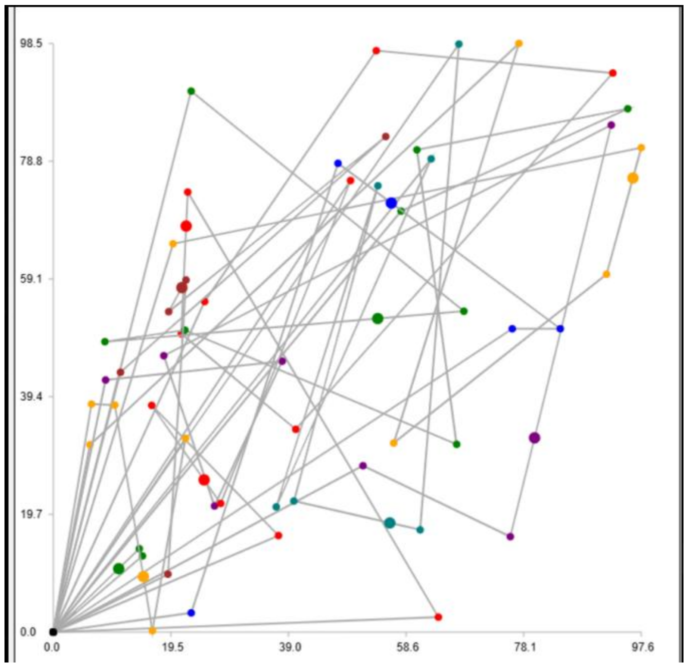
Input:



```
genetic/simulated  10{[100],[100],[100],[100],[100],[100],[100],[100],[100],[100]} {[63.94,2.5,13,2],[22.32,73.65,26,5],
[8.69,42.19,5,1],[21.86,50.54,5,5],[19.88,64.99,22,4],[22.04,58.93,30,1],[75.88,15.97,18,3],[27.79,21.53,29,3],
[10.22,37.99,16,3],[60.37,80.71,28,4],[53.62,97.31,17,1],[55.2,82.94,24,3],[57.74,70.46,6,2],[77.31,98.52,12,1],
[38.01,45.34,16,2],[37.02,20.95,13,1],[60.91,17.11,28,2],[16.34,37.95,25,5],[21.96,32.43,29,1],[22.9,3.21,15,4],
[26.77,21.1,23,3],[21.26,49.92,25,4],[14.29,13.96,28,5],[53.9,74.7,18,5],[39.94,21.93,9,5],[49.35,75.58,8,2],
[62.74,79.21,18,5],[6.35,38.16,19,5],[25.14,55.32,5,1],[68.17,53.7,13,3],[11.16,43.48,19,1],[95.38,87.59,13,5],
[76.2,50.77,8,3],[84.17,50.77,11,2],[37.39,16.15,29,5],[91.85,59.89,20,1],[11.19,36.3,30,3],[23.95,24.09,23,1],
[8.57,48.6,7,5],[76.58,12.84,20,5],[16.51,52.77,24,4],[96.44,92.89,29,2],[71.29,39.9,26,3],[43.81,51.76,8,2],
[22.47,33.81,23,5],[23.01,22.02,7,1],[22.89,90.54,15,1],[51.42,27.85,20,2],[53.92,72.34,23,5],[47.27,78.46,30,4],
[19.04,9.69,18,3],[42.36,46.7,28,1],[67.34,98.42,8,1],[40.26,33.93,8,2],[19.16,53.63,9,4],[18.35,46.26,7,4],[80.8,85.6,8,1],
[65.21,54.06,5,1],[92.64,84.87,10,4],[48.56,21.37,17,1],[16.46,0.22,17,3],[92.65,78.51,14,4],[69.66,73.05,30,5],
[66.19,48.67,11,3],[21.77,5.85,28,5],[6.1,31.36,6,5],[47.68,91.94,21,2],[5.69,50.78,10,1],[59.5,67.52,12,4],
[11.99,89.03,12,5],[59.45,61.94,18,5],[56.52,31.64,13,2],[66.97,31.42,13,4],[13.09,64.55,19,3],[92.9,93.57,5,4],
[62.12,56.3,8,1],[53.76,50.59,9,3],[88.09,87.93,16,3],[15.77,83.37,27,3],[61.17,98.72,25,5],[0.78,81.71,14,1],
[93.89,13.43,8,1],[74.24,15.54,14,5],[21.06,34.29,26,3],[50.54,25.11,6,1],[63.43,82.93,6,1],[33.36,13.08,13,2],
[74.12,55.17,18,5],[0.97,7.52,27,2],[54.56,83.46,23,5],[14.81,12.74,14,3],[89.9,79.61,6,3],[21.01,24.95,8,3],
[78.01,88.41,18,5],[74.95,92.58,12,2],[97.62,81.08,18,1],[17.94,92.45,30,4],[80.22,86.41,30,2],[26.68,78.74,8,4],
[87.22,85.86,12,2]}
```

Output:





the main goal of this test case is to make sure both algos will take reasonable time and wont crash for large number of packages and that is exactly what we got. The annealing showed results in few seconds. also, the genetic showed it in less than minute (and obviously we got results from both algorithms)

Test Case 7: User Interface Display Test

As shown in the previous test cases, the UI presents vehicle routes, package assignments, and relevant metrics clearly. All information is accurate and user-friendly, and there are no display errors or inconsistencies. Also we added animation to the UI in order to make it easy to understand how the packages are delivered.