Faculty of Engineering & Technology

Electrical & Computer Engineering Department

Circuits & Electronics Laboratory – ENCS3340

Report for project Optimization Strategies for Local Package Delivery Operations

---

Prepared by:

PartnerOneName: Aya AbuSneineh	**ID_1:** 1221414	Section : 1
PartnerTwoName: Nagham Halloum	**ID_2:** 1220965	Section : 4

Date : 5/1/2025

Instructor: Dr . Yazan AbuFarha

Table of Contents

Table of Figures

List of Tables :

Problem Formulation :

1. **Initial State**

   A solution is initialized by randomly assigning packages to vehicles such that the total weight of the packages assigned to any vehicle does not exceed its capacity. Within each vehicle, the delivery order of the packages is also randomized.

2. **Actions (Successor Function)**

   The way neighbor states are generated differs slightly between algorithms:

   - **Simulated Annealing (SA)**: A single neighbor is generated by applying one of the following random changes:
     - Swapping two packages between different vehicles (if capacity constraints allow).
     - Reordering the delivery sequence within a vehicle.
     - Moving a package from one vehicle to another.
   - **Genetic Algorithm (GA)**: A population of solutions evolves using:
     - **Crossover** between two parent solutions to produce offspring.
     - **Mutation**, where small changes are made to a solution similar to the actions used in SA.

3. **State Space**

   The state space includes all possible combinations of package-to-vehicle assignments and all possible delivery orders within each vehicle. This space is extremely large due to the combinatorial nature of the problem. Therefore, heuristic and metaheuristic algorithms are used to explore it efficiently without exhaustively evaluating every possible state.

4. **State Representation**

   A state represents:

   - The assignment of each package to a specific vehicle.
   - The order in which each vehicle will deliver its assigned packages.

5. **Goal Test**

   Since this is an optimization problem rather than a classical search problem, the goal is not a specific state but rather a *good* or *optimal* one. Each algorithm defines its own stopping condition:

   - **Simulated Annealing**: Stops when the temperature drops below a predefined threshold.
   - **Genetic Algorithm**: Stops after a fixed number of generations or when there is no significant improvement in fitness over time.

6. **Objective Function / Path Cost**

   The cost of a solution is calculated as the total distance traveled by all vehicles (using Euclidean distance). A penalty is added if capacity constraints are violated or if high-priority packages are delivered too late (after lower-priority ones). The lower the cost, the better the solution.

7. **Solution Definition**

   A complete solution includes:

   - A feasible assignment of packages to vehicles that respects vehicle capacities.
   - A defined delivery sequence for the packages in each vehicle that attempts to minimize total travel distance and priority violations.

Using Heuristics in Algorithms

1 . Simulated Annealing :

Neighbor Generation using random moves such as **swap, reorder, and move** was implemented to generate new solutions randomly, helping explore the solution space more widely.

Acceptance or rejection of moves is based on cost improvement; if the cost is better, the move is accepted directly . If the cost is higher, it may still be accepted with a probability calculated using the formula: P = $e^{\frac{-\Delta cost}{T}}$ This helps avoid getting trapped in local optima, especially during the early stages of the search. The total cost is calculated based on the total distance traveled by all vehicles, in addition to the priority delay penalty. The distance is calculated by considering both the distance from the shop to the package and between the packages, while the penalty is applied if a lower-priority package is delivered after a higher-priority one, taking into account the delay ratio and priority impact.

Cooling is an essential component of the algorithm, as it helps transition from broad exploration to more precise exploitation as the solution approaches the optimal.

2. Genetic Algorithm:

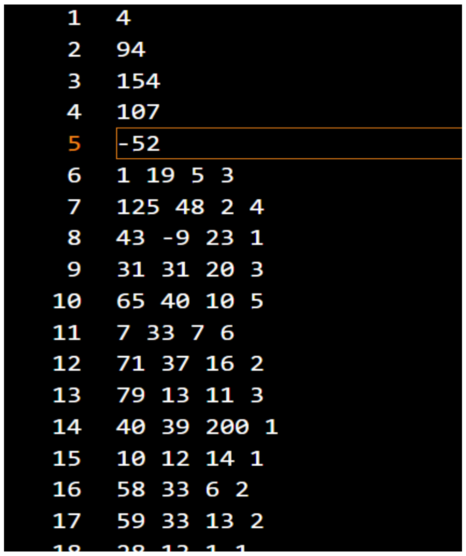
Tournament selection was used to randomly select the best solutions from a subset of the population, balancing selection pressure and maintaining diversity in the new generation.
Crossover combines features from parent solutions to generate new ones, enabling the creation of offspring that may outperform the originals, while ensuring feasibility and constraint satisfaction.
Mutation introduces random changes to the solutions, such as **swapping packages between vehicles**, **reordering packages within a vehicle**, or **moving a package from one vehicle to another**, which increases the chance of discovering high-quality solutions and escaping local optima. It was also ensured that mutations do not violate constraints, such as **exceeding vehicle capacity**.

The cost is calculated in the same way as it was in Simulated Annealing

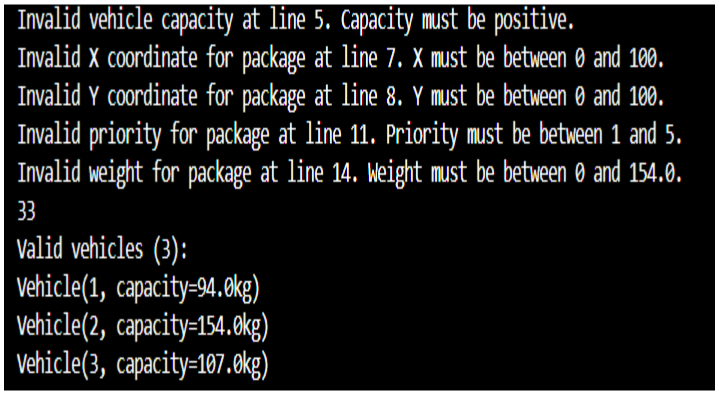
Constraint Handling and Input Validation in the System

1. Input File Validation

When reading data from the input file, both vehicle and package information are validated before being used. Vehicles must have a positive capacity value. If a vehicle has zero or negative capacity, it is ignored, and an error message is printed. Similarly, for packages, the program checks that the weight is positive and does not exceed the maximum capacity among all vehicles. It also validates that the priority is between 1 and 5. Any invalid package is skipped, and a warning is shown. This input validation helps prevent the program from processing incorrect or logically impossible data.



```
 1   4
 2   94
 3   154
 4   107
 5   -52
 6   1 19 5 3
 7   125 48 2 4
 8   43 -9 23 1
 9   31 31 20 3
10   65 40 10 5
11   7 33 7 6
12   71 37 16 2
13   79 13 11 3
14   40 39 200 1
15   10 12 14 1
16   58 33 6 2
17   59 33 13 2
```

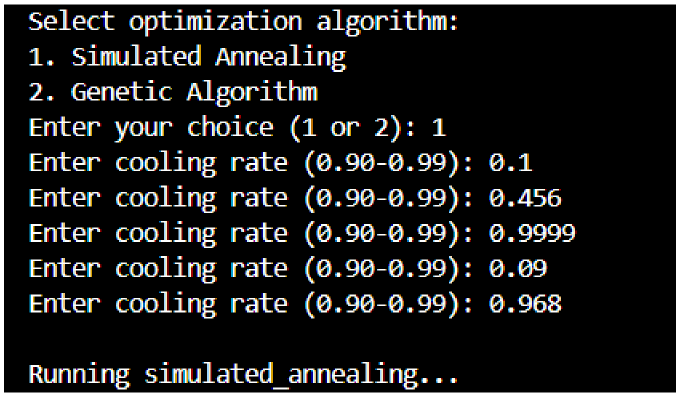


```
Invalid vehicle capacity at line 5. Capacity must be positive.
Invalid X coordinate for package at line 7. X must be between 0 and 100.
Invalid Y coordinate for package at line 8. Y must be between 0 and 100.
Invalid priority for package at line 11. Priority must be between 1 and 5.
Invalid weight for package at line 14. Weight must be between 0 and 154.0.
33
Valid vehicles (3):
Vehicle(1, capacity=94.0kg)
Vehicle(2, capacity=154.0kg)
Vehicle(3, capacity=107.0kg)
```

Figure 1 : Rejected packages due to invalid weight, coordinates, or priority

2. Verifying User Input at Runtime

Before running the algorithm, the user is asked to select the algorithm type and enter key parameters such as the cooling rate for Simulated Annealing, or population size and mutation rate for the Genetic Algorithm. The program checks that these values fall within reasonable bounds (e.g., cooling rate between 0.90 and 0.99). If an invalid value is entered, the user is prompted to re-enter it.



```
Select optimization algorithm:
1. Simulated Annealing
2. Genetic Algorithm
Enter your choice (1 or 2): 1
Enter cooling rate (0.90-0.99): 0.1
Enter cooling rate (0.90-0.99): 0.456
Enter cooling rate (0.90-0.99): 0.9999
Enter cooling rate (0.90-0.99): 0.09
Enter cooling rate (0.90-0.99): 0.968

Running simulated_annealing...
```

Figure 2 : Invalid user input: Cooling rate outside acceptable range

3. Handling Vehicle Capacity Constraints During Algorithm Execution

During the execution of the algorithms, the code always ensures that any proposed changes to the solution do not violate the vehicle capacity constraint. For example ,when swapping or reordering packages between vehicles, the program checks whether the vehicles can still carry the new set of packages without exceeding their capacity. If the change results in a capacity violation, it is rejected and the algorithm continues searching for a valid move. The same logic applies in the mutate function, where any package swap or change in order is only accepted if it doesn’t cause a vehicle to exceed its limit.

4. Validating Final Solutions

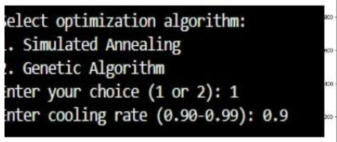
After the algorithm completes generating or improving a solution, it performs a final check to ensure that all vehicle capacity constraints are respected. It calculates the total weight of the packages assigned to each vehicle and confirms that none exceed their maximum capacity. If any vehicle is overloaded, the solution is marked as invalid.

Effects of parameters tuning

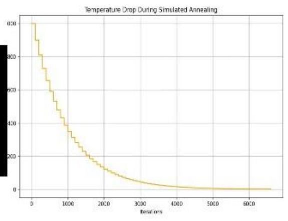
Simulated Annealing algorithm

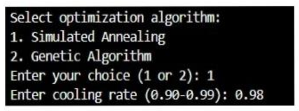
The **cooling rate** in the **Simulated Annealing** algorithm plays a crucial role in balancing **exploration and exploitation** during the optimization process. A **high cooling rate** (e.g., 0.98–0.99) leads to a **slower decrease in temperature**, allowing the algorithm to **explore the solution space more extensively** and increasing the chance of escaping local minima. However, this comes at the cost of **longer runtime** since more iterations are required to reach the stopping condition.

Conversely, a **lower cooling rate** (e.g., 0.90–0.93) causes the temperature to drop quickly, making the algorithm converge faster but **reducing its ability to escape local optima**, which may lead to **suboptimal solutions**. Therefore, tuning the cooling rate is essential to achieving a good trade-off between solution quality and computational efficiency. In our implementation, we observed that using a cooling rate between **0.95 and 0.99** provided the best balance for complex solution spaces such as **package delivery optimization**.

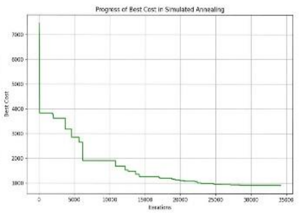


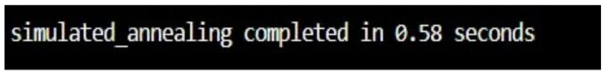
```
elect optimization algorithm:
. Simulated Annealing
. Genetic Algorithm
nter your choice (1 or 2): 1
nter cooling rate (0.90-0.99): 0.9
```



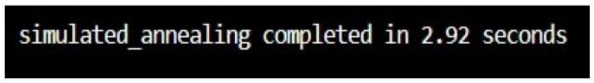


```
Select optimization algorithm:
1. Simulated Annealing
2. Genetic Algorithm
Enter your choice (1 or 2): 1
Enter cooling rate (0.90-0.99): 0.98
```

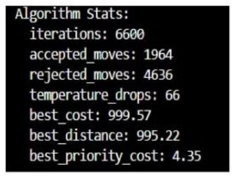




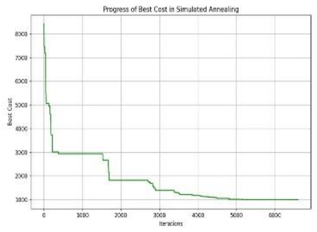
```
simulated_annealing completed in 0.58 seconds
```

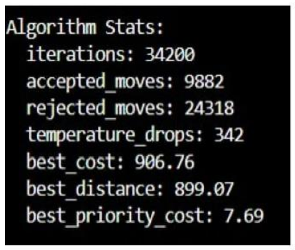


```
simulated_annealing completed in 2.92 seconds
```

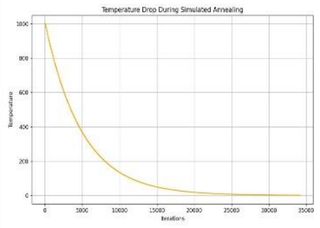


```
Algorithm Stats:
  iterations: 6600
  accepted_moves: 1964
  rejected_moves: 4636
  temperature_drops: 66
  best_cost: 999.57
  best_distance: 995.22
  best_priority_cost: 4.35
```





```
Algorithm Stats:
  iterations: 34200
  accepted_moves: 9882
  rejected_moves: 24318
  temperature_drops: 342
  best_cost: 906.76
  best_distance: 899.07
  best_priority_cost: 7.69
```

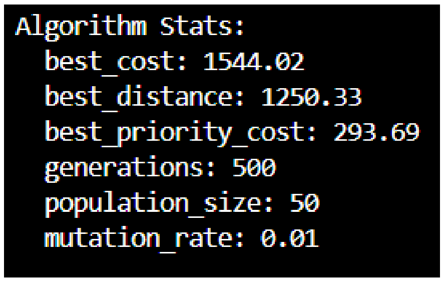


Genetic algorithm

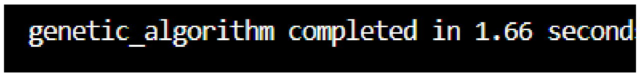
The population plays a **critical role** in the performance of the **Genetic algorithm**. The population size ranges between 50 and 100. When the population size is small (e.g., **50**), the algorithm **runs faster** since **fewer solutions** need to be evaluated in each **generation**. This can be beneficial in terms of **execution time**. However, having fewer individuals reduces **genetic diversity**, which increases the risk of the algorithm getting stuck in **local optima** and failing to discover **better solutions**. On the other hand, **increasing the population size** to 100 enhances **diversity** and allows the algorithm to explore a **wider range of possible solutions**, improving the chances of finding an **optimal or near-optimal result**. The trade-off, however, is that a **larger** population size **increases computational cost and execution time**. In summary, a larger population helps improve **solution quality** at the cost of **speed,** while a smaller population **saves time** but may lead to **suboptimal results**.

The mutation rate in a **genetic algorithm** controls how frequently random changes are applied to individuals in the **population**, thus influencing the balance between **exploration and exploitation of the solution space**. When using a mutation rate between 0.01 and 0.1, the performance of the algorithm is notably affected. A **low mutation rate, such as 0.01**, means changes to individuals will be infrequent or minimal. This can hinder the algorithm's ability to **explore new solutions**, potentially causing it to settle on **suboptimal solutions** or **get stuck in local minima**. However, it also **increases stability** by limiting unwanted or unnecessary changes.

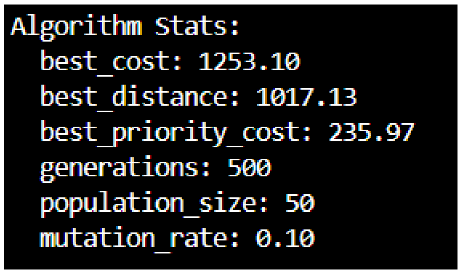
Conversely, increasing **the mutation rate to 0.1** increases the likelihood of significant changes to individuals. This allows the algorithm to **explore the solution space** more thoroughly, potentially **leading to better solutions**. However, a **higher mutation rate** can reduce stability, as larger changes might result in **undesirable or non-optimal solutions**. Therefore, achieving a balance between **exploration and stability is essential** .



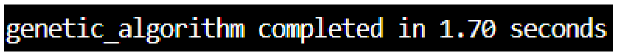
```
Algorithm Stats:
  best_cost: 1544.02
  best_distance: 1250.33
  best_priority_cost: 293.69
  generations: 500
  population_size: 50
  mutation_rate: 0.01
```



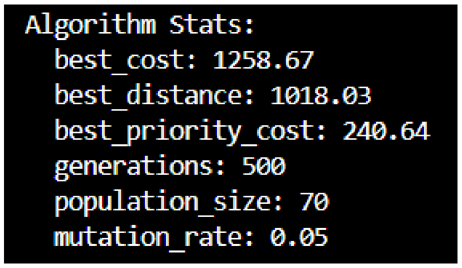
```
genetic_algorithm completed in 1.66 second
```

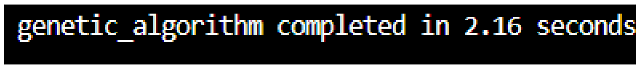


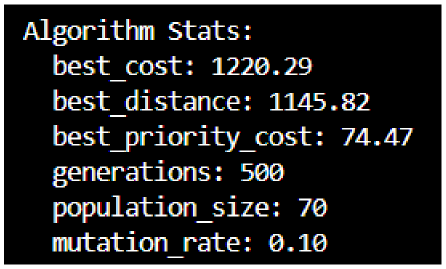
```
Algorithm Stats:
  best_cost: 1253.10
  best_distance: 1017.13
  best_priority_cost: 235.97
  generations: 500
  population_size: 50
  mutation_rate: 0.10
```

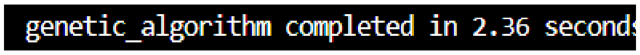


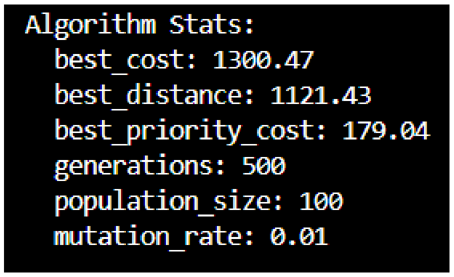
```
genetic_algorithm completed in 1.70 seconds
```

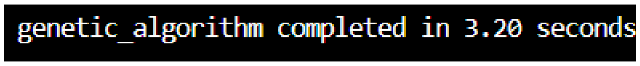

```
Algorithm Stats:
  best_cost: 1258.67
  best_distance: 1018.03
  best_priority_cost: 240.64
  generations: 500
  population_size: 70
  mutation_rate: 0.05
```

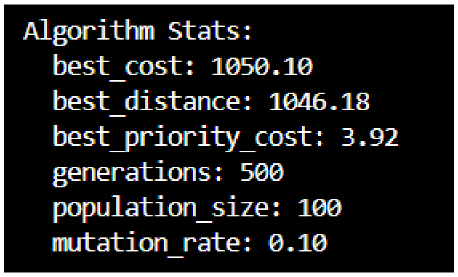

```
genetic_algorithm completed in 2.16 seconds
```


```
Algorithm Stats:
  best_cost: 1220.29
  best_distance: 1145.82
  best_priority_cost: 74.47
  generations: 500
  population_size: 70
  mutation_rate: 0.10
```


```
genetic_algorithm completed in 2.36 second
```


```
Algorithm Stats:
  best_cost: 1300.47
  best_distance: 1121.43
  best_priority_cost: 179.04
  generations: 500
  population_size: 100
  mutation_rate: 0.01
```


```
genetic_algorithm completed in 3.20 seconds
```


```
Algorithm Stats:
  best_cost: 1050.10
  best_distance: 1046.18
  best_priority_cost: 3.92
  generations: 500
  population_size: 100
  mutation_rate: 0.10
```

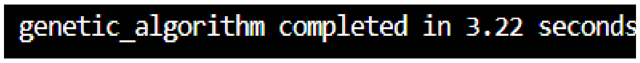
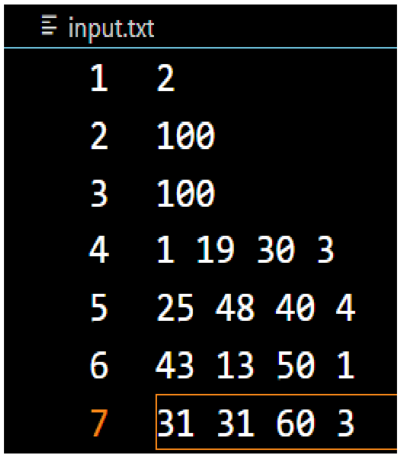

```
genetic_algorithm completed in 3.22 seconds
```

Figure 3 : Effect of different cooling rate values on temperature decay and solution cost over iterations

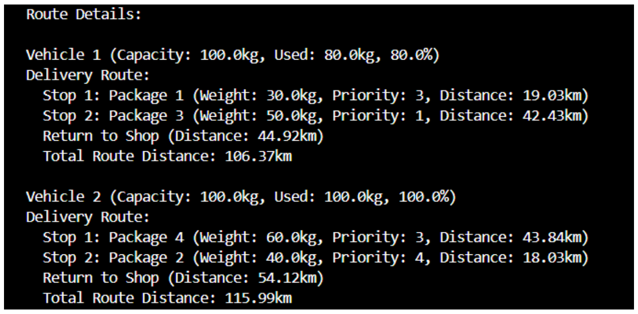
Test cases

Test Case 1: Basic Feasibility Test



input.txt

```
2
100
100
1 19 30 3
25 48 40 4
43 13 50 1
31 31 60 3
```



```
Route Details:

Vehicle 1 (Capacity: 100.0kg, Used: 80.0kg, 80.0%)
Delivery Route:
  Stop 1: Package 1 (Weight: 30.0kg, Priority: 3, Distance: 19.03km)
  Stop 2: Package 3 (Weight: 50.0kg, Priority: 1, Distance: 42.43km)
  Return to Shop (Distance: 44.92km)
  Total Route Distance: 106.37km

Vehicle 2 (Capacity: 100.0kg, Used: 100.0kg, 100.0%)
Delivery Route:
  Stop 1: Package 4 (Weight: 60.0kg, Priority: 3, Distance: 43.84km)
  Stop 2: Package 2 (Weight: 40.0kg, Priority: 4, Distance: 18.03km)
  Return to Shop (Distance: 54.12km)
  Total Route Distance: 115.99km
```

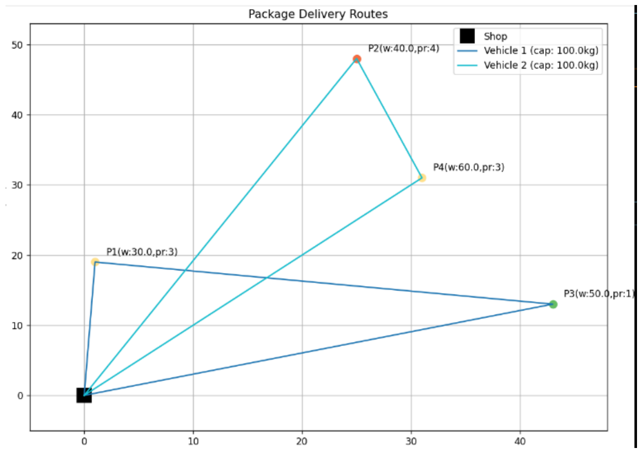
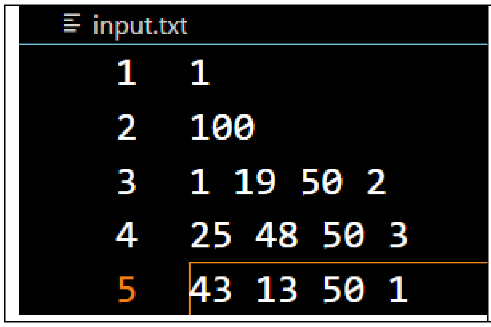


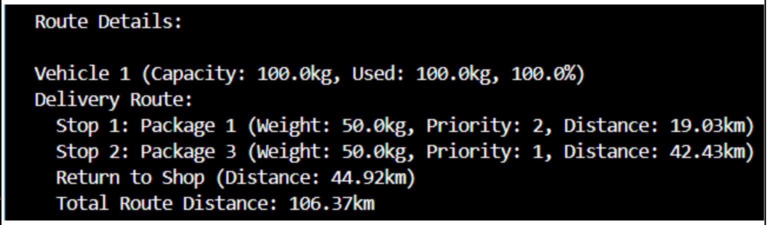
Figure 4 : Example of test case 1

Test Case 2: Priority Handling Test



input.txt

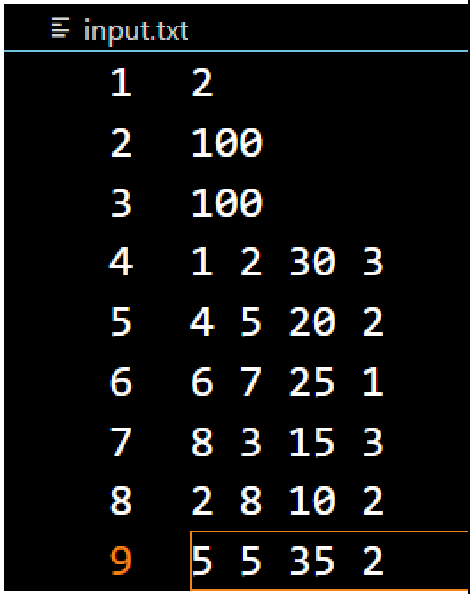
```
1
100
1 19 50 2
25 48 50 3
43 13 50 1
```



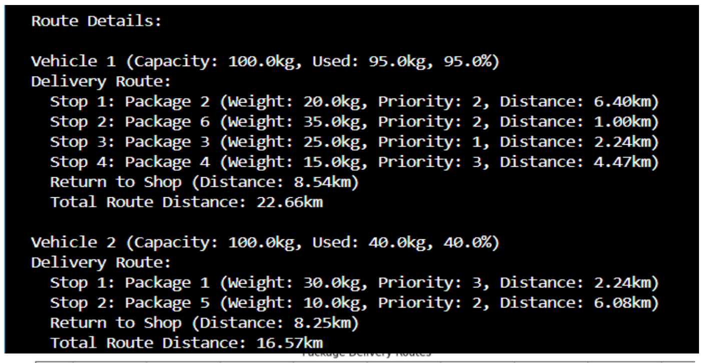
```
Route Details:

Vehicle 1 (Capacity: 100.0kg, Used: 100.0kg, 100.0%)
Delivery Route:
  Stop 1: Package 1 (Weight: 50.0kg, Priority: 2, Distance: 19.03km)
  Stop 2: Package 3 (Weight: 50.0kg, Priority: 1, Distance: 42.43km)
  Return to Shop (Distance: 44.92km)
  Total Route Distance: 106.37km
```

Figure 5 : Example of test case 2

Test Case 3: Distance Optimization Test



```
input.txt
2
100
100
1 2 30 3
4 5 20 2
6 7 25 1
8 3 15 3
2 8 10 2
5 5 35 2
```



```
Route Details:

Vehicle 1 (Capacity: 100.0kg, Used: 95.0kg, 95.0%)
Delivery Route:
  Stop 1: Package 2 (Weight: 20.0kg, Priority: 2, Distance: 6.40km)
  Stop 2: Package 6 (Weight: 35.0kg, Priority: 2, Distance: 1.00km)
  Stop 3: Package 3 (Weight: 25.0kg, Priority: 1, Distance: 2.24km)
  Stop 4: Package 4 (Weight: 15.0kg, Priority: 3, Distance: 4.47km)
  Return to Shop (Distance: 8.54km)
  Total Route Distance: 22.66km

Vehicle 2 (Capacity: 100.0kg, Used: 40.0kg, 40.0%)
Delivery Route:
  Stop 1: Package 1 (Weight: 30.0kg, Priority: 3, Distance: 2.24km)
  Stop 2: Package 5 (Weight: 10.0kg, Priority: 2, Distance: 6.08km)
  Return to Shop (Distance: 8.25km)
  Total Route Distance: 16.57km
```

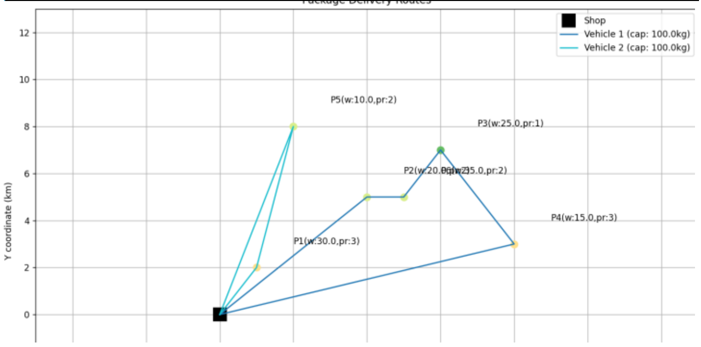
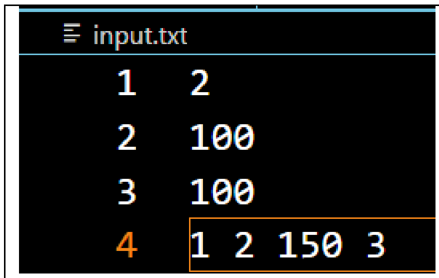
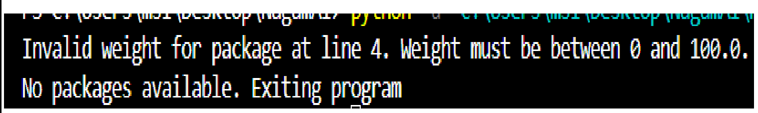


Figure 6 : Example of test case 3

Test Case 4: Edge Case - Overcapacity Package



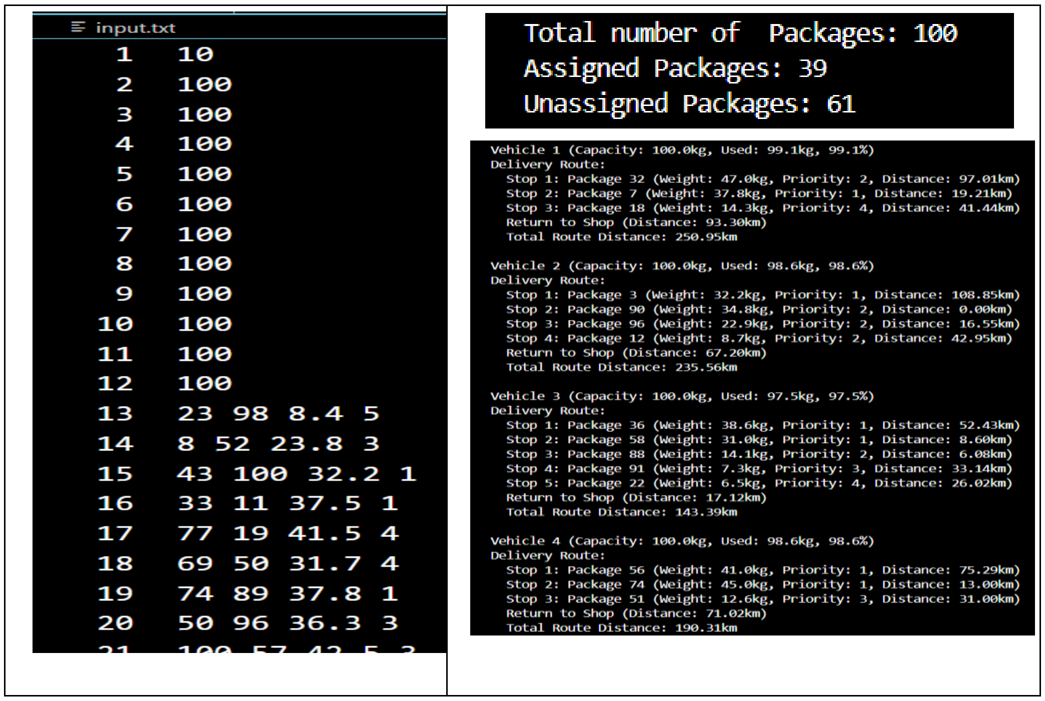
```
input.txt
1   2
2   100
3   100
4   1 2 150 3
```



```
Invalid weight for package at line 4. Weight must be between 0 and 100.0.
No packages available. Exiting program
```

Figure 7 : Example of test case 4

Test Case 6: Scalability Test



```
input.txt
10
100
100
100
100
100
100
100
100
100
100
100
23 98 8.4 5
8 52 23.8 3
43 100 32.2 1
33 11 37.5 1
77 19 41.5 4
69 50 31.7 4
74 89 37.8 1
50 96 36.3 3
```

```
Total number of  Packages: 100
Assigned Packages: 39
Unassigned Packages: 61
```

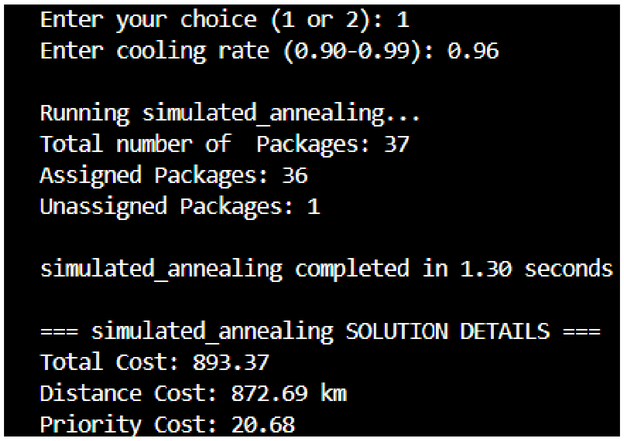
```
Vehicle 1 (Capacity: 100.0kg, Used: 99.1kg, 99.1%)
Delivery Route:
  Stop 1: Package 32 (Weight: 47.0kg, Priority: 2, Distance: 97.01km)
  Stop 2: Package 7 (Weight: 37.8kg, Priority: 1, Distance: 19.21km)
  Stop 3: Package 18 (Weight: 14.3kg, Priority: 4, Distance: 41.44km)
  Return to Shop (Distance: 93.30km)
  Total Route Distance: 250.95km

Vehicle 2 (Capacity: 100.0kg, Used: 98.6kg, 98.6%)
Delivery Route:
  Stop 1: Package 3 (Weight: 32.2kg, Priority: 1, Distance: 108.85km)
  Stop 2: Package 90 (Weight: 34.8kg, Priority: 2, Distance: 0.00km)
  Stop 3: Package 96 (Weight: 22.9kg, Priority: 2, Distance: 16.55km)
  Stop 4: Package 12 (Weight: 8.7kg, Priority: 2, Distance: 42.95km)
  Return to Shop (Distance: 67.20km)
  Total Route Distance: 235.56km

Vehicle 3 (Capacity: 100.0kg, Used: 97.5kg, 97.5%)
Delivery Route:
  Stop 1: Package 36 (Weight: 38.6kg, Priority: 1, Distance: 52.43km)
  Stop 2: Package 58 (Weight: 31.0kg, Priority: 1, Distance: 8.60km)
  Stop 3: Package 88 (Weight: 14.1kg, Priority: 2, Distance: 6.08km)
  Stop 4: Package 91 (Weight: 7.3kg, Priority: 3, Distance: 33.14km)
  Stop 5: Package 22 (Weight: 6.5kg, Priority: 4, Distance: 26.02km)
  Return to Shop (Distance: 17.12km)
  Total Route Distance: 143.39km

Vehicle 4 (Capacity: 100.0kg, Used: 98.6kg, 98.6%)
Delivery Route:
  Stop 1: Package 56 (Weight: 41.0kg, Priority: 1, Distance: 75.29km)
  Stop 2: Package 74 (Weight: 45.0kg, Priority: 1, Distance: 13.00km)
  Stop 3: Package 51 (Weight: 12.6kg, Priority: 3, Distance: 31.00km)
  Return to Shop (Distance: 71.02km)
  Total Route Distance: 190.31km
```

Figure 8 : Example of test case 6

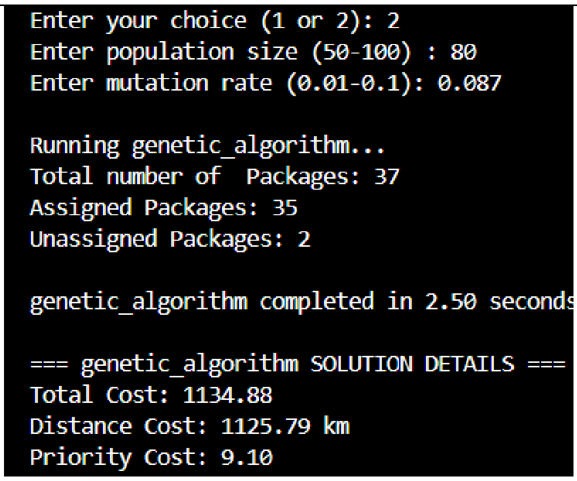
Test Case 5: Simulated Annealing vs. Genetic Algorithm Comparison

In terms of	Simulated Annealing (SA)	Genetic Algorithm (GA)
Parameter	Cooling Rate = 0.96	Population size = 80 Mutation = 0.087
Execution Time (s)	1.30 sec	2.5 sec
Total Cost	893.37 km	1134.88 km
Total Distance	872.69 km	1125.79 km
Priority Penalty	20.68 km	9.1 km
Total number of Packages	37	37
Packages Assigned	36	35

Table 1 : Simulated Annealing vs. Genetic Algorithm Comparison



```
Enter your choice (1 or 2): 1
Enter cooling rate (0.90-0.99): 0.96

Running simulated_annealing...
Total number of  Packages: 37
Assigned Packages: 36
Unassigned Packages: 1

simulated_annealing completed in 1.30 seconds

=== simulated_annealing SOLUTION DETAILS ===
Total Cost: 893.37
Distance Cost: 872.69 km
Priority Cost: 20.68
```



```
Enter your choice (1 or 2): 2
Enter population size (50-100) : 80
Enter mutation rate (0.01-0.1): 0.087

Running genetic_algorithm...
Total number of  Packages: 37
Assigned Packages: 35
Unassigned Packages: 2

genetic_algorithm completed in 2.50 seconds

=== genetic_algorithm SOLUTION DETAILS ===
Total Cost: 1134.88
Distance Cost: 1125.79 km
Priority Cost: 9.10
```

Figure 9 : Example of test case 5