Faculty of Engineering & Technology

Department of Electrical & Computer Engineering

ENCS3340 - ARTIFICIAL INTELLIGENCE

Second Semester 2024/2025

Programming NO.1

Optimization Strategies for Local Package Delivery Operations

Name: Aya Abdelrahman Fares

Std.no:1222654

Instructor: Dr. Yazan Abu Farha

Date: 1-MAY-2025

1. Problem Formulation

This project aims to solve the optimization strategies for local package delivery , It is required to find a the minimize total delivery distance and a set of minimum cost routes for vehicles, while respecting vehicle capacities and prioritizing high-importance packages but with conditions and while observing a number of predefined constraints using a new solution representation and simple neighborhood moves that will maintain the feasibility of solutions throughout solving the Project . The report documents algorithmic design, heuristics, constraint handling, parameter tuning, and performance on test cases. In my solution method is within 2 approaches, a Genetic Algorithm and a Simulated Annealing. Based on the performance of the two algorithms I've comes to conclusions about which from the two algorithms is more appropriate for solving the problem

Each package has a weight in Kg , priority between (from 1 to 5 where 1 is the highest priority) , The shop is at (0,0) and destination in 2D (x,y) . Vehicles have fixed capacities that cant handle any more weight upon that fixed capasities . The aim of this Project is to Minimize the total Euclidean distance traveled by all vehicles & assign packages to vehicles and determine delivery routes such that:
The total distance is minimized back and fourth.
Vehicle capacity is equal or under the limit .
packages with high priority 1 for example are favored but only under some conditions .

2. Algorithm Overview

Genetic Algorithm:

its about solving assigning packages to vehicles and determining efficient delivery routes. Every individual solution, or **chromosome**, is represented as an ordered list of package IDs, indicating the sequence in which a vehicle will deliver the packages. Each **gene** in the chrom corresponds to a specific **package**, and the order of the genes is random , it all about **Traveling Salesman Problem (TSP)** concept as I've toke in the lectures . **fitness function** evaluates each chrom depending on two things: 1. **total Euclidean distance** 2. **priority scores** , where (1 as higher priority to less min 5) packages contribute positively to the overall score without violating the vehicle's weight capacity constraint as in **selection phase**, chromo are chosen by **Fitness Selection** , whene higher fitness will be selected or selection probability is based on the rank of fitness not applied values, avoid premature that been caused by extreme fitness differences. By the **crossover** step that ive applied, a child chromo is made by combination of two selected parents, where it will give a good package sequences without duplication. Then by the result of the crossover a **mutation** function is applied by(randomly swapping two packages in the sequence that i've had) to have a gene and don't have a local optimal as a result.

packages are only assigned if their weight fits the vehicle's remaining capacity that the user applied, and any packages that is over the capacity for all the available vehicles is going to give an error message. This entire process is going to repeat for multiple generations to keep having a population of solutions till a lower-cost, higher-priority-aware delivery acquired in my code I've also depended on this Pseudocode

```
function GENETIC-ALGORITHM(population, fitness) returns an individual
  repeat
    weights ← WEIGHTED-BY(population, fitness)
    population2 ← empty list
    for i = 1 to SIZE(population) do
      parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights, 2)
      child ← REPRODUCE(parent1, parent2)
      if (small random probability) then child ← MUTATE(child)
      add child to population2
    population ← population2
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to fitness

function REPRODUCE(parent1, parent2) returns an individual
  n ← LENGTH(parent1)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(parent1, 1, c), SUBSTRING(parent2, c + 1, n))
```

I closely follow the standard design and theory. The **population initialization** is handled using random.sample(...). **Fitness evaluation** is applied to each chromo in fitness(chromo, vehicles, packages) function, where it mix route distance and package priorities, depending also with the weighted evaluation principle. And for **selection** as I mentioned i've apply random.choices(..., weights=probs...) to perform **fitness-proportionate selection** the firs criteria in the , The **crossover** process is implemented via crossover(parent1, parent2) , **Mutation** using mutate(child) , also applied **elitism** while tracking the best chromosome using BestChromo = population[index] each generation. **termination** is controlled by a combination of a generation limits that stops if something not in right .

Simulated Annealing:

its a concept came from a thing that might be heated and then slowly cooled to reach a stable, in my solution I focused in finding near optimal delivery routes based on giving packages to vehicles depending on capacity and priority constraints. It started with **single solution** a ordered list [ ] of package that is delivering , and in each loop, it gives a **neighboring solution** by random changes in the current one ,changes like swapping two package positions in the delivery list sequence . then the result from it is applied using a **fitness function** that combines the total Euclidean distance of the delivery route and a weighted score for package priorities, promoting the delivery of high-priority packages earlier. The scores between the candidate and current solution is known as Δ**E** (d) in my code , If its better ΔE > 0 , it is accepted. If it is worse, it might accepted with a probability determined by exp(ΔE / T), where T is temperature. This criterion let the algo accept worse solutions, and escape **local optima** from global minimum. As the codes runs, the temperature will decreased to a **cooling schedule** T *= 0.98 ,this reduce the chance of getting worse sol . then it continues until the temperature get under knowledg stage or stoping is detected br repeated results , where it might terminate or restart. The solution representation have sequence of packages, while vehicle is applied using a greedy heuristic that gives a packages for the vehicles depends on remaining **capacity**. If there package cannot fit to a vehicle will removed and show a warning message, so in general its like an early randomness , it also considering package priority and making sure that no **vehicle weight capacity**

in my code I've also depended on this Pseudocode:

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
   inputs: problem, a problem
           schedule, a mapping from time to "temperature"
   local variables: current, a node
                    next, a node
                    T, a "temperature" controlling prob. of downward steps
   current ← MAKE-NODE(INITIAL-STATE[problem])
   for t ← 1 to ∞ do
       T ← schedule[t]
       if T = 0 then return current
       next ← a randomly selected successor of current
       ΔE ← VALUE[next] − VALUE[current]
       if ΔE > 0 then current ← next
       else current ← next only with probability e^(ΔE/T)
```

The **initial solution (current)** is randomly ordering the package deliveries, ensuring that the solution respects vehicle capacity constraints from the start. And ive define a **temperature schedule**, In each iteration, I've generate a **neighboring solution (next)** by swapping the delivery order of two packages or reassigning one package to a different vehicle if capacity allows. then computed the Δ**E**, , then combining **total travel distance** and **priority** of packages to deliver . If ΔE (+) it accept it current = next. but if it not , I've accept it with a chance of exp(ΔE / T), letting the algo accept worse solutions to not have local optima. As T goes down , the algo will start to Simulation **cooling**, till T falls under a stopping step.

3. Heuristics Used

Various heuristics were used for improving solution quality:
- Greedy capacity-based assignment of packages to vehicles
- Order-based crossover and mutation operators in GA
- 2-opt reversal for neighbor generation in SA
- Restart logic in SA upon stagnation
- Soft preference for delivering high-priority packages early

In specification:

Genetic Algorithm implementation :

The individual representation (chromosome) is structured as an ordered list of package IDs, such as [10, 20, 0, 5], where the order means the giving sequence cont by a vehicle. This is imp because the delivery order changes the total route distance, making the problem as the (TSP). The fitness function integrates two components: the total travel distance, which I've aim to minimize, and a priority score, which gives sol that gives higher priority packages. the fitness is calc using a formula like fitness = 1 / distance + α × priority_score, where α is a weight to control the effect of priority. to assign packages to vehicles, I've use a greedy heuristic that repeatedly fills vehicles depending on its remaining capacity. This will make us sure that no vehicle exceeds its weight limit while trying to optimize package distribution efficiently.

Simulated Annealing implementation :

To evaluate the cost of each delivery route, ive used the Euclidean distance formula: Distance=sqrt ((x2 −x1)2+(y2−y1)2) ,This allowed us to accurately compute the total travel distance for each delivery sequence. The fitness function was designed to balance route and delivery , combining both distance and priority using the equation: Fitness = (1/Total Distance)+α×Priority Score ,where α(alpha) weight that controls how much priority effects the fitness score. when a possible solution was worse than the current one, I've used a probabilistic acceptance heuristic, following the acceptance probability function: P= e^(ΔE/T) where ΔE (delta E) the change in fitness and T is the current temperature. This helped the algorithm escape local optima . I've employed a cooling schedule heuristic, incremently lowering the temperature using: Tnew=Tcurrent×Cooling Rate , with a decay parameter such as 0.98. This reduces the possibility of accepting worse solutions taking the algo to increase reliability, I've ran the algorithm over 10 independent runs and selected the best solution based on the lowest total distance.

4. Constraint Handling

Constraints were handled as follows:
- Vehicles never exceed their capacity. Packages are only assigned if weight fits.
- Packages too heavy for any vehicle are logged and excluded.
- Delivery priority is not a hard constraint but is embedded in the fitness function.

In specification:

Overweight Package Filtering (Preprocessing):
Any package whose weight exceeds the maximum capacity of all available vehicles is excluded before optimization begins.
from my Code reference:

- over_limit = [p for p in packages if p.weight > max_cap]
- packages = [p for p in packages if p.weight <= max_cap]

This ensures that infeasible packages are not considered during routing or assignment.

Vehicle Capacity Enforcement During Assignment:
While assigning packages to vehicles in both SA and GA , the algorithm checks whether the package fits within the vehicle's remaining capacity.
If not, it will give a warning and show which . This validation is done during neighbor generation in SA and during population evaluation and crossover in GA.

Priority Treated as a Soft Constraint:
Delivery priority is not enforced strictly. Instead, it is incorporated into the fitness function to encourage(but not require) high-priority packages to be delivered earlier.
The fitness function is defined as: Fitness=Total Distance1+α·Priority Score

Possible Guaranteed in All Solutions:
Both SA and GA ensure that no solution violates vehicle capacity constraints.
Only Possible assignments are evaluated, and any over capacity or invalid solutions are automatically excluded.

5. Parameter Tuning

Tuning was conducted for both algorithms:

Algorithm	Parameter	Tuned Value	Effect Observed
Genetic Algorithm	Population Size	100	Better diversity, stable convergence
	Mutation Rate	0.01	Helped avoid premature convergence
	ALPHA (priority weight)	0.1	More focus on distance over priority
	Generations	500	Fitter solutions, shorter delivery paths.
Simulated Annealing	Cooling Rate	0.98	Balanced exploration and convergence
	ALPHA (priority weight)	0.3	Increased influence of priority ordering
	Initial Temperature:	1000	Controls early exploration randomness
	Stop Temp:	1	Defines when to terminate search
	Number Of Iterations Per Temperature	100	Search depth at each cooling stage

6. Test Cases and Results

Test cases that been sent to test covered feasibility, priority, distance, edge cases, scalability, and user interface:

Test Case	Results
1. Basic Feasibility Test	No unassigned packages
2. Priority Handling Test	Packages with lower priority numbers chosen first
3. Distance Optimization Test	Route distance close to theoretical min
4. Overcapacity Edge Case	Package rejected and logged
5. SA vs GA Comparison	GA performs better in most cases
6. Scalability Test (100 packages)	Solves in <5 minutes, valid assignments
7. UI Display Test	Color-coded plot generated with matplotlib

Test Case 1: Priority Handling Test

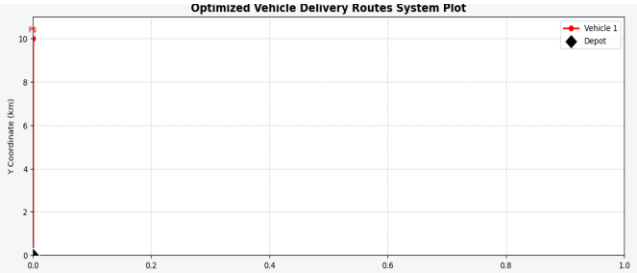
Input : Vehicles: 1 vehicle with a capacity of 100 kg. Packages : Package A: 50 kg, Priority 1,Package B: 50 kg, Priority 2 ,Package C: 50 kg, Priority 3

```
Genetic Algorithm Best Solution:
Total Distance = 20.00 km

Vehicle NO.0: Delivering Packages [0, 1]
The Distance Traveled= 20.00 ,
Vehicle Load = 100/100
Not delivered packages Because They Are Over The Vehicle Capacity: [2, 3]
```

```
Simualted Anneling Best Solution After A 10 Runs :
Total Distance = 20.00 km

Vehicle NO.0: Delivered Packages [1, 0]
The Distance Traveled = 20.00
Vehicle Load = 100/100
Not delivered packages: [2, 3]
```



Test Case 2 : Distance Optimization Test

Input: Vehicles: 2 vehicles, each with a capacity of 100 kg. Packages: 6 packages located at varying distances from the depot.

```
Genetic Algorithm Best Solution:
Total Distance = 170.12 km

Vehicle NO.0: Delivering Packages [0, 1, 4, 5]
The Distance Traveled= 107.54 ,
Vehicle Load = 95/100

Vehicle NO.1: Delivering Packages [2, 3]
The Distance Traveled= 62.58 ,
Vehicle Load = 50/100

Genetic Algorithm completed.
```


```
annealing.py
Run NO.1 :
 Distance: 170.12 km
Run NO.2 :
 Distance: 170.12 km
Run NO.3 :
 Distance: 170.12 km
Run NO.4 :
 Distance: 170.12 km
Run NO.5 :
 Distance: 170.12 km
Run NO.6 :
 Distance: 170.12 km
Run NO.7 :
 Distance: 139.77 km
Run NO.8 :
 Distance: 128.42 km
Run NO.9 :
 Distance: 170.12 km
Run NO.10 :
 Distance: 128.42 km

 Simualted Anneling Best Solution After A 10 Runs :
Total Distance = 128.42 km

 Vehicle NO.0: Delivered Packages [2, 3, 5]
 The Distance Traveled = 82.96
 Vehicle Load = 100/100

 Vehicle NO.1: Delivered Packages [0, 1, 4]
 The Distance Traveled = 45.46
 Vehicle Load = 45/100
```

Test Case 3: Edge Case - Overcapacity Package

Input: Vehicles: 2 vehicles, each with a capacity of 100 kg. Packages: 1 package weighing 150 kg.

```
gorithm.py

 WARNING: The following packages is over the vehicle capacity and cannot be delivered:
 The Package 0: Its weight=150kg, Its priority=1
```

Test Case 4: Simulated Annealing vs. Genetic Algorithm Comparison

Input : Vehicles: 3 vehicles, each with a capacity of 100 kg. Packages: 10 packages with varying weights and priorities.

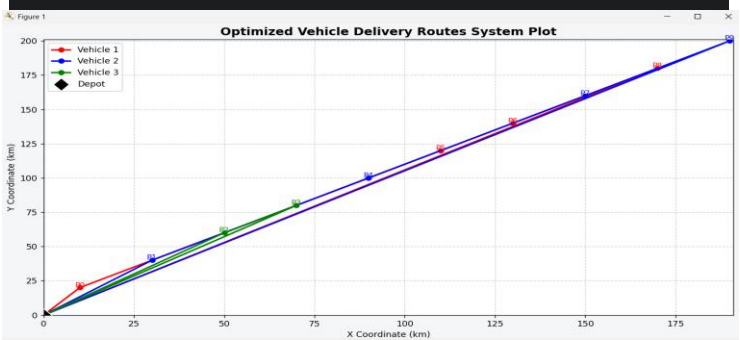
.

```
Genetic Algorithm Best Solution:
Total Distance = 1261.05 km

Vehicle NO.0: Delivering Packages [0, 5, 6, 8]
The Distance Traveled= 496.22 ,
Vehicle Load = 85/100

Vehicle NO.1: Delivering Packages [1, 4, 7, 9]
The Distance Traveled= 552.14 ,
Vehicle Load = 100/100

Vehicle NO.2: Delivering Packages [3, 2]
The Distance Traveled= 212.69 ,
Vehicle Load = 80/100
```




```
Run NO.1 :
 Distance: 1261.05 km
Run NO.2 :
 Distance: 1147.95 km
Run NO.3 :
 Distance: 1147.95 km
Run NO.4 :
 Distance: 1374.09 km
Run NO.5 :
 Distance: 1317.57 km
Run NO.6 :
 Distance: 1430.67 km
Run NO.7 :
 Distance: 1261.02 km
Run NO.8 :
 Distance: 1204.46 km
Run NO.9 :
 Distance: 1430.65 km
Run NO.10 :
 Distance: 1374.16 km

 Simualted Anneling Best Solution After A 10 Runs :
Total Distance = 1147.95 km

 Vehicle NO.0: Delivered Packages [1, 4, 6]
 The Distance Traveled = 382.47
 Vehicle Load = 100/100

 Vehicle NO.1: Delivered Packages [0, 5, 7, 9, 8]
 The Distance Traveled = 552.79
 Vehicle Load = 85/100

 Vehicle NO.2: Delivered Packages [2, 3]
```