Faculty of Engineering and Technology

Department of Electrical and Computer Engineering

ENCS3340, Artificial Intelligence

Course project

Prepared by (Name,ID):

Ro’a Nafi , 1201959.

Dunia Al’amal Hamada, 1201001.

Instructor: Dr. Yazan Abu Farha**.**

Date: 11 | Sept | 2023.

Contents

Table of figures III
1.Problem Formulation IV
  1.1.State Representation IV
  1.2.Initial State IV
  1.3.Goal State IV
  1.4.Actions IV
  1.5.Constraints IV
2.Genetic Algorithm V
  2.1.Initialize Population: V
  2.2.Selection of Parents: VI
  2.3.Crossover: VI
  2.4.Mutation: VII
3.CSP Algorithm VII
4.Test data and Output IX
  4.1.Test Data: IX
  4.2.Genetic algorithm solution's IX
  4.3.CSP algorithm solution's X
  4.4. Genetic and CSP algorithm solution's XI

Table of figures

Figure 1 initialize V
Figure 2 parents........ VI
Figure 3 crossover VI
Figure 4 mutation........ VII
Figure 5 CSP function........ VII
Figure 6 test data IX
Figure 7 GA interface........ IX
Figure 8 GA console output IX
Figure 9 CSP interface X
Figure 10 SCP console output........ X
Figure 11 GA and SCP interface XI
Figure 12 GA and SCP console output XI

1.Problem Formulation

1.1.State Representation

Package: Each package is represented by a dictionary with the following keys:

- Id: A unique ID for each Package.
- destination: The coordinates of the package's destination (x,y).
- weight: Represented the weight of the package in kilogram.

Vehicle: Each vehicle is represented by a dictionary with the following keys:

- Id: A unique ID for each vehicle.
- packages: A list of dictionaries, where each dictionary is a package that the vehicle is carrying.
- remaining_capacity: Represented how much more weight the vehicle can carry.

State:

The State is a list possible solution to the problem. a solution is a list of vehicles with their assigned packages.

1.2.Initial State

All vehicles have an empty list of assigned packages and an empty travel route. All packages are unassigned.

1.3.Goal State

All packages are assigned to vehicles without exceeding any vehicle's weight capacity, and the total travel distance for all vehicles is minimized.

1.4.Actions

For any given state, we can perform the following actions:

- Assign a package to the vehicle within the permitted capacity of the vehicle
- Change the order of delivery for the packages already assigned to a vehicle
- Swap packages between two vehicles

1.5.Constraints

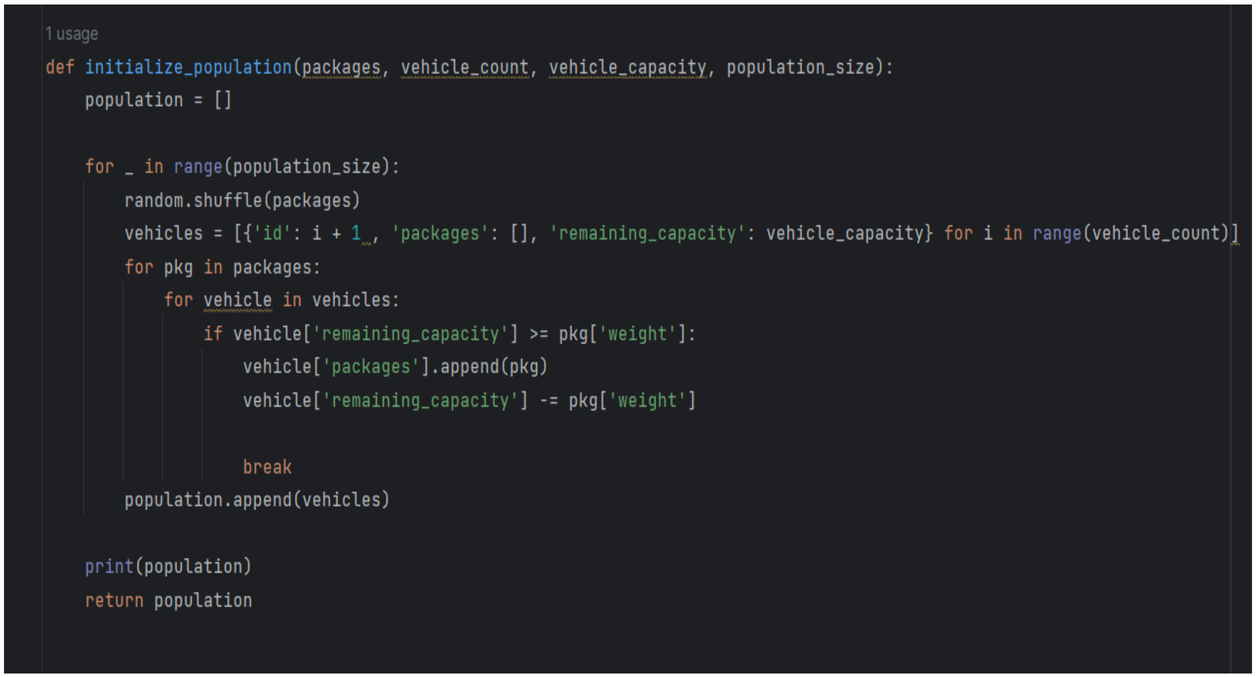
- Total weight of packages in a vehicle must not exceed its capacity.
- A package can only be assigned to one vehicle.
- Each package must be delivered.

2.Genetic Algorithm

Genetic Algorithm's components:

- Initialize Population
- Select Parents
- Crossover
- Mutation

2.1.Initialize Population:



```
1 usage
def initialize_population(packages, vehicle_count, vehicle_capacity, population_size):
    population = []

    for _ in range(population_size):
        random.shuffle(packages)
        vehicles = [{'id': i + 1 , 'packages': [], 'remaining_capacity': vehicle_capacity} for i in range(vehicle_count)]
        for pkg in packages:
            for vehicle in vehicles:
                if vehicle['remaining_capacity'] >= pkg['weight']:
                    vehicle['packages'].append(pkg)
                    vehicle['remaining_capacity'] -= pkg['weight']

                    break
        population.append(vehicles)

    print(population)
    return population
```

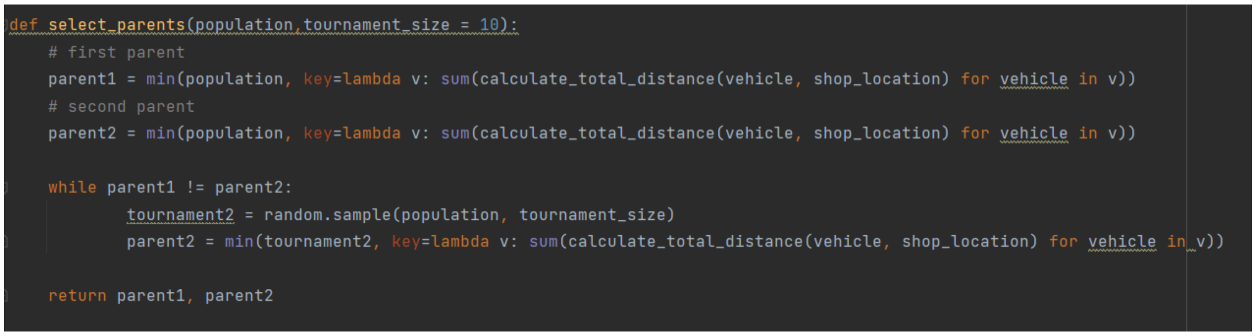
Figure 1 initialize

This function generates an initial population of potential solutions.

The population was represented by a list of 20 possible solutions, each solution is a list of vehicles where these vehicles are represented by dictionaries as we mentioned in Problem Formulation section.

In our optimization process using Genetic Algorithm, we used a maximum of 100 iterations. This limit was set to control the algorithm's runtime and ensure it stops after a specific number of rounds.

2.2.Selection of Parents:

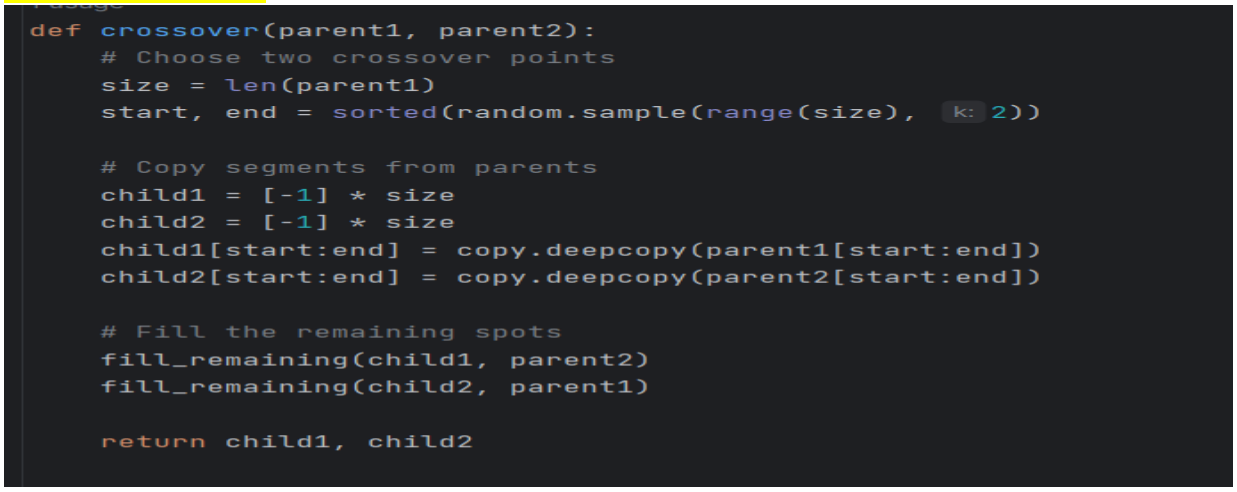


```python
def select_parents(population,tournament_size = 10):
    # first parent
    parent1 = min(population, key=lambda v: sum(calculate_total_distance(vehicle, shop_location) for vehicle in v))
    # second parent
    parent2 = min(population, key=lambda v: sum(calculate_total_distance(vehicle, shop_location) for vehicle in v))

    while parent1 != parent2:
            tournament2 = random.sample(population, tournament_size)
            parent2 = min(tournament2, key=lambda v: sum(calculate_total_distance(vehicle, shop_location) for vehicle in v))

    return parent1, parent2
```

Figure 2 parents

This function implements a tournament selection mechanism to choose two parents from the current population based on their fitness, which is calculated as the total travelled distance by the vehicles in their routes.

2.3.Crossover:

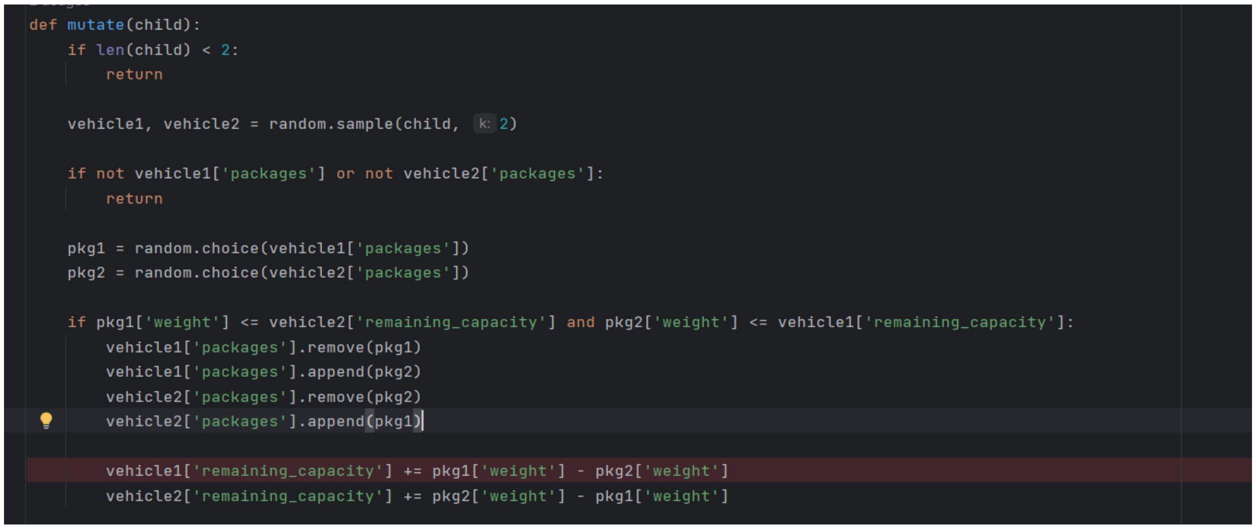


```python
def crossover(parent1, parent2):
    # Choose two crossover points
    size = len(parent1)
    start, end = sorted(random.sample(range(size), k: 2))

    # Copy segments from parents
    child1 = [-1] * size
    child2 = [-1] * size
    child1[start:end] = copy.deepcopy(parent1[start:end])
    child2[start:end] = copy.deepcopy(parent2[start:end])

    # Fill the remaining spots
    fill_remaining(child1, parent2)
    fill_remaining(child2, parent1)

    return child1, child2
```

Figure 3 crossover

The crossover function performs one-point crossover on two parent solutions to create two child solutions. It selects two crossover points, copies segments of routes from the parents to the children, and fills in the remaining parts while preserving package assignments and vehicle capacity constraints.

2.4.Mutation:



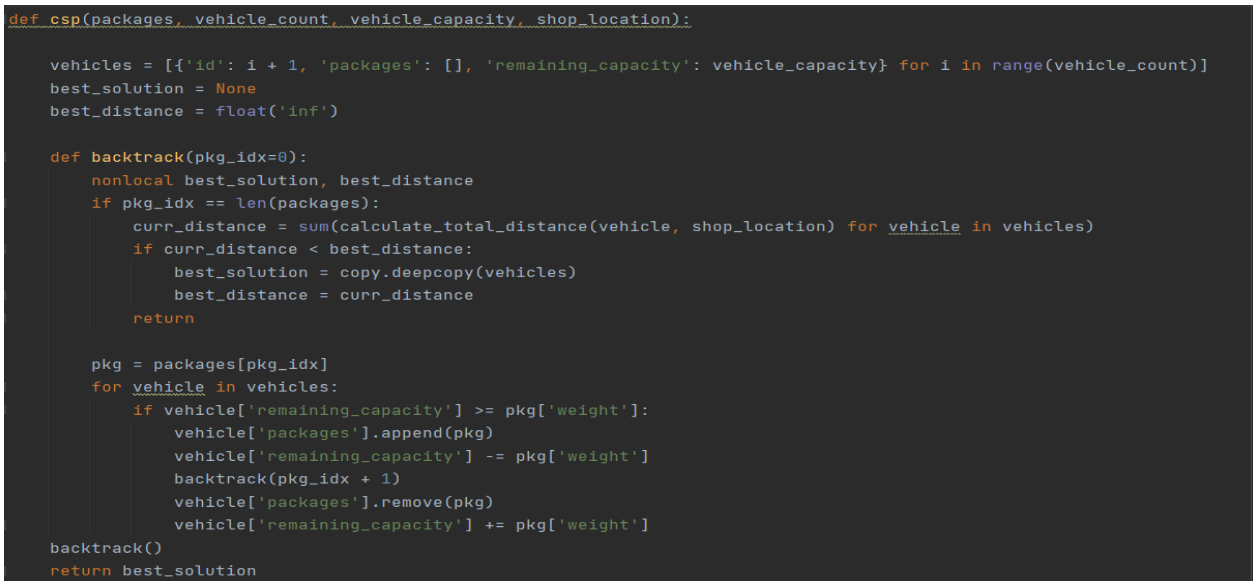
```
def mutate(child):
    if len(child) < 2:
        return

    vehicle1, vehicle2 = random.sample(child,  k: 2)

    if not vehicle1['packages'] or not vehicle2['packages']:
        return

    pkg1 = random.choice(vehicle1['packages'])
    pkg2 = random.choice(vehicle2['packages'])

    if pkg1['weight'] <= vehicle2['remaining_capacity'] and pkg2['weight'] <= vehicle1['remaining_capacity']:
        vehicle1['packages'].remove(pkg1)
        vehicle1['packages'].append(pkg2)
        vehicle2['packages'].remove(pkg2)
        vehicle2['packages'].append(pkg1)

        vehicle1['remaining_capacity'] += pkg1['weight'] - pkg2['weight']
        vehicle2['remaining_capacity'] += pkg2['weight'] - pkg1['weight']
```

Figure 4 mutation

The mutation function introduces small changes into a solution by swapping a package between two randomly selected vehicles. This mutation helps introduce diversity into the population and can lead to the discovery of better solutions

3.CSP Algorithm

csp function employs the principles of a backtracking algorithm for CSPs to find an assignment of packages to vehicles that minimizes the total travelled distance while respecting vehicle capacity constraints. It uses the variables (vehicles), their domains (remaining capacity), and constraints (vehicle capacity) to iteratively explore and evaluate possible solutions until a valid and optimized assignment is found or until all possibilities have been exhausted.



```
def csp(packages, vehicle_count, vehicle_capacity, shop_location):

    vehicles = [{'id': i + 1, 'packages': [], 'remaining_capacity': vehicle_capacity} for i in range(vehicle_count)]
    best_solution = None
    best_distance = float('inf')

    def backtrack(pkg_idx=0):
        nonlocal best_solution, best_distance
        if pkg_idx == len(packages):
            curr_distance = sum(calculate_total_distance(vehicle, shop_location) for vehicle in vehicles)
            if curr_distance < best_distance:
                best_solution = copy.deepcopy(vehicles)
                best_distance = curr_distance
            return

        pkg = packages[pkg_idx]
        for vehicle in vehicles:
            if vehicle['remaining_capacity'] >= pkg['weight']:
                vehicle['packages'].append(pkg)
                vehicle['remaining_capacity'] -= pkg['weight']
                backtrack(pkg_idx + 1)
                vehicle['packages'].remove(pkg)
                vehicle['remaining_capacity'] += pkg['weight']
    backtrack()
    return best_solution
```

Figure 5 CSP function

Variable Selection:

the variables correspond to the delivery vehicles. We have a list of vehicles, each with an ID and remaining capacity. The variable selection is implicit, as you loop through the vehicles in ‘for vehicle in vehicles:’ loop.

Value Selection:

For each vehicle (variable), consider assigning a package to it. The value selection occurs when you iterate over the packages and try assigning each package to the current vehicle if its capacity allows.

Assign the Value:

When a package is assigned to a vehicle, it's appended to the vehicle['packages'] list, and the remaining capacity of the vehicle is updated accordingly.

Check Constraints:

Constraints are checked in the form of vehicle capacity constraints. Before assigning a package to a vehicle, the code checks if the vehicle's remaining capacity is sufficient to accommodate the package's weight. If this constraint is violated, the assignment is not made.

Repeat or Backtrack:

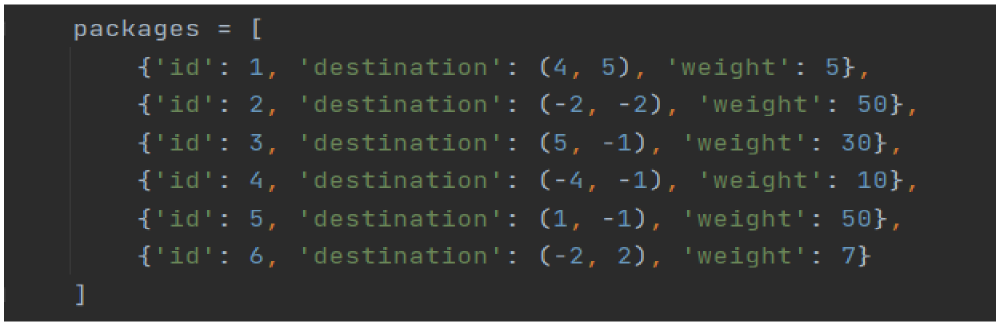
The backtracking aspect of the algorithm is evident in the recursive backtrack function. It explores possible assignments of packages to vehicles, and if a constraint is violated (i.e., a package cannot be assigned to any available vehicle without exceeding its capacity), it backtracks to the previous package assignment and tries a different vehicle.

Termination:

The termination condition for the backtrack function is when pkg_idx becomes equal to the total number of packages. At this point, it means all packages have been assigned, and the function calculates the total distance travelled by all vehicles to determine if it's the best solution found so far.

4.Test data and Output

4.1.Test Data:



```
packages = [
    {'id': 1, 'destination': (4, 5), 'weight': 5},
    {'id': 2, 'destination': (-2, -2), 'weight': 50},
    {'id': 3, 'destination': (5, -1), 'weight': 30},
    {'id': 4, 'destination': (-4, -1), 'weight': 10},
    {'id': 5, 'destination': (1, -1), 'weight': 50},
    {'id': 6, 'destination': (-2, 2), 'weight': 7}
]
```

Figure 6 test data

4.2.Genetic algorithm solution's

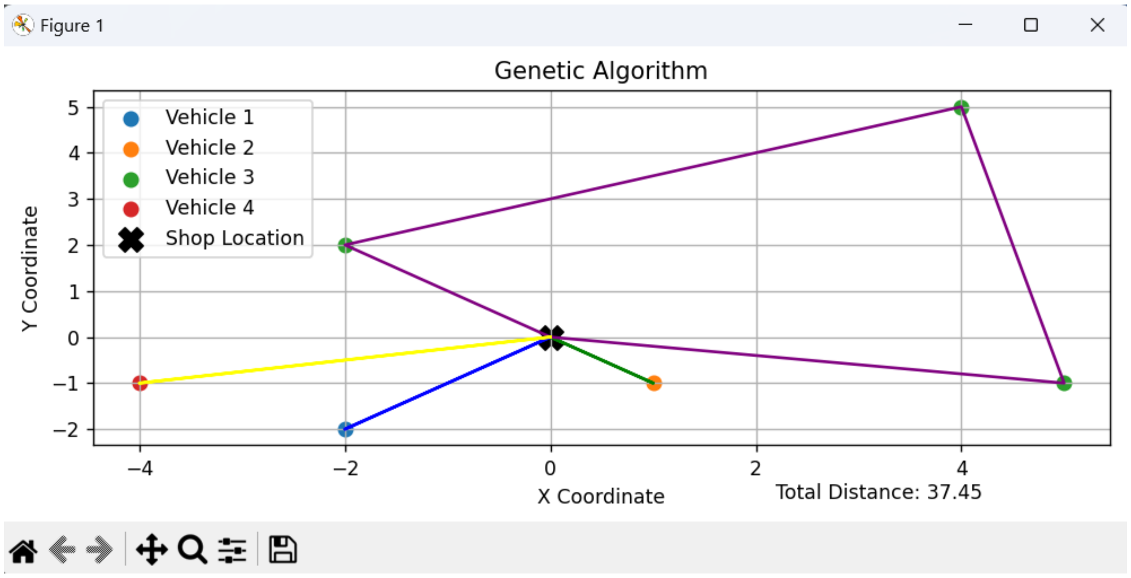
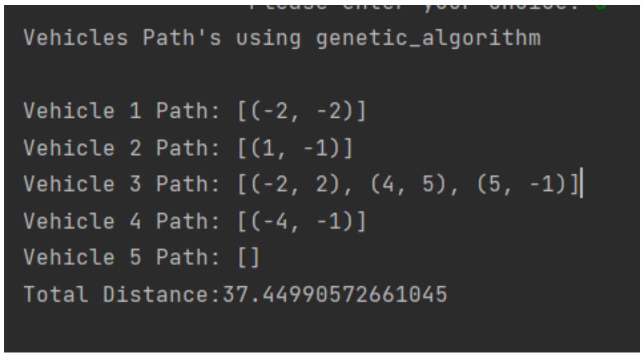


Figure 7 GA interface



```
Vehicles Path's using genetic_algorithm

Vehicle 1 Path: [(-2, -2)]
Vehicle 2 Path: [(1, -1)]
Vehicle 3 Path: [(-2, 2), (4, 5), (5, -1)]
Vehicle 4 Path: [(-4, -1)]
Vehicle 5 Path: []
Total Distance:37.44990572661045
```

Figure 8 GA console output

4.3.CSP algorithm solution's

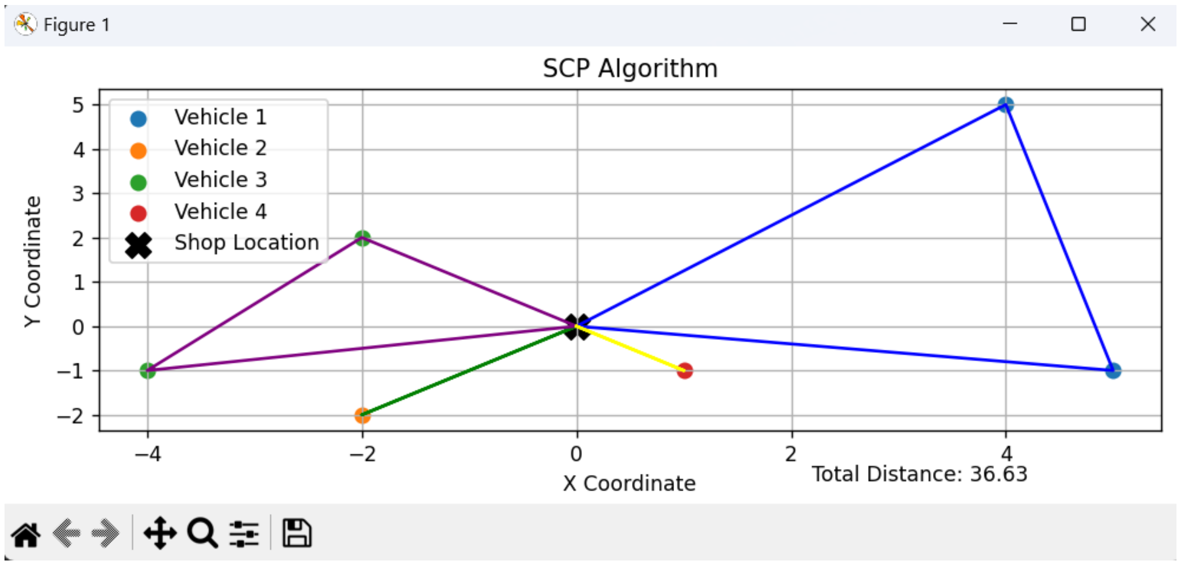
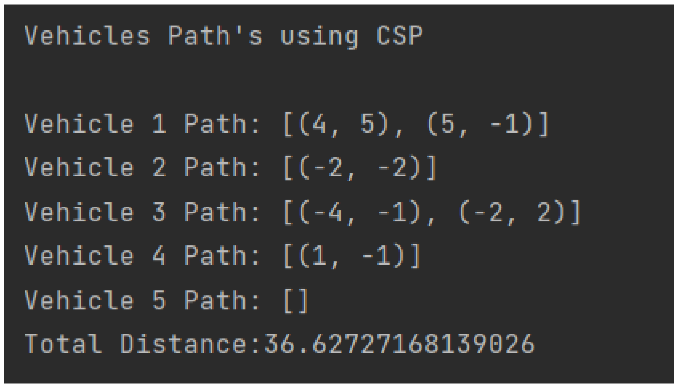


Figure 9 CSP interface



```
Vehicles Path's using CSP

Vehicle 1 Path: [(4, 5), (5, -1)]
Vehicle 2 Path: [(-2, -2)]
Vehicle 3 Path: [(-4, -1), (-2, 2)]
Vehicle 4 Path: [(1, -1)]
Vehicle 5 Path: []
Total Distance:36.62727168139026
```

Figure 10 SCP console output

4.4. Genetic and CSP algorithm solution’s

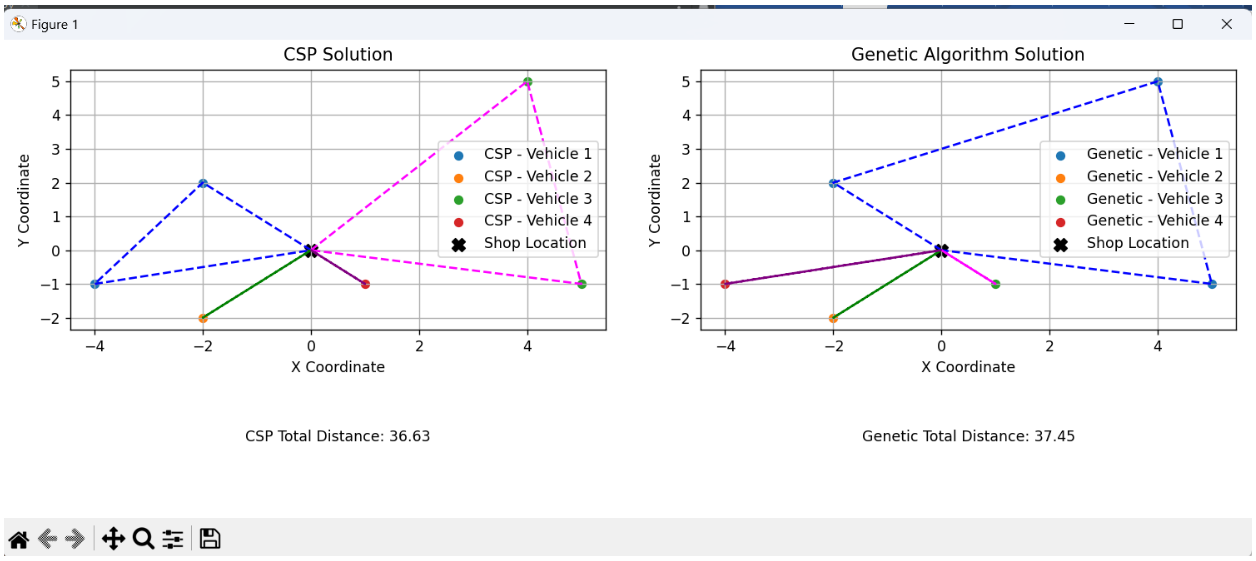
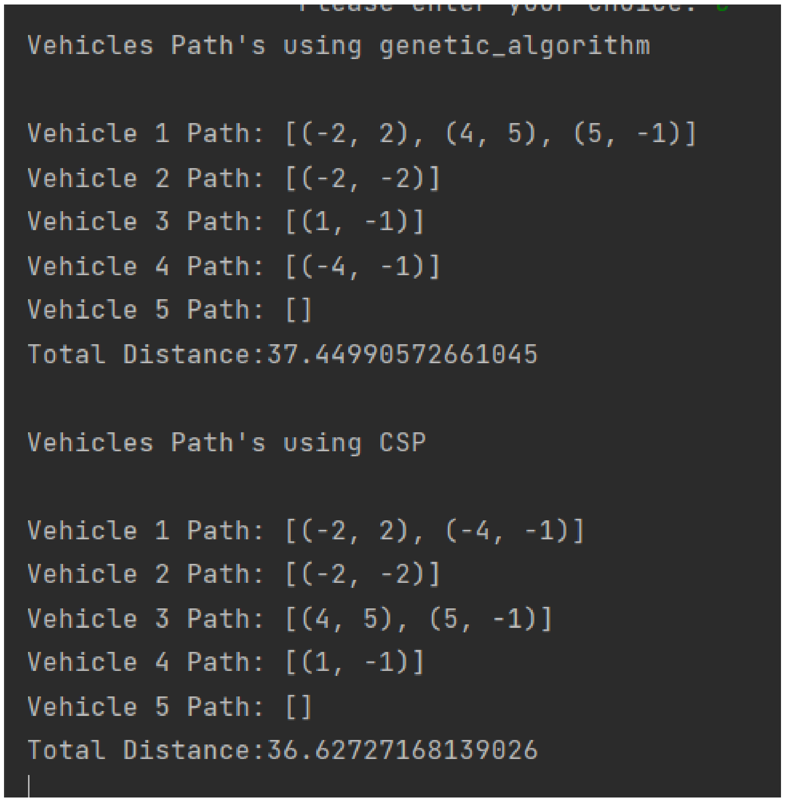


Figure 11 GA and SCP interface



```
Please enter your choice:
Vehicles Path's using genetic_algorithm

Vehicle 1 Path: [(-2, 2), (4, 5), (5, -1)]
Vehicle 2 Path: [(-2, -2)]
Vehicle 3 Path: [(1, -1)]
Vehicle 4 Path: [(-4, -1)]
Vehicle 5 Path: []
Total Distance:37.44990572661045

Vehicles Path's using CSP

Vehicle 1 Path: [(-2, 2), (-4, -1)]
Vehicle 2 Path: [(-2, -2)]
Vehicle 3 Path: [(4, 5), (5, -1)]
Vehicle 4 Path: [(1, -1)]
Vehicle 5 Path: []
Total Distance:36.62727168139026
```

Figure 12 GA and SCP console output