Athens University of Economics & Business
MSc in Business Analytics
Large Scale Optimization

Vehicle Routing Problem

Competitive Assignment

Teacher: Emmanouil Zachariadis

Students:
Maria Zafeiropoulou: P2822113
Marianna Konstantopoulou: P2822122
Spyridon Despotis: P2822111
Thanasis Papailiou: P2822128

2023

1. The problem

The problem is to transport relief supplies to 200 geographically dispersed locations after a natural disaster. Each location has a predetermined product demand and 26 homogeneous vehicles with a maximum carrying capacity of 3tn are used. The vehicles start from a central point, visit some of the 200 locations, and all routes start simultaneously at t=0. Every location is served once by a single vehicle, vehicles have a constant speed of 40km/hr and an unloading time of 15 minutes is necessary at each location. The goal is to minimize the service time completion of the last customer to be served.

2. The Deliverables

The final deliverables are the following:

- Main.py
- Solver.py
- SolutionDrawer.py
- VRP_Model.py
- Final_solution.txt

3. The Methodology

The following methodology was implemented in Python Programming Language. Firstly, we integrated the given code in the **VRP_Model** file. This code defines a class called Model, which is used to construct a mathematical model for the transportation problem. The Model class has several variables such as nodes, gpoints, tmatrix, distmatrix, and capacity, which are used to store data related to the problem. The time matrix is calculated based on the distance between the nodes and the vehicles' constant speed of 40km/hr, and an unloading time of 15 minutes is added. When an instance of the Model class is created, the **init** method is called, which initializes these variables to empty lists or -1. The BuildModel method is used to generate problem data by setting a random seed, creating a "depot" node with the ID 0, and generating a specified number of "gpoints" (customers), each with random x and y coordinates, a random demand between 100 and 399, and a fixed unloading time of 0.25. The BuildModel method then creates two matrices, tmatrix, and distmatrix, with the same number of rows and columns as the number of nodes and calculates the time and distance between each pair of nodes, storing them in the corresponding matrix. The class Node is also defined and has several attributes such as x and y coordinates, ID, demand and a flag indicating if it has been routed or not. The class Route is also defined and has several attributes such as the sequence of nodes, cost, capacity, and load.

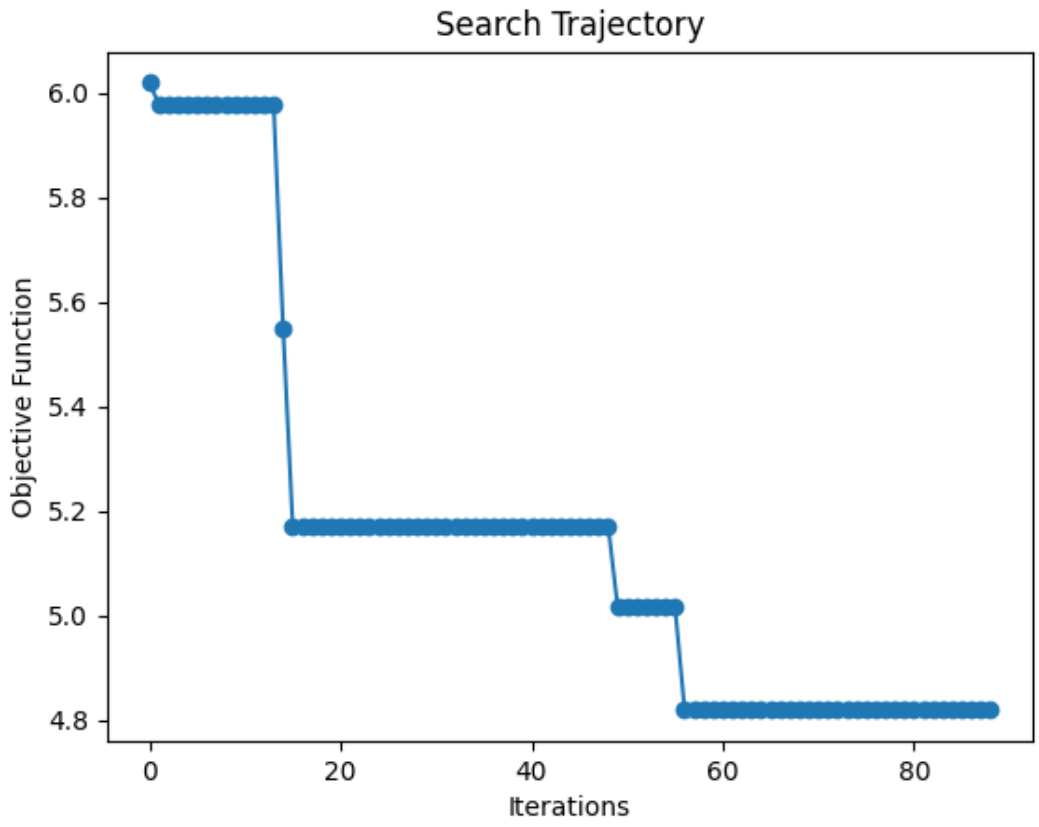
The **Solver** file defines several classes to solve a problem of routing a single vehicle to visit customers in the shortest distance. The classes include Solution, which represents a solution and includes the total distance traveled and a list of routes for each vehicle, RelocationMove, which represents a possible move of moving a customer from one route to another, CustomerInsertion and CustomerInsertionAllPositions for representing the insertion of a customer in a route, and the Solver

class which uses Minimum Insertions to start with a feasible solution, then applies local search to improve it and finally reports the final solution.

The local search algorithm is a method used to optimize the solution of a problem. It takes an initial solution as input and tries to improve it by making changes to the routes. Specifically, it uses the FindBestRelocationMove function to find the best possible move by evaluating the total cost of moving a point from one route to another or within the same route. If this move results in a lower cost and does not increase the value of the objective function, it is temporarily saved using the StoreBestRelocationMove function.

The code uses a local search algorithm to improve a given solution to the problem by making small adjustments to the routes and points. It does this by repeatedly finding the best possible change to make, as determined by the "FindBestRelocationMove" function, which calculates the effect of moving a point from one route to another or within the same route. If the change results in a reduction of total cost, the move is stored temporarily. Once the optimal move is found, the "ApplyRelocationMove" function is used to implement the change in the solution.

The final solution, as represented by the "Search Trajectory" diagram, shows that after 87 iterations and 26 routes, the longest route was reduced from 6 hours and 10.8 minutes to 4 hours and 48 minutes. However, the solution has reached a local minimum and further improvement is not possible.



Initial Solution	Final Solution
Objective:	Objective:
6.0182920842048935 hr	4.818495871627511 hr
Routes:	Routes:
26	26
Routes Summary:	Routes Summary:
0 196 36 128 175 7 16 91 194	0 196 4 128 175 7 16 62 91 194
0 6 26 125 88 116 115 200 133	0 6 29 26 125 46 200 133
0 117 24 13 118 77 53 142	0 117 24 13 118 77 53 142
0 43 162 170 179 199 5 147 71	0 43 197 162 179 48 105 71
0 123 195 186 155 165 183 143 109	0 195 186 145 155 165 183 143 193 109
0 161 15 130 2 61 54 73 119	0 97 107 15 130 2 61 73 131 119
0 60 134 154 98 135 59 17 51	0 140 134 154 98 135 59 17 51
0 97 137 149 126 80 84 10 178	0 137 149 84 44 110 10 49 178
0 87 169 129 1 48 44 49 132	0 60 87 169 129 168 115 116 132
0 197 141 46 18 168 82 150 19	0 83 141 18 19
0 29 41 45 171 21 152 99 156	0 45 171 21 187 152 99 20 156
0 107 79 145 40 30 191 106 192	0 161 79 124 40 30 191 106 192
0 23 63 163 158 62 3 144 153	0 23 63 163 158 174 3 144 153
0 42 96 198 174 90 122 31	0 42 198 90 31
0 124 35 167 25 11 66 146 38	0 35 167 25 66 39 146 38
0 189 75 69 12 57 190 182 127	0 75 69 11 12 57 190 182 127
0 74 52 55 72 112 50 64 101	0 180 74 52 55 72 80 112 54 101
0 131 180 173 103 102 111 27 164	0 173 103 102 111 199 27 5 147 164
0 58 14 47 138 187 39 20 113	0 58 14 47 138 176 104 122 113
0 108 33 67 159 81 32 193 28	0 108 33 159 67 81 32 28
0 22 139 185 114 65 76 85	0 22 139 185 114 65 76 64 85
0 140 86 166 151 93 121 9	0 86 166 126 151 93 121 50 9
0 83 68 94 100 105 110 92	0 68 170 94 1 100 82 150 92
0 160 184 104 176 70 172 188	0 160 96 184 70 172 188
0 89 136 37 95 157 148 78	0 189 89 41 136 37 88 95 157 148 78
0 181 4 120 56 177 8 34	0 123 181 36 120 56 177 8 34

The objective function was trapped in the optimal cost of 4.818495871627511 hours after 87 iterations.