Large Scale Optimization: MAIN ASSIGNMENT

2021-2022 (Part Time)

Professor: Emmanouil Zachariadis

Group: Mentakis Dimitris – 2822024

Ntetsika Alexandra – 2822014

Vlassi Georgia – 2822001

Vretteas Stylianos – 2822003

Description

The scope of this project is to design and construct a greedy algorithm, which consists of five routes and a list of different customers who will be served. The problem we need to solve is a Vehicle Routing Problem (VRP) with Profits in which each route starts from depot and ends to depot. The algorithm has been constructed to follow two specific rules which form our objective function. First and foremost, our goal is to maximize the profit gained from the customers who are visited in each route. Subsequently, the cost is needed to be minimized. Every route should not exceed a time limit set in 150.

For this purpose, we built four Python files: **main**.py, **Solver**.py, **SolutionDrawer**.py, **VRP_Model**.py. In the VRP_Model.py the three following Python classes are constructed:

The **Node** class, represents every customer which has the following attributes: id, x, y (coordinates), service time, profit, isRouted (a flag which denotes if the customer has been visited).

The **Route** class, represents every route which has the following attributes: sequenceOfNodes which is a list of the nodes (customers) who belong in this route, duration which denotes the summary of distance time between nodes and service time of each node, profit which denotes the summary of profits of all nodes including in the route, timeLimit that is equal to 150. Every route has been initialized to start and end with depot.

The **Model** class, represents the model which is executed and has the following attributes: all_nodes a list which includes depot and customers, customers, dist_matrix, number_of_tracks.

In **Solver**.py the below classes are created and initialized: Solution, RelocationMove, SwapMove, UnroutedInsertionMove, ProfitableSwapMove, CustomerInsertionAllPositions and Solver. The solution of our problem is executed from a function inside Solver named *solve*.

Our solution was implemented by using Minimum Insertions Algorithm. More specifically, we developed the function *IdentifyMinimumCostInsertion* which checks every customer at every route and every position in the algorithm. It also calculates if the customer's service time respects the route time limit and if there is a maximization in total profit. After that, we store the attributes of the customer that meets the above restrictions and apply them to our final solution.

Having created an initial solution with Minimum Insertions methodology, we used different local search operators to reconstruct and optimize our initial solution. Additionally, in order to minimize the cycling effect we use the Tabu Search iterator method.

The Local Search operators are:

- **Relocation Move**: With this operator we check if the cost of a relocated customer, who is already visited, will be reduced depending on his new distance from the before and after customer.
- **Swap Move**: With this operator we check if the cost of a swapped customer, will be reduced depending on his new distance from the before and after customer. The swap moves are implemented both in the nodes of the same route and between the nodes of every other route.

- **Insertion Move**: With this operator we check if the total profit of the solution will be increased by inserting a non-visited customer respecting the cost of the solution.

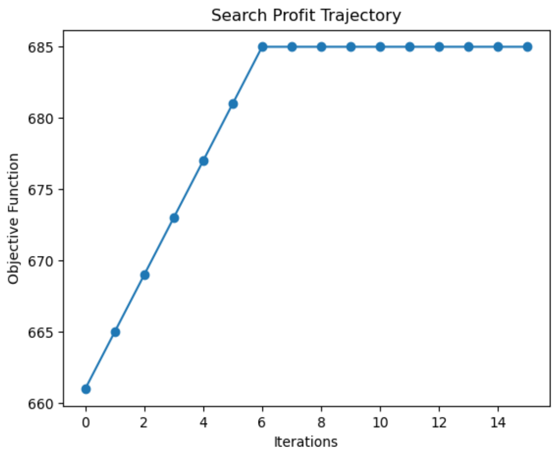
- **Profitable Swap Move**: With this operator we check if the total profit will be increased compared with the profit of the new unvisited customer. Moreover, we check if the total cost of the solution will be reduced depending on the distance from the before and after customer when we swap a customer with a visited one.

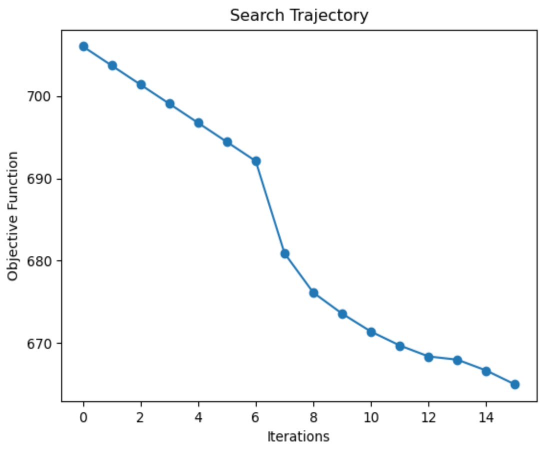
In the development of **VND** (Variable Neighborhood Descent) all operators that we built above are applied repeatedly in order to get the local optima of the solution. The last brings profit maximization and cost minimization. To be more specific, we create an instance of each operator and for each of them we find the best move, store it and apply it if and only if it respects total profit maximization and cost minimization. In every reputation we only store the results that respect the prerequisites in order to draw relevant visualizations.

In **main**.py a new empty instance of Model class is created. After that we build the model in order to assign values to customers. Moreover, an instance of class Solver is created and the solve function is called.

In **SolutionDrawer**.py is used to draw the solution routes and their evolution until the final result. Furthermore, two figures show the progress of cost and profit.

Results and Observations





After implementing VND method with the following order at operators Profitable Swap, Insertion, Swap, Relocation, we reach the solution profit equal to 685 compared to 657 we had at the beginning. The solution cost was reduced from 708 to 665. The above results accomplished after 16 repetitions. After the 6th iteration the profit stabilized and the cost reduced abruptly.