Heuristic Approaches for Heterogeneous Vehicle Routing Problem with 2D Vector Demands

Taha Huzeyfe Aktaş

January 11, 2022

Abstract

This study considers a routing problem with a heterogeneous fleet. The fleet size is not limited, and demands are 2D vectors. Thus, this study cares additional contraints for demands unlike the classical vehicle routing problems. The problem is modeled mathematically, and a feasible solution is obtained with Cplex solver. In this study, heuristics are built to get close feasible solutions to Cplex in more reasonable time. Various approaches are developed to investigate their behaviour and parameter tuning routines. Two different constructive heuristics are created, and their effect to improvement heuristics as an initial solutions are analyzed. There are also three different improvement heuristics, to improve an initial solution. Well-known first improvement and best imropovement strategies are analyzed in the study. The stochastic approaches while accepting non-improving solutions to prevent stuck are also investigated with simulated annealing algorithm. Finally population based heuristics are also evaluated. Two different graph representation is built to solve the problem. A genetic algorithm is also introduced for future studies but not completed. These study contributes to see heuristic approaches to deal with unusual constraints in an optimization problem. Experiments also show that why heuristic approaches should be used to deal with such complex problems that considerably close, even better, objective values are obtained in significantly less time compared to an exact solver. Moreover, results also show that tuning parameters has great importance and this study provides some pre-knowledge about parameters' effects on performance.
Keywords: heuristic algorithms, NP-hard, VRP, heterogeneous fleet

1 Introduction

Vehicle Routing Problem(VRP) have been investigated for long time and has various types. VRP is simple assign vehicles to destinations according to an objective. The book of Toth and Vigo(2014)[9] describes the task for the problem in a general form. They explain task as determination of routes to perform partial or a complete tranfers with a vehicle fleet by minimizing the cost. The solution is also explained as assignment of vehicles to requests in which sequence

for the feasibility.
Objective, vehicle, visit and customer restrictions determines the type of the VRP. This problem represents delivery of goods in real life. There are terms in VRP whose variation determines difficulty and type of a problem.

- *Vehicles* are used to transfer good from one place to another. Vehicles can have a *capacity* that is not exceeded or be *non-capacitated*.
- *Customers* are distributed on map in VRP and they are tied with arcs. Customers have also some amount of demand that should be delivered.
- *Depot* is the point where all goods are started to be distributed. Vehicles start their transportation from the depot. There might be *one* or *more* depots according to the problem. Vehicles are usually expected to turn this location again after all goods are delivered.
- *Arcs* represents the ways between customers or depot. Arcs are usually constant cost according to their distances.

Even though these are only main terms, many approaches have been made via small restrictions or changes on these terms. Numerous variations of VRP, that is called also as rich VRP's, have been introduced by many researchers. The potential objective could be minimizing the total road cost, minimizing the number of vehicles used and so on. Adding limitations like time window will also restricts the objective value. Furthermore, more restriction can be added to vehicles like a certain capacity that cannot be exceeded. These type of variations produce new terms for VRP like capacitated vehicles, time window, heterogeneous fleet etc. and are made use of according to real life requirements.

2 Problem Statement

This project aims to complicate the objective of the VRP by enabling various types of vehicles that is relatively less studied in literature. Vehicle types refer to different capacity and fixed costs of them. This situation creates the trade-off between more loading capacity and cheaper transportation. Variation of vehicles will harden the objective by enlarging the feasible space and so the search area. The cost of arcs also depend upon the vehicles.
Another view of the project is its carriage restrictions. In typical vehicle routing problems, only one dimension is used. This project will assume both volume and weight restrictions on vehicles. This approaches the problem 2D bin packing problem while items are also needed to be transferred. This constraint will also approach the problem to a real life problem. Two-dimension can be considered as weight and volume specifies of items. Thus, this situation is more like a real situation since only weight or volume is do not represent an item perfect anytime. For instance, bales of straw is low dense material so its weight restriction could easily be satisfied while the offered vehicle does not have required volume capacity. Reverse of this situation exists when the items are high dense.

3 Objectives

The main objective of the project is to search for meta-heuristic algorithm for the problem defined. The algorithm should be able to find a point close enough to optimal point. Desired *mean absolute percentage error* should be close to zero and the algorithm also should be improved to give a significant effect on this metric.
Another criterion is time. *Metaheuristics* has to provide an huge time improvement compared with exact algorithms. Thus, just a short time interval should be used to reach desired complexity. Moreover, some traditional meta-heuristics will also be tried to be overcome both in terms of time and error.
This project also aims to add new constraints related with the dimension of the items. This change can also bring 2D bin-packing approach to VRP and so the problem can be considered as various size bin packing problem with the objective of *traveling sales problem*. This approach will also be evaluated to observe its effect on the objective and be used to improve efficiency of the meta-heuristic.

4 Preliminary Literature Review

Vehicle routing problem is well known computer science and operational research subject since more than half century. This project focuses only a sub-subject of it and will try to increase its scope. Therefore, heterogeneous fleet vehicle problems are mostly utilized resources for this project. *Baldacci et al. (2008)*[1] gives a deep introduction to this subject and summarizes the related researches up until release of the paper. This paper used as book chapter and a great resource for the researchers. It is more like a lecture book therefore give all the information needed categorize and formulate the problem. It has introduced variety of mixed fleet VRP's by referring related works on subjects. Fleet size, fixed cost and routing costs are stated as problem elements and determinant factor of type of heterogeneous fleet VRP's.
Fleet size is placed in literature either as limited or unlimited. Fixed costs mean the cost of using a vehicle which is ignored in some of the studies. Last determining factor for problem type is routing cost. The paper also separates group of researches whether routing costs are dependent to vehicle or not. This project aims to also add dimension of items to these grouping fields.
Golden et al.(1984)[2] is declared as where the mixed fleet vehicle problem defined first. There are many types of rich vehicle routing problems as stated before, and that study presents a new view to the problem. It takes care of classical methods to these newly generated problem. Instead of it is the first paper about the subject, many heuristics are considered and evaluated briefly even though they are not deeply investigated. *Liu and Shen (1999)*[5] added time window to this problem. They again enrich the the heterogeneous VRP and experiments with sufficient set of hyper-parameters.
Heuristic for VRP can certainly inspire for this study and *Toth and Vigo (2014)*[9] provide a book chapter for the subject. Their book is all about vehicle rout-

ing and allocate a complete chapter for usage of heuristic algorithms in VRP. This chapter tells about the heuristic algorithms for the VRP and separates them according to the groups of constructive, improvement and meta-heuristic methods. The chapter does not state theoretical background of the methods but many algorithms and approaches are addressed. Thus, the book includes a deep library for the problem and its heuristics.
Schneider et al.(2014)[8] have put an modern approach to the problem and they investigated electric vehicle routing problem. The key point of this approach is that vehicles have limited battery capacity and has to visit recharging stations. This paper is crucial since it first takes environmental factors into account in this subject. Thus, it also gives spotlights to new researchers. *Hiermann et al.(2016)*[3] takes it one step further and provide mixed electric fleet to the problem. Vehicles also differ in their battery size rather than just cost and capacity in mixed electric fleet. Former study is more subject-oriented and give detailed information about the subject since it introduces relatively new constraint to the problem. On the other hand, latter study is like complementary of it since it provides wide variety of experiments and investigates various approaches in more complex problem.
Bin packing approach is another branch of that problem, it is also deeply investigated research area in operation research. *Lodi et al.(1999)*[6] investigates heuristics and metaheuristics for different 2D bin packing problems. It also provides experiment results and comparision between algorithms. Then many papers are also published about the subject. For instance, *Hong et al.(2014)*[4] gives a hybrid heuristic method to this problem and *Wei et al.(2013)*[10] examines a different objective. However, most studies are on orientation based bin packing and so it does not appropriate for this since since the dimension are thought as weight and volume for this project. These heuristics would be helpful for future works to take intended study to one step forward. *Pessoa et al.(2021)*[7] published a recent study that uses VRPSolver for a variety of bin packing problem. This paper arranges the bin packing problem as VRP and provides a exact solution. Any heuristic is not offered in this paper, but would be helpful to investigate close views of two different problems.

5 Mathematical Modeling

The described problem is an optimization problem. Therefore, objectives, decision variables and constraints are needed to be claimed clearly. Heterogeneous fleet vehicle problems already have a clear formulation. First of all, decision variables are needed to be defined. *Golden et al (1984)*[2] formulate their heterogeneous fleet vehicle routing problem for single dimensional items. However, since experimented problem cares both weight and volume atributes then additional variables are required.
Decision variables;

- x_{ij}^k is binary and states whether route between node i and node j is used

by vehicle type k.

- r_i^1 is flow variables for weight dimension in node i. It shows how much weight reached by the vehicle that visits node i just after visiting node i.
- r_i^2 is flow variables for volume dimension in node i. It shows how much volume reached by the vehicle that visits node i just after visiting node i.

Problem parameters;

- d_i^1 shows weight of the load for node i.
- d_i^2 shows volume of the load for node i.
- α_k^1 shows weight capacity of the vehicle type k.
- α_k^2 shows volume capacity of the vehicle type k.
- f_k shows fixed cost of the vehicle type k.
- u_k shows unit per distance cost of the vehicle type k.
- c_{ij}^k shows the cost to go between node i and node j with the vehicle type k.

Then, the objective is given below.

$$min \sum_{k=1}^{T} f_k \sum_{j=1}^{n} x_{0j}^k + \sum_{k=1}^{T} \sum_{i=0}^{n} \sum_{j=0}^{n} c_{ij}^k x_{ij}^k$$

The problem try to minimize both fixed and routing costs. Unlike *Golden et al (1984)*[2], the unit costs also vary in this problem. There are T different types of vehicles in total. The index for the depot iz zero, so sum of x_{0j}^k actually show how many vehicles are used. There are also constraints, and they depend on the type of the mixed fleet VRP like whether there is a limit on the number of vehicles or not. This project might also add new constraints so the vehicle should satisfy 2D necessities instead of only one.

st

$$\sum_{k=1}^{T} \sum_{i=0}^{n} x_{ij}^k = 1 \qquad (j = 1, \cdots, n)$$

$$\sum_{i=0}^{n} x_{ip}^k - \sum_{j=0}^{n} x_{pj}^k = 0 \qquad \substack{(k=1,\cdots,T;\\ p=1,\cdots,n)}$$

$$r_0^b = 0 \qquad (b\epsilon\{1,2\})$$

$$r_j^b - r_i^b \geq (d_j^b + a_T^b)\sum_{k=1}^{T} x_{ij}^k - a_T^b \qquad \begin{smallmatrix}(i=0,\cdots,n;\\ j=1,\cdots,n;\\ b\epsilon\{1,2\})\end{smallmatrix}$$

$$r_j^b \leq \sum_{k=b}^{T}\sum_{i=0}^{n} a_k^b x_{ij}^k \qquad \begin{smallmatrix}(j=1,\cdots,n;\\ b\epsilon\{1,2\})\end{smallmatrix}$$

$$x_{ij}^k \in \{0,1\}\ for\ all\ i,j,k$$

The additional constrainsts are added to the problem for second dimension. The same constraints has to satisfied for the second dimension.
The main purpose of this project is to create heuristic algorithm to solve an NP-hard problem. A problem with such a complexity is so complex to build up. Therefore, a meta-heuristic would be the best helpful to find relatively fine solutions without huge time and space requirements.

6 Data Description

The data consists of two main parts in the project which are customers and fleet properties.

6.1 Customer Data

In a typical VRP, customers have two properties which are their coordinates and demand. Coordinates of customers and depot are taken from the Duhamel et al.[11]. This paper provides a set of HVRP samples and their solutions. Different than general concept, demands are 2D vectors, volume and weight, in this project. Therefore, customers have one additional property for the demand. For instance, if a customer is represented as [36,26,7,131], it means that customer customer is located at [36,26] in a 2D coordinate plane and its demand's weight is 7 units, while volume is 131 units. Since the project has one additional property unlike the others, weight and volume properties are just created. Hence, weights are created as a random integer in a uniform plane of [5,30]. Similarly, volumes are uniformly distributed between 50 and 250.
In this sense, two different population size is used which are 20 and 50. Moreover, three different weight and volume properties is created for each population size. Thus, six different samples are obtained in total.

6.2 Fleet Data

n the heterogeneous VRP's, more than one vehicle types exist. In this problem vehicles are not restricted with any time constraint or not limited with a certain number. They only have a fixed cost and unit costs per distance unit. Like the customer data, vehicles should have one additional property for both their

Table 1: Fleet Data

Fleet#1	**Fleet#2**	**Fleet#3**
[40,500,750,0.8]	[40,500,750,0.8]	[40,500,700,0.8]
[80,750,1000,1]	[80,750,1000,1]	[80,750,1100,1]
	[100,1000,1250,1.4]	

volume and weight constraints. A vehicle represented with four number which are weight capacity, volume capacity, fixed cost and unit cost respectively. For example, if a line like [40,500,750,0.8] is used to represent a vehicle, that means vehicles can carry up to 40 units weight and 500 units volume. Its fixed cost, that is cost of using one vehicle from this type, 750 and the cost for per distance unit is 0.8.

Three different fleets are created for the project. First fleet is created with two vehicles in the way that their weight/volume capacity ratios are different, so both weight and volume become more likely to a limitation for different customers. Second fleet is created by adding a larger vehicle to the fleet to observe whether algorithms are capable of decreasing cost by using additional vehicle. The fixed cost difference between vehicles is increased for the last fleet to see the effect of it, for example, whether algorithms can response by lowering usage of more expensive vehicle. Furthermore, vehicles in a fleet are created in a growing order, and neither weight nor volume capacity of a small truck can exceed larger truck's weight or volume.

Finally, 18 different samples are obtained with 3 different samples for each three fleet and two population size combinations. Each samples is represented like (N=n, fleet#H, sample#d) where n is population size(20,50), H is the number of fleet available(1,2,3), and d is the sample number(1,2,3) for this combination.

7 Benchmark

The described problem is an unfamiliar problem in the literature. Therfore, unfortunetely there exists no heuristic solver that gives reliable and fast results. On the other hand, this is an optimization problem, and exact solvers are available to be used. Cplex is used for the purpose. Mathematical modeling is already been shown in previos section, and so this model is implemented into Cplex solver. Since the problem too complex, the solver is limited with 1000 seconds time limit. The results are presented in Appendix B.1. Results are satisfactory but it is clear that Cplex does not use a smart search method, probably because of that it evaluates every single branch. Therefore, it is run with such long times.

8 Construction Heuristics

8.1 Cluster-first-route-second strategy

8.1.1 Steps

Constructive heuristic is used to get a feasible and relatively acceptable solution to the problem. *Cluster-first-route-second* approach is used to build the algorithm. The codes are presented in appendix A.1 and this algorithm will be called as **Algorithm 1a** by now.

Cluster

Clustering phase divides customers to cluster according to their distances to each other. Total weight or volume required for the customers in any cluster cannot exceed the weight or volume capacity of the largest vehicle.

1. Sort distances for each node pair in increasing order.
   - d_{ij} : distance between customer i and j
2. For each pair(i,j) in the created list, there are three possibilities.
   - both i and j are already in cluster.
   - One of i or j is in a cluster but other is not. In this case, if unplaced customer does not violate the maximum capacity constraints of largest vehicle, place it into the same cluster.
   - None of them is in a cluster. Thus, if they fit into a cluster without exceeding any constraint place them in the same cluster. Otherwise, build separate cluster for both of them.
3. If there is any customer who is not assigned to any cluster after iterations, build a cluster that contains only that customer.

Route

In this process, routes tried to be generated for each cluster with minimum cost. These routes have to start from the depot and visit all customers in the cluster and return the depot. Below processes are executed for each cluster.

1. Assign one vehicle for each customer. Prefer the cheapest vehicle that is capable.
2. Sort distances for each node pair(i,j) in increasing order.
3. For each pair(i,j) in the list, if both i and j are whether first or last visited nodes of their relevant vehicle, merge these two vehicles in a proper way.
4. Repeat these processes for each cluster.

8.1.2 Experiments and Discussion

As stated in the data description section, 18 different samples are available. Each sample is run 5 times, and so 90 experiments have been made for the constructive heuristic in total. Since constructive heuristic is designed in deterministic way results are all same for each sample except the time. Appendix refexp-cfrs, show the summary of experiments. Average values are shown for time, cost and total travel distance. Vehicles column shows that how many vehicles from each type on average. For example, in the first line [2 4] shows that smaller vehicle is used 2 times on average while larger one is needed 4 on average. Since, this is a deterministic algorithm all runs have exactly same solution.
Experiments shows that increase is population size significantly affect the run time of the algorithm. When population size is increased 2.5 times from 20 to 50, algorithm time increases more than 200 times. This issue becomes more problematic for larger size, thus simpler but faster ways would be tried to overcome with it. Moreover, larger size solutions clearly cannot get close enough to cplex solutions.
Another drawback of the constructive heuristic is it does not do any cost analysis. The difference between fleet#1 and fleet#2 exhibits this situation. Even though fleet#2 has one additional vehicle, which should lower the cost since alternatives increases, the cost increases for almost every case. Moreover, this situation can also be observed by comparing fleet#1 and fleet#3 vehicle usages. Event though, fixed cost of larger vehicle is magnificently high in the fleet#3, constructive algorithm could not response the this change and same vehicles are used since cost analysis are not taken into consideration.

8.2 Closest Neighbor Heuristic

Closest neighborhood algorithm is well-known traveling salesman problem heuristic. In this case, this heuristic is adapted to the vehicle routing problem. The logic behind it is same that is to tie closer nodes to each other. However, the additional constraints must also be satistified for a VRP.

8.2.1 Steps

1. Find closest node to depot among unassigned nodes. Assign a new vehicle to it.

2. Find closest node to end node that satisfies maximum vehicle capacity constraints, and assign that node to as new end.

3. Repeat these steps until a vehicle cannot take any more node.

4. Repeat all steps until there is no assigned node.

This algorithm simply checks all nodes from closest to furthest and add them to vehicle unless any capacity is not exceeded. This is a so greedy heuristic

and does not consider rather than closeness of the nodes. Codes are shown in *Appendix A.2*. This algorithm will also be stated as **Algorithm 1b**.

8.3 Experiments and Discussion

5 runs are made for each sample as in the previous constructive heuristic. All attributes like total distance, and vehicle counts are collected and evaluated. Results are presented in appendix B.3. This heuristic is a deterministic, but 5 different runs are made to observe how much time deviates over runs.
Experiments clearly show that closest neighborhood algorithm outperformed the previous algorithm in terms of both performance and time. Population size does not cause time to increase dramatically which gives advantage when an initial solution is needed for improvement heuristics. Moreover, performance is always better than previous heuristics. Then solution is even better than provided benchmark for two samples that are N=50,fleet#2, sample#2 and sample#3.

9 Improvement Heuristics

Even though constructive heuristics provide solution, they are probably open to improvements in most cases. Improvement heuristics will help to get more qualitative solutions by using the output of constructive heuristics as an initial solution.An initial solution obtained after the constructive heuristic. Improvement heuristics are designed to search for improvement in the solution. Three different algorithms are used which are *first improvement, best improvement,* and *simulated annealing*. Before diving into details, common methods used for all improvement method are explained. These are the strategies that improve the quality of a feasible solution, and they are all used for all the algorithms.
add/remove: This method selects a random customer from a random vehicle and look for a place in another vehicle's route for it.

1. Select a random customer, *c1*, and random vehicle, *v1*, that does not visit the selected customer.
2. Search for the closest customer c2 that v1 visits.
3. Look for *c2*'s neighborhoods in the *v1*'s routes. Take the one closer to *c1*, assume it is *c2'*.
4. Remove *c1* from its current place and put it between *c2* and *c2'*.

inter-switch: This method selects two vehicles randomly, and two customers of them and switch the place of them.

1. Randomly select two different vehicles, *v1* and *v2*.
2. Randomly select two customers from *v1* and *v2*, they are *c1* and *c2* respectively.

3. Switch *c1* and *c2*. Thus, *v1* visits *c2*, and *v2* visits *c1*.

intra-switch: This method selects a random vehicle, and two customers from that vehicle. Switch the place of these two customers.

1. Randomly select one vehicle, *v1*.
2. Randomly select two customers from *v1*, they are *c1* and *c2*.
3. Switch *c1* and *c2*.

mix: These method combines all their strategies and randomly perform one of them in each iteration with equal chance.
These methods are evaluated for each improvement heuristic. They are used as parameters, so only one of the methods has to selected for each run. These parameters called as method for the remaining part of this section.

9.1 First Improvement Heuristic

9.1.1 Steps

First improvement heuristics uses one of the methods described above and perform the first improved option in each iteration. This heuristic will be called as **Algorithm 2a**.

1. List all combinations for the given method. For instance, all customer pairs one from *v1* and one from *v2* for the inter-switch case.
2. Shuffle all the combinations.
3. For each combination, check whether an improvement is made. If it does, perform the action and pass to the next iteration.
4. Repeat until stopping criteria exists.

9.1.2 Experiments and Discussion

Stopping criteria is used as decision parameter for experiments. Non-improved steps are evaluated for 15 and 30, that means that algorithm stopped if no improvement has been made for last 15 or 30 steps. There are also four different methods, which are also other parameters. Moreover, two different initial solutions, that comes from two constructive heuristics, are also experimented. Thus, four different methods and two different stopping criteria with two initial solutions are used for 18 different samples. Each parameter combination is run 5 times and so 1440 runs are taken in total. Solutions are presented in *Appendix B.4* for only three samples. First of all, when initial feasible solution is better algorithm tends to give better results. Most best cases are reached with Algorithm 1b. Moreover, stopping condition does not seem to create a clear effect on results. When also considering the time stopping condition can be one of two values. When methods are compared intra-switch method does

not contribute much to the constructive heuristic. This situation is the cause of it is simply approach and does not affect vehicle types, and so their costs. It only gains from unit costs, that does not have an important impact in these cases. When other three methods are compared, it is clear that inter-switch method for first improvement algorithm is somehow behind of other two methods. 'mix' and 'addremove' methods seems competitive when initial solution is created with Algorithm 1a. Even though add/remove method beats 5 times of 6 cases on average when stopping condition is 15, mix method exhibit and crucial average improvement when stopping condition becomes 30. However, when initial constructive is altered to Algorithm 1b, 'mix' method clearly works better with it. Finally, parameters of first improvement heuristic can be set to mixed search method and 30 non-improving steps as stopping condition together with Algorithm 1b. This parameter generally works well, it even can outperform the benchmark over than 5% improvement in the cost.

9.2 Best Improvement Heuristic

9.2.1 Steps

Similar to first improvement heuristic, best improvement heuristic make use of every method and search for an improvement. Unlike the first improvement, this heuristic evaluates all possible combinations and perform only the best one. This algorithm will be called as **Algorithm 2b** for the report.

1. List all combination for the given method. For instance, all customer pairs in a vehicle for the *intra-switch* case.

2. For each combination, check whether an improvement is made over the best cost recorded. If it does, record the cost as best cost and also combination parameter.

3. After iteration comes to end, perform the best combination found. If no better action is found, just skip the next iteration.

4. Repeat until stopping criteria exists.

9.2.2 Experiments and Discussion

Like the first improvement strategy, same stopping criteria are used with 4 different methods. 5 runs are taken for each 18 samples with these methods and stopping conditions. Finally, 720 total run is obtained. Experiment results are summarized in *Appendix B.5* for same three samples.
Stopping condition shows different effect for different initial solutions. When stopping condition is increased algorithm with initial solution Algorithm 1b gives considerably better results for mix and add/remove search methods. Population size is also another factor that change the effect of the stopping condition.
When looking from the methods sense, situation is similar to first improvement heuristic. Intra-switch method does not have even 1% effect on the objective.

Standard deviation is almost zero, which means that algorithm is stuck in similar points. From different view, this situation shows that route strategy for the constructive heuristics is good enough that even best improvement strategy cannot find any better points.

9.3 Simulated Annealing

9.3.1 Steps

Simulated annealing uses a stochastic approach while accepting a solution unlike the previous improvement algorithms. If the searched point provides an improvement on the problem, it is accepted. If it does not contribute to objective, then it can also be accepted with a probability. As the algorithm approaches to end, it more unlikely to accept non-improving solution. This algorithm will be called as **Algorithm 2c**.

1. Set temperature, T,
2. Select a random combination for the given method. For instance, take one random customer c_1 and one random vehicle v_1 for the add/remove method.
3. . Calculate the new cost if the randomly selected action is performed. For example, remove c_1 from its current vehicle and put into v_1.
4. If new cost is better than the current, then accept it and execute the action.
5. If calculated cost is worse than current, calculate acceptance probability:
   - *acceptance probability = exp((current cost - calculated cost)/current Temperature)*
6. If acceptance probability exists, then accept the new solution.
7. Perform steps 2 to 6 until given number of epochs.
8. Update temperature. Cooling parameter is used to update temperature.
   - *temperature = temperature * cooling parameter*
9. Repeat until stopping criteria exists.

9.3.2 Experiments and Discussion

There are various parameters for the simulated annealing that are initial temperature, stopping criterion, cooling parameter, number of epochs, acceptance criterion etc. In this project number of epochs and cooling parameter are experimented. Limited resources prevent running each combination, but sufficient data is collected. Experimented parameters are determined on the way. For

instance, two is added as epoch length hyper-parameter first, but then it is not used since its performance is not competitive. In this sense, these combinations of paramters are performed with four different search method.
initial solution, cooling parameter, max epochs = (Alg1a, 0.5, 2), (Alg1a, 0.5, 10), (Alg1a, 0.7, 5), (Alg1a, 0.7, 10), (Alg1b, 0.7, 5), (Alg1b, 0.7, 10), (Alg1b, 0.9, 5), (Alg1b, 0.9, 10)
Four methods are also run for each parameter set, and so 32 combinations are obtained. 5 runs are taken for each possibility and so 2880 total run is taken for 18 samples. Appendix B.6 shows a summary of results for selected three samples.
The initial solution is not as determinant as previos methods, it is still again an effective parameter to determine the quality of the solution. This situation is probably because of the Algorithm1b is clearly much better than the Algorithm 1a.
Experiments also show that parameters should be configurated according to population properties. For instance, while smaller instances like to be cooled slower, larger instances can be preferred with a bit faster coolings. Like this example, larger fleets also generally works better with lower cooling but this assumption is not as strong as previous. In general, more iterations with higher cooling rate and more epochs give better results and requires more time, but population properties are also crucial to set hyper-parameters.

10 Population Based Heuristics

Population based heuristics rely on cooperative behaviour of multiple solutions. Keeping diversification with some stochastic approaches while searching for better feasible points is the main logic behind it. In this study, two different types of ant colonies and a genetic algorithm for decomposed parts of the problem are implemented.

10.1 Node-to-node Ant Colony

Ant colony optimization is a good way to search solutions for routing problems since they are easy to present in graph structure. In this study two different representations are tried. This first algorithm is a classical approach that draw an arc between nodes.
Figure 10.1 illusturates a small graph representation of the algorithm for three nodes. The thick red lines shows the route of an ant, where dotted lines are other feasible arcs that the ant would visit. Thus, the solution comes from that ant will be *0-1-0-3-2-0*. This solutions tells that the route for node#1 differs from node#2 and node#3. Thus, first vehicle visits only node#1, while a different second vehicle visit node#3 and node#2 respectively. Vehicle types are chosen according to smallest type that satisfy the related routes capacity requirements.

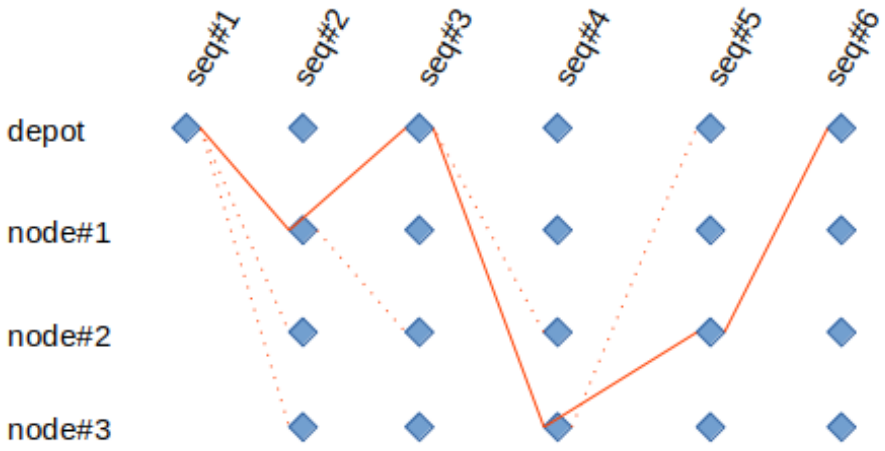

Figure 1: Graph representation of node-to-node ant colony algorithm

10.1.1 Steps

Graph representation is already stated. Pheromones level updates and decision criterion are most crucial part. These parameters are determined after a set of experiments. This algorithm will be named as **Algoritm 3a** for rest of this study. The related code file is available in *Appendix A.5*.

1. Set population.
2. Set initial setup for pheromones.
3. For each ant;
   (a) Start from the depot, so current node is *node 0*.
   (b) Move a feasible next node from the current node according to pheromone levels. More pheromone levels are favored.
   (c) Repeat these steps until all nodes are visited.
4. Update pheromone levels according to the objective value, then evaporation is made.
   - $pheromones+ = 1/\log_n totalCost$
   - $pheromones* = evaporationRate$
5. Repeat all until stopping criterion exists.

10.1.2 Experiments and Discussion

There are many paremeter to be tuned for the algorithm. It is hard to optimize all of them, but some pre-experiments are also performed to decide which

parameters to be tuned and which values should be used. Evaporation rate is fixed to *0.8*, and number of ants in the population is selected as *30* after all. Moreover, stopping criterion exists after *20 non-improving steps*. Weight determination of routes seems crucial, and so related parameters are experimented. First of all, two different fitness function is determined while selecting routes. These are *arithmetic* and *geometric* functions.

- $ArithmeticFitness = \alpha * pheromones(i,j) + \beta * (1/distance(i,j)$
- $GeometricFitness = pheromones(i,j)^{\alpha} * (1/distance(i,j)^{\beta}$

These function determines how likely a *[i,j* route will be selected. The probability of selecting a node after visiting *node i* is determined according to normalized values of above function. Both functions have also α *and* β coefficients, which are other parameter. α is experimented with values *1 and 1.5*, while β is also tested for *0* additional to same values.
Ant colony algorithm was expected to take longer times since it is population based approach, so many feasible solutions are evaluated at a time. The structure, on the other hand, is appropriate to multi-process runnings, so the time can actually be decreased dramatically to more accetable values.
Even though, its much time and space requirement, results are not so satisfactory. Ant colony as many parameters to consider. Thus, more experiments should be executed to covnerge better parameter values.

10.2 Couple-to-couple Ant Colony

This is again an ant colony heuristic, but a different approach has given a try. Node presentations are changed to node couples in this algorithm. Thus, a node is represented by two customer index that are not equal like *[i,j]*. This approach has more knowledge, since tying two nodes shows three customers that are visited concecutively. This knowledge, on the other hand, requires more space and more nodes. The graph representation is shown in *Figure 10.2*.

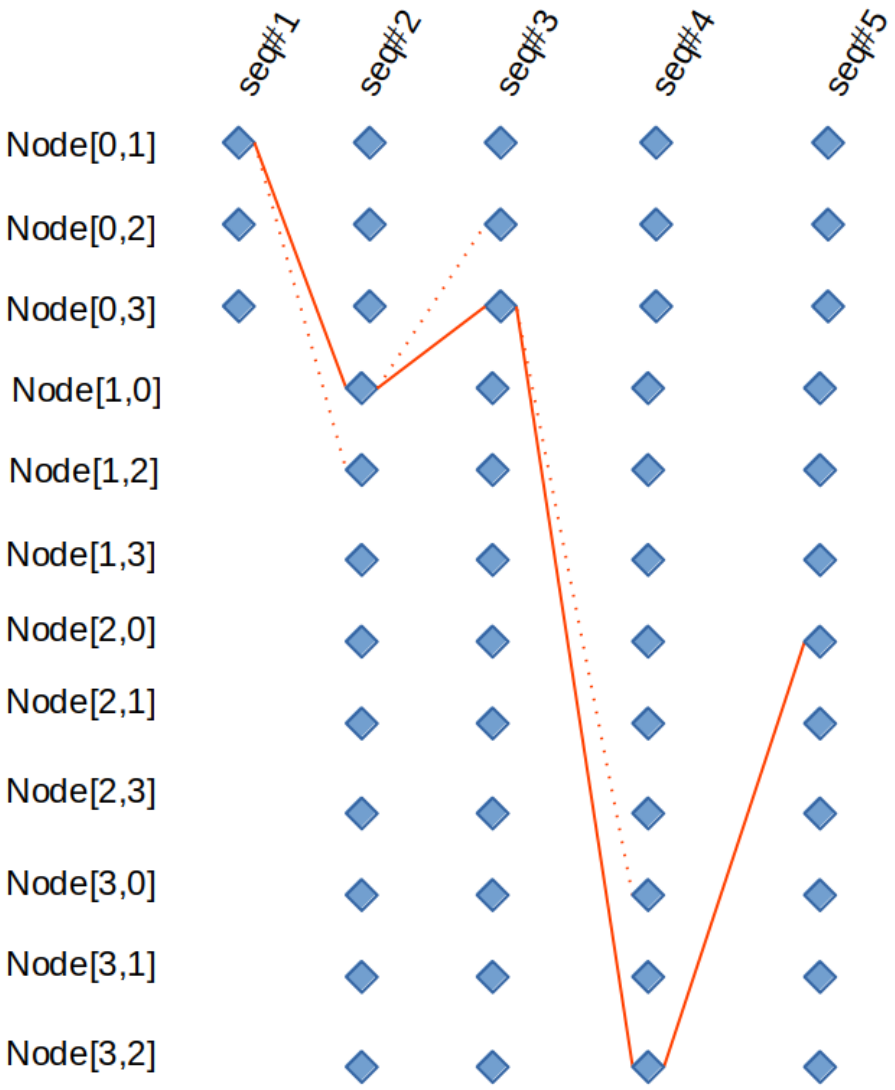

Figure 2: Graph representation of couple-to-couple ant colony algorithm

10.2.1 Steps

The steps are same as the previous ant colony model. In this case, difference is based on graph representation, so heurictic approach is added appropriate to structure. This algorithm will be called as **Algorithm 3b**.

10.2.2 Experiments and Discussion

Parameter settings are also same with the previous algorithm. Solutions are hard distinguish, are outcome is similar that more investigation is needed on parameter settings. Time is similar with Algorithm 3a, and performances depend on the sample.

10.3 Genetic Algorithm for Decomposed Parts

This algorithm divide problem into two sub-problems. First part is assigning nodes to vehicles. This part is actually a bin-packing problem and try to minimize total fixed costs. The problem with relatively high fixed costs, like in this study, cen prefer fewer vehicles usage instead of traveling less. Thus, this first part has importance that search for minimizing the fixed costs exist here.
On the other hand, vehicle assignment problem will most probably have multiple optimals when routing cost is ignored. Therefore, an auxiliary fitness function can be used in this stage. The purpose behind using a population based heuristic also comes to fore, that many individuals are aimed to produced with similar number of vehicle usages with minimal costs, and their routing phase will be determinant to select best option.
Fitness value should be also good prototype of routing phase in order to achieve qualitative solutions. However, this is not an easy issue, and needed to be investigated deeply. An simple function is used for now, but this function is also aimed to be improved. This simple function just consider fixed costs and favors individual with bins that has fewer items in itself.
After deciding clusters, routing phase is the only remaining part. The work till now only considers best individual obtained by genetic algorithm, and routing is the same as in Algorithm 1a. It is used since is ready to go, but there would be better routing ideas even though they were not implemented yet. One of the idea is that getting best some number of solution from genetic algorithm, and building a new genetic algorithm for the routing part. This seems a nice idea, because more indivuals are made of use in this situation, as stated above.
Another option might be using an exact solver like Concorde for the remaining part, but the quality, and so the fitness function, of the genetic algorithm should be more reliable. One instance ha a finite number of routes and they are smaller compared with all population. Thus, using exact solvers might not be so costly. However, these are all experimental issues, so they are needed to be implemented.

10.3.1 Steps

Steps of the genetic algorithm that is implemented up until now is listed below. However, this is not a final work and has many lacks. There is no cross-over, it is worth to try. The development ideas has also stated, so they can also change the steps of the algorithm. This is just a main skeleton.

1. A random feasible generation is created.

2. Calculate fitness values.
3. Elites directly goes to next generation
4. Tournament is made for remaining of the next generation. Best individual continues
5. Mutation is executed.
6. All steps are executed until stopping criterion exists.
7. Best solution is used to build cluster.
8. A route is built with the same approach in Algorithm 1a.

10.3.2 Experiments and Discussion

This algorithm is just in its baby steps, so there are not sufficient experiments to reach a conclusion. Discussion about building steps are made at the begining of the section. It can be stated that with a few experimens, genetic algorithm works fine, because clustering phase compared to Algorithm 1a is much better.

11 Conclusion

- Initial solution for improvement heuristics are matter to qualitative solutions. Two different initial solution highly effective on the final solution.
- Intra-route method is not an effective search method for whole problem, but it is beneficial when used with other search methods together like mixed method or variable neighborhood search.
- Mixed methods usually the one that gives the best solution. Its run time is also higher, but it is probably because its search space is larger.
- The required time for the best improvement method is a bit higher than other improvement methods. This issue can become more problematic in larger datasets.
- Simulated annealing also performs better in the small size populations. However, when population size increases, other improvement heuristics can be competitive.
- Run times of simulated annealing and first improvement heuristics are reasonable that multiple runs can be taken to reach better solutions.
- Vehicle assignments are more determinant than routing phase.
- Even the cplex solver could not find optimal, it reaches acceptable solutions so fast but its improvement is not as good. Thus, getting multiple short runs and best of them, instead of getting one long run could give better results.

- Performances and deviations show that parameter optimization is a crucial steps. A lot of pre-experiments are run to observe which parameter sets should be tested with multipl runs.
- Populations based heuristics have many parameters to be evaluated. All these are should be considered to improve solution quality.
- Solution quality is highly dependent to samples, so it would be a great work to find dynamic parameter settings for different samples. One example is the fitness function used in ant colony heuristics, that take population size into account.
- Population based heuristics needs more attention to structure of the algorithm like fitness function, weight determination etc. These are part highly important to keep both diversification and convergence on stand. Ant colonies, for instance, are tried to with reinforcement options but a logarithmic function has carried objectives to desired levels.
- Simulated annealing seems working better than population based heuristics, but population based heuristics have actually many design issues to consider. Thus, population based algorithms are more open to improvements and less deviated solutions.

11.1 Future Studies

In this study, many alternative heuristics are explained for this new type of vehicle routing problem. These experiments are a good way to see behaviours of different approaches. On the other hand, reaching a single best heuristic would be a great work. Thus, as stated in previous sections, buiding a genetic algorithm whose properties are set specific to this problem is improvement area. Even though experimented results and heuristics are competitive with cplex solver, and simulated annealing can outperform it in more reasoble times, this solution still can be improved. Population based heuristics coomes to fore in the sense, because many feasible solutions are evaluated at a time and it is more suitable for multi-processing. There are many improvement points in the genetic algorithm from fitness function to mutation methods. They are all needed to be evaluated and experimented.

Appendices

A Code Files

A.1 ConstructiveHeuristic.py

```
import numpy as np
from copy import copy, deepcopy

class ConstructiveHeuristic:
    def __init__(self, CustomerManager, Fleet):
        self.cm = deepcopy(CustomerManager)
        self.fleet = deepcopy(Fleet)
        self.clusters =dict()
        self.totalClusterDemands = dict()

    def run(self, isPrint=False):
        self.BuildClusters()
        self.BuildRoutes()
        if isPrint:
            self.printClusters()
            self.fleet.printVehicleRoutes()
        return self.fleet


    def BuildClusters(self):
        ClusterNumber = 0

        arcs = self.cm.sortArcs()

        for arc in arcs:
            if arc[0] == 0 or arc[1] == 0:
                continue
            elif arc[0] in self.clusters.keys() and arc[1] in self
                .clusters.keys():
                continue
            elif arc[0] in self.clusters.keys():
                cCluster = self.clusters[arc[0]]
                if (self.totalClusterDemands[cCluster][0]+self.cm.
                    get_demands(arc[1])[0]<=self.fleet.maxCapacity
                    ()[0]) \
                & (self.totalClusterDemands[cCluster][1]+self.cm.
                    get_demands(arc[1])[1]<=self.fleet.maxCapacity
                    ()[1]):
```

```
                self.totalClusterDemands[cCluster][0] += self.
                    cm.get_demands(arc[1])[0]
                self.totalClusterDemands[cCluster][1] += self.
                    cm.get_demands(arc[1])[1]
                self.clusters[arc[1]] = cCluster
                continue
            elif arc[1] in self.clusters.keys():
                cCluster = self.clusters[arc[1]]
                if (self.totalClusterDemands[cCluster][0]+self.cm.
                    get_demands(arc[0])[0]<=self.fleet.maxCapacity
                    ()[0]) \
                & (self.totalClusterDemands[cCluster][1]+self.cm.
                    get_demands(arc[0])[1]<=self.fleet.maxCapacity
                    ()[1]):
                    self.totalClusterDemands[cCluster][0] += self.
                        cm.get_demands(arc[0])[0]
                    self.totalClusterDemands[cCluster][1] += self.
                        cm.get_demands(arc[0])[1]
                    self.clusters[arc[0]] = cCluster
                    continue
            elif(self.cm.get_demands(arc[0])[0]+self.cm.get_
                demands(arc[1])[0]<=self.fleet.maxCapacity()[0]) \
            & (self.cm.get_demands(arc[0])[1]+self.cm.get_demands(
                arc[1])[1]<=self.fleet.maxCapacity()[1]):
                ClusterNumber += 1
                self.clusters[arc[0]] = ClusterNumber
                self.clusters[arc[1]] = ClusterNumber
                self.totalClusterDemands[ClusterNumber] = [self.cm.
                    get_demands(arc[0])[0]+self.cm.get_demands(arc
                    [1])[0], \
                                                           self.cm.get_
                                                               demands(
                                                               arc[0])
                                                               [1]+self
                                                               .cm.get_
                                                               demands(
                                                               arc[1])
                                                               [1]]
            else:
                ClusterNumber +=1
                self.clusters[arc[0]] = ClusterNumber
                self.totalClusterDemands[ClusterNumber] = [self.cm.
                    get_demands(arc[0])[0], self.cm.get_demands(arc
                    [0])[1]]
                ClusterNumber +=1
                self.clusters[arc[1]] = ClusterNumber
```

```
            self.totalClusterDemands[ClusterNumber] = [self.cm.
                get_demands(arc[1])[0], self.cm.get_demands(arc
                [1])[1]]

    def printClusters(self):
        for key in self.totalClusterDemands:
            nodes = self.getCluster(key)
            print("Cluster #" +str(key) + " has " + str(self.
                totalClusterDemands[key][0]) + \
                  " kg and " + str(self.totalClusterDemands[key
                      ][1]) + " liters of total demands, and for
                      customers: " \
                  + str(nodes))

    def getCluster(self, cluster):
        nodes = []
        for key2 in self.clusters:
            if(self.clusters[key2]==cluster):
                nodes.append(key2)
        return nodes


    def BuildRoutes(self):
        for key, demand in self.totalClusterDemands.items():
            customers = self.getCluster(key)

            for customer in customers:
                self.fleet.assignVehicle(self.cm, [customer])

            arcs = self.cm.sortArcs(customers)

            for arc in arcs:
                if self.fleet.isLast(customers[arc[0]]) & self.
                    fleet.isFirst(customers[arc[1]]):
                    vehicle1 = self.fleet.
                        getAssignedVehicletoCustomer(customers[arc
                        [0]])
                    vehicle2 = self.fleet.
                        getAssignedVehicletoCustomer(customers[arc
                        [1]])
                    if vehicle1 != vehicle2:
                        self.fleet.mergeVehicles(vehicle1, vehicle2,
                             customers[arc[0]], self.cm, reverse=
                            False)
                    del vehicle1, vehicle2
```

```
            elif self.fleet.isLast(customers[arc[1]]) & self.
                fleet.isFirst(customers[arc[0]]):
                vehicle1 = self.fleet.
                    getAssignedVehicletoCustomer(customers[arc
                    [1]])
                vehicle2 = self.fleet.
                    getAssignedVehicletoCustomer(customers[arc
                    [0]])
                if vehicle1 != vehicle2:
                    self.fleet.mergeVehicles(vehicle1, vehicle2,
                         customers[arc[1]], self.cm, reverse=
                        False)
                del vehicle1, vehicle2
            elif self.fleet.isLast(customers[arc[0]]) & self.
                fleet.isLast(customers[arc[1]]):
                vehicle1 = self.fleet.
                    getAssignedVehicletoCustomer(customers[arc
                    [0]])
                vehicle2 = self.fleet.
                    getAssignedVehicletoCustomer(customers[arc
                    [1]])
                if vehicle1 != vehicle2:
                    self.fleet.mergeVehicles(vehicle1, vehicle2,
                         customers[arc[0]], self.cm, reverse=
                        True)
                del vehicle1, vehicle2
            elif self.fleet.isFirst(customers[arc[0]]) & self.
                fleet.isFirst(customers[arc[1]]):
                vehicle1 = self.fleet.
                    getAssignedVehicletoCustomer(customers[arc
                    [0]])
                vehicle2 = self.fleet.
                    getAssignedVehicletoCustomer(customers[arc
                    [1]])
                if vehicle1 != vehicle2:
                    self.fleet.mergeVehicles(vehicle1, vehicle2,
                         0, self.cm, reverse=True)
                del vehicle1, vehicle2
```

A.2 ClosestNeighborInitializer.py

```
import numpy as np
import random
from copy import copy, deepcopy

class ClosestNeighborInitializer:
    def __init__(self, CustomerManager, Fleet):
        self.cm = deepcopy(CustomerManager)
        self.fleet = deepcopy(Fleet)

    def run(self, isPrint=False):

        n = self.cm.getCustomers()
        C = self.fleet.maxCapacity()

        customers = list(range(1,n))

        while(len(customers)>0):
            tempC = self.cm.findClosest(0, customers)
            self.fleet.assignVehicle(self.cm, [tempC])
            veh = self.fleet.getAssignedVehicletoCustomer(tempC)
            customers.remove(tempC)

            neigbList = customers.copy()
            while(len(neigbList)>0):
                neig = self.cm.findClosest(tempC, neigbList)
                if (self.fleet.vehicles[veh].getLoad()[0]+self.cm.
                    get_demands(neig)[0]<=self.fleet.maxCapacity()
                    [0]) and \
                    (self.fleet.vehicles[veh].getLoad()[1]+self.cm.
                        get_demands(neig)[1]<=self.fleet.
                        maxCapacity()[1]):
                    self.fleet.vehicles[veh].addCustomer(neig,
                        tempC ,self.cm, justAfter=True)
                    tempC = neig
                    customers.remove(tempC)

                neigbList.remove(neig)

        if isPrint:
            self.fleet.printVehicleRoutes()
        return self.fleet
```

A.3 ImprovementHeuristic.py

```
import random
import itertools
from copy import copy, deepcopy

class ImprovementHeuristic:
    def __init__(self, CustomerManagement, fleet):
        self.cm = deepcopy(CustomerManagement)
        self.fleet = deepcopy(fleet)


    def run(self, strategy='FirstImprovement', method='mix',
        stopping=15,isPrint=False):
        if strategy=='FirstImprovement':
            self.FirstImprovementStrategy(stopping=stopping,
                method=method)
        elif strategy=='BestImprovement':
            self.BestImprovementStrategy(stopping=stopping, method
                =method)
        else:
            raise ValueError("Unknown strategy")
        if isPrint:
            self.fleet.printVehicleRoutes()
        return self.fleet


    def FirstImprovementStrategy(self, method, stopping):
        curr_cost = self.fleet.totalCost()
        nonImprovedSteps = 0
        if method == 'mix':
            while(nonImprovedSteps < stopping):
                vehicles = self.fleet.vehicleList()
                strategy = random.randint(0,2)
                if strategy == 0:
                    tempVeh = random.choice(vehicles)
                    customerList = self.fleet.
                        getAssignedCustomerstoVehicle(tempVeh)
                    switchCombinations = list(itertools.
                        combinations(customerList,2))
                    random.shuffle(switchCombinations)

                    for couple in switchCombinations:
                        if self.fleet.EstimateIntraSwitchCost(
                            tempVeh, couple[0], couple[1], self.cm)<
                            curr_cost:
```

```
                        self.fleet.IntraSwitchCustomers(tempVeh,
                             couple[0], couple[1], self.cm)
                        curr_cost = self.fleet.totalCost()
                        nonImprovedSteps = 0
                        break
                nonImprovedSteps +=1
            elif strategy == 1:
                tempVeh1, tempVeh2 = random.sample(vehicles,2)
                customerList1 = self.fleet.
                    getAssignedCustomerstoVehicle(tempVeh1)
                customerList2 = self.fleet.
                    getAssignedCustomerstoVehicle(tempVeh2)
                switchCombinations = []
                for customer1 in customerList1:
                    for customer2 in customerList2:
                        switchCombinations.append([customer1,
                            customer2])
                random.shuffle(switchCombinations)
                for couple in switchCombinations:
                    if self.fleet.EstimateInterSwitchCost(
                        tempVeh1, tempVeh2, couple[0], couple
                        [1], self.cm)<curr_cost:
                        self.fleet.InterSwitchCustomers(tempVeh
                            1, tempVeh2, couple[0], couple[1],
                            self.cm)
                        curr_cost = self.fleet.totalCost()
                        nonImprovedSteps = 0
                        break
                nonImprovedSteps +=1
            else:
                selVeh = random.choice(vehicles)
                selCust = random.choice(self.fleet.
                    getAssignedCustomerstoVehicle(selVeh))
                random.shuffle(vehicles)
                for VehicleToAdd in vehicles:
                    if VehicleToAdd == selVeh:
                        continue
                    estimatedCost = self.fleet.estimateRemoveAdd
                        (selCust, VehicleToAdd, self.cm)
                    if estimatedCost < curr_cost:
                        self.fleet.RemoveAdd(selCust,
                            VehicleToAdd, self.cm)
                        curr_cost = self.fleet.totalCost()
                        nonImprovedSteps = 0
                        break
                nonImprovedSteps +=1
```

```
        elif method == 'addremove':
            while(nonImprovedSteps < stopping):
                vehicles = self.fleet.vehicleList()
                selVeh = random.choice(vehicles)
                selCust = random.choice(self.fleet.
                    getAssignedCustomerstoVehicle(selVeh))
                random.shuffle(vehicles)
                for VehicleToAdd in vehicles:
                    if VehicleToAdd == selVeh:
                        continue
                    estimatedCost = self.fleet.estimateRemoveAdd(
                        selCust, VehicleToAdd, self.cm)
                    if estimatedCost < curr_cost:
                        self.fleet.RemoveAdd(selCust, VehicleToAdd,
                            self.cm)
                        curr_cost = self.fleet.totalCost()
                        nonImprovedSteps = 0
                        break
                nonImprovedSteps +=1
        elif method == 'inter-switch':
            while(nonImprovedSteps < stopping):
                vehicles = self.fleet.vehicleList()
                tempVeh1, tempVeh2 = random.sample(vehicles,2)
                customerList1 = self.fleet.
                    getAssignedCustomerstoVehicle(tempVeh1)
                customerList2 = self.fleet.
                    getAssignedCustomerstoVehicle(tempVeh2)
                switchCombinations = []
                for customer1 in customerList1:
                    for customer2 in customerList2:
                        switchCombinations.append([customer1,
                            customer2])
                random.shuffle(switchCombinations)
                for couple in switchCombinations:
                    if self.fleet.EstimateInterSwitchCost(tempVeh1,
                         tempVeh2, couple[0], couple[1], self.cm)<
                        curr_cost:
                        self.fleet.InterSwitchCustomers(tempVeh1,
                            tempVeh2, couple[0], couple[1], self.cm)
                        curr_cost = self.fleet.totalCost()
                        nonImprovedSteps = 0
                        break
                nonImprovedSteps +=1
        elif method == 'intra-switch':
                vehicles = self.fleet.vehicleList()
                tempVeh = random.choice(vehicles)
```

```
            customerList = self.fleet.
                getAssignedCustomerstoVehicle(tempVeh)
            switchCombinations = list(itertools.combinations(
                customerList,2))
            random.shuffle(switchCombinations)

            for couple in switchCombinations:
                if self.fleet.EstimateIntraSwitchCost(tempVeh,
                    couple[0], couple[1], self.cm)<curr_cost:
                    self.fleet.IntraSwitchCustomers(tempVeh,
                        couple[0], couple[1], self.cm)
                    curr_cost = self.fleet.totalCost()
                    nonImprovedSteps = 0
                    break
            nonImprovedSteps +=1
    else:
        raise ValueError("Unknown method")




def BestImprovementStrategy(self, method, stopping):

    bestCost = self.fleet.totalCost()
    nonImprovedSteps = 0
    if method == 'mix':
        while(nonImprovedSteps < stopping):
            vehicles = self.fleet.vehicleList()
            bestCouple =None
            strategy = random.randint(0,2)
            if strategy == 0:
                tempVeh = random.choice(vehicles)
                customerList = self.fleet.
                    getAssignedCustomerstoVehicle(tempVeh)
                switchCombinations = list(itertools.
                    combinations(customerList,2))

                for couple in switchCombinations:
                    if self.fleet.EstimateIntraSwitchCost(
                        tempVeh, couple[0], couple[1], self.cm)<
                        bestCost:
                        bestCouple = couple
                        bestCost = self.fleet.
                            EstimateIntraSwitchCost(tempVeh,
```

```
                        couple[0], couple[1], self.cm)
                    nonImprovedSteps = 0

            if bestCouple is not None:
                self.fleet.IntraSwitchCustomers(tempVeh,
                    bestCouple[0], bestCouple[1], self.cm)
            else:
                nonImprovedSteps +=1
        elif strategy== 1:
            tempVeh1, tempVeh2 = random.sample(vehicles,2)
            customerList1 = self.fleet.
                getAssignedCustomerstoVehicle(tempVeh1)
            customerList2 = self.fleet.
                getAssignedCustomerstoVehicle(tempVeh2)
            switchCombinations = []
            for customer1 in customerList1:
                for customer2 in customerList2:
                    switchCombinations.append([customer1,
                        customer2])

            for couple in switchCombinations:
                if self.fleet.EstimateInterSwitchCost(
                    tempVeh1, tempVeh2, couple[0], couple
                    [1], self.cm)<bestCost:
                    bestCouple = couple
                    bestCost = self.fleet.
                        EstimateInterSwitchCost(tempVeh1,
                        tempVeh2, couple[0], couple[1], self
                        .cm)
                    nonImprovedSteps = 0

            if bestCouple is not None:
                self.fleet.InterSwitchCustomers(tempVeh1,
                    tempVeh2, bestCouple[0], bestCouple[1],
                    self.cm)
            else:
                nonImprovedSteps +=1
        else:
            selVeh = random.choice(vehicles)
            selCust = random.choice(self.fleet.
                getAssignedCustomerstoVehicle(selVeh))

            for VehicleToAdd in vehicles:
                if VehicleToAdd == selVeh:
                    continue
                estimatedCost = self.fleet.estimateRemoveAdd
```

```
                    (selCust, VehicleToAdd, self.cm)
                if estimatedCost < bestCost:
                    bestCouple = VehicleToAdd
                    bestCost = estimatedCost
                    nonImprovedSteps = 0

            if bestCouple is not None:
                self.fleet.RemoveAdd(selCust, bestCouple,
                    self.cm)
            else:
                nonImprovedSteps +=1
    elif method == 'addremove':
        while(nonImprovedSteps < stopping):
            vehicles = self.fleet.vehicleList()
            bestCouple =None
            selVeh = random.choice(vehicles)
            selCust = random.choice(self.fleet.
                getAssignedCustomerstoVehicle(selVeh))

            for VehicleToAdd in vehicles:
                if VehicleToAdd == selVeh:
                    continue
                estimatedCost = self.fleet.estimateRemoveAdd(
                    selCust, VehicleToAdd, self.cm)
                if estimatedCost < bestCost:
                    bestCouple = VehicleToAdd
                    bestCost = estimatedCost
                    nonImprovedSteps = 0

            if bestCouple is not None:
                self.fleet.RemoveAdd(selCust, bestCouple, self.
                    cm)
            else:
                nonImprovedSteps +=1
    elif method == 'inter-switch':
        while(nonImprovedSteps < stopping):
            vehicles = self.fleet.vehicleList()
            bestCouple =None
            tempVeh1, tempVeh2 = random.sample(vehicles,2)
            customerList1 = self.fleet.
                getAssignedCustomerstoVehicle(tempVeh1)
            customerList2 = self.fleet.
                getAssignedCustomerstoVehicle(tempVeh2)
            switchCombinations = []
            for customer1 in customerList1:
                for customer2 in customerList2:
```

```
                    switchCombinations.append([customer1,
                        customer2])

            for couple in switchCombinations:
                if self.fleet.EstimateInterSwitchCost(tempVeh1,
                     tempVeh2, couple[0], couple[1], self.cm)<
                    bestCost:
                    bestCouple = couple
                    bestCost = self.fleet.
                        EstimateInterSwitchCost(tempVeh1,
                        tempVeh2, couple[0], couple[1], self.cm)
                    nonImprovedSteps = 0

            if bestCouple is not None:
                self.fleet.InterSwitchCustomers(tempVeh1,
                    tempVeh2, bestCouple[0], bestCouple[1],
                    self.cm)
            else:
                nonImprovedSteps +=1
        elif method == 'intra-switch':
            while(nonImprovedSteps < stopping):
                vehicles = self.fleet.vehicleList()
                bestCouple =None
                tempVeh = random.choice(vehicles)
                customerList = self.fleet.
                    getAssignedCustomerstoVehicle(tempVeh)
                switchCombinations = list(itertools.combinations(
                    customerList,2))

                for couple in switchCombinations:
                    if self.fleet.EstimateIntraSwitchCost(tempVeh,
                        couple[0], couple[1], self.cm)<bestCost:
                        bestCouple = couple
                        bestCost = self.fleet.
                            EstimateIntraSwitchCost(tempVeh, couple
                            [0], couple[1], self.cm)
                        nonImprovedSteps = 0

                if bestCouple is not None:
                    self.fleet.IntraSwitchCustomers(tempVeh,
                        bestCouple[0], bestCouple[1], self.cm)
                else:
                    nonImprovedSteps +=1
```

A.4 SimulatedAnnealing.py

```
from copy import copy, deepcopy
import math
import random

class SimulatedAnnealing:
    def __init__(self, CustomerManagement, fleet):
        self.cm = deepcopy(CustomerManagement)
        self.fleet = deepcopy(fleet)

    def run(self, method='intra-switch', initialTemp=1000, minTemp
        =0.1, coolingP=0.5, maxEpoch=2, isPrint=False):


        bestCost = self.fleet.totalCost()
        initialTemp = initialTemp
        minTemp = minTemp
        coolingP = coolingP

        currentTemp = initialTemp

        if method == 'mix':
            while(currentTemp>minTemp):
                epoch=0
                while(epoch<=maxEpoch):
                    vehicles = self.fleet.vehicleList()
                    strategy = random.randint(0,2)
                    if strategy == 0:
                        tempVeh = random.choice(vehicles)
                        customerList = self.fleet.
                            getAssignedCustomerstoVehicle(tempVeh)
                        if len(customerList)>=2:
                            customer1, customer2 = random.sample(
                                customerList,2)
                        else:
                            continue

                        estimatedCost = self.fleet.
                            EstimateIntraSwitchCost(tempVeh,
                            customer1, customer2, self.cm)
                        if estimatedCost < bestCost:
                            self.fleet.IntraSwitchCustomers(tempVeh,
                                 customer1, customer2, self.cm)
                            bestCost = self.fleet.totalCost()
                        else:
```

```
                        acceptProb = (bestCost-estimatedCost)/
                            currentTemp
                        acceptProb = math.exp(acceptProb)
                        randVar = random.random()
                        if (acceptProb>randVar):
                            self.fleet.IntraSwitchCustomers(
                                tempVeh, customer1, customer2,
                                self.cm)
                            bestCost = self.fleet.totalCost()
                elif strategy == 1:
                    vehicles = self.fleet.vehicleList()
                    tempVeh1, tempVeh2 = random.sample(vehicles
                        ,2)
                    customer1 = random.choice(self.fleet.
                        getAssignedCustomerstoVehicle(tempVeh1))
                    customer2 = random.choice(self.fleet.
                        getAssignedCustomerstoVehicle(tempVeh2))

                    estimatedCost = self.fleet.
                        EstimateInterSwitchCost(tempVeh1,
                        tempVeh2, customer1, customer2, self.cm)
                    if estimatedCost < bestCost:
                        self.fleet.InterSwitchCustomers(tempVeh
                            1, tempVeh2, customer1, customer2,
                            self.cm)
                        bestCost = self.fleet.totalCost()
                    else:
                        acceptProb = (bestCost-estimatedCost)/
                            currentTemp
                        acceptProb = math.exp(acceptProb)
                        randVar = random.random()
                        if (acceptProb>randVar):
                            self.fleet.InterSwitchCustomers(
                                tempVeh1, tempVeh2, customer1,
                                customer2, self.cm)
                            bestCost = self.fleet.totalCost()
                else:
                    vehicles = self.fleet.vehicleList()
                    selVeh = random.choice(vehicles)
                    selCust = random.choice(self.fleet.
                        getAssignedCustomerstoVehicle(selVeh))
                    VehicleToAdd = random.choice(vehicles)

                    estimatedCost = self.fleet.estimateRemoveAdd
                        (selCust, VehicleToAdd, self.cm)
```

```
                    if estimatedCost < bestCost:
                        self.fleet.RemoveAdd(selCust,
                            VehicleToAdd, self.cm)
                        bestCost = self.fleet.totalCost()
                    else:
                        acceptProb = (bestCost-estimatedCost)/
                            currentTemp
                        acceptProb = math.exp(acceptProb)
                        randVar = random.random()
                        if (acceptProb>randVar):
                            self.fleet.RemoveAdd(selCust,
                                VehicleToAdd, self.cm)
                            bestCost = self.fleet.totalCost()
                epoch += 1
            currentTemp = currentTemp * coolingP
    elif method == 'addremove':
        while(currentTemp>minTemp):
            epoch=0
            while(epoch<=maxEpoch):
                vehicles = self.fleet.vehicleList()
                selVeh = random.choice(vehicles)
                selCust = random.choice(self.fleet.
                    getAssignedCustomerstoVehicle(selVeh))
                VehicleToAdd = random.choice(vehicles)

                estimatedCost = self.fleet.estimateRemoveAdd(
                    selCust, VehicleToAdd, self.cm)

                if estimatedCost < bestCost:
                    self.fleet.RemoveAdd(selCust, VehicleToAdd,
                        self.cm)
                    bestCost = self.fleet.totalCost()
                else:
                    acceptProb = (bestCost-estimatedCost)/
                        currentTemp
                    acceptProb = math.exp(acceptProb)
                    randVar = random.random()
                    if (acceptProb>randVar):
                        self.fleet.RemoveAdd(selCust,
                            VehicleToAdd, self.cm)
                        bestCost = self.fleet.totalCost()
                epoch += 1
            currentTemp = currentTemp * coolingP
    elif method == 'inter-switch':
        while(currentTemp>minTemp):
            epoch=0
```

```
            while(epoch<=maxEpoch):
                vehicles = self.fleet.vehicleList()
                tempVeh1, tempVeh2 = random.sample(vehicles,2)
                customer1 = random.choice(self.fleet.
                    getAssignedCustomerstoVehicle(tempVeh1))
                customer2 = random.choice(self.fleet.
                    getAssignedCustomerstoVehicle(tempVeh2))

                estimatedCost = self.fleet.
                    EstimateInterSwitchCost(tempVeh1, tempVeh2,
                     customer1, customer2, self.cm)
                if estimatedCost < bestCost:
                    self.fleet.InterSwitchCustomers(tempVeh1,
                        tempVeh2, customer1, customer2, self.cm)
                    bestCost = self.fleet.totalCost()
                else:
                    acceptProb = (bestCost-estimatedCost)/
                        currentTemp
                    acceptProb = math.exp(acceptProb)
                    randVar = random.random()
                    if (acceptProb>randVar):
                        self.fleet.InterSwitchCustomers(tempVeh
                            1, tempVeh2, customer1, customer2,
                            self.cm)
                        bestCost = self.fleet.totalCost()
                epoch += 1
            currentTemp = currentTemp * coolingP
    elif method == 'intra-switch':
        while(currentTemp>minTemp):
            epoch=0
            while(epoch<=maxEpoch):
                vehicles = self.fleet.vehicleList()
                tempVeh = random.choice(vehicles)
                customerList = self.fleet.
                    getAssignedCustomerstoVehicle(tempVeh)
                if len(customerList) >= 2:
                    customer1, customer2 = random.sample(
                        customerList,2)
                else:
                    continue

                estimatedCost = self.fleet.
                    EstimateIntraSwitchCost(tempVeh, customer1,
                     customer2, self.cm)
                if estimatedCost < bestCost:
                    self.fleet.IntraSwitchCustomers(tempVeh,
```

```
                        customer1, customer2, self.cm)
                    bestCost = self.fleet.totalCost()
                else:
                    acceptProb = (bestCost-estimatedCost)/
                        currentTemp
                    acceptProb = math.exp(acceptProb)
                    randVar = random.random()
                    if (acceptProb>randVar):
                        self.fleet.IntraSwitchCustomers(tempVeh,
                            customer1, customer2, self.cm)
                        bestCost = self.fleet.totalCost()
                epoch += 1
            currentTemp = currentTemp * coolingP



    if isPrint:
        self.fleet.printVehicleRoutes()
    return self.fleet
```

A.5 AntColony.py

```
import numpy as np
import random
from copy import copy, deepcopy
import math


class AntColony:
    def __init__(self, CustomerManager, Fleet):
        self.cm = deepcopy(CustomerManager)
        self.fleet = deepcopy(Fleet)
        self.pheromones = dict()

    def run(self, nAnt, evaporation=0.8, stop=20, alpha=1, beta=0,
         fitness='g', isPrint=False):
        n = self.cm.getCustomers()
        C1, C2 = self.fleet.maxCapacity()
        bestCost = math.inf

        if fitness=='g':
            FitnessFunc = self.GeometricFitness
        elif fitness=='a':
            FitnessFunc = self.ArithmeticFitness
        else:
            raise("Unknown fitness. Fitness can be 'g' or 'a'.")

        customers = list(range(1,n))
        allNodes = [0] + customers

        arcs = [(i, j) for i in allNodes for j in allNodes if i !=
             j]



        for arc in arcs:
            self.pheromones[arc] = 1

        population = [deepcopy(self.fleet) for i in range(nAnt)]
        nonImp=0

        while(nonImp<stop):
            improved=False
            for ant in range(nAnt):
                remainings = customers.copy()
```

```
            route = [0]

            while(len(remainings)>0):
                if route[-1] == 0:
                    curNode = random.choices(remainings, weights
                        =[FitnessFunc(0,j,alpha,beta) for j in
                        remainings])[0]
                    route.append(curNode)
                    remainings.remove(curNode)
                    population[ant].assignVehicle(self.cm, [
                        curNode])
                    veh = population[ant].
                        getAssignedVehicletoCustomer(curNode)
                    curC1 = C1 - self.cm.get_demands(curNode)[0]
                    curC2 = C2 - self.cm.get_demands(curNode)[1]
                else:
                    S = self.cm.getFeasibleSpace(curC1, curC2,
                        [0]+remainings)
                    tmpNode = random.choices(S, weights=[
                        FitnessFunc(curNode,j,alpha,beta) for j
                        in S])[0]
                    if tmpNode!=0:
                        population[ant].vehicles[veh].
                            addCustomer(tmpNode, curNode ,self.
                            cm, justAfter=True)
                        remainings.remove(tmpNode)
                    curNode = tmpNode
                    route.append(curNode)
                    curC1 = curC1 - self.cm.get_demands(curNode)
                        [0]
                    curC2 = curC2 - self.cm.get_demands(curNode)
                        [1]


            tmpCost = population[ant].totalCost()
            if bestCost > tmpCost:
                self.fleet = deepcopy(population[ant])
                bestCost = self.fleet.totalCost()
                improved=True


        for i in range(len(route)-1):
            self.pheromones[route[i],route[i+1]] += 1 / math.
                log(tmpCost, n)

        self.pheromones.update({k: evaporation*v for k, v in
```

```
            self.pheromones.items()})

        if improved is False:
            nonImp += 1


    if isPrint:
        self.fleet.printVehicleRoutes()
    return self.fleet

def ArithmeticFitness(self, node1, node2, pAlpha, pBeta):
    return self.pheromones[node1, node2]*pAlpha+(1/self.cm.get
        _arc(node1,node2))*pBeta

def GeometricFitness(self, node1, node2, pAlpha, pBeta):
    return self.pheromones[node1, node2]**pAlpha*(1/self.cm.
        get_arc(node1,node2))**pBeta
```

A.6 AntColony_v2.py

```
import numpy as np
import random
from copy import copy, deepcopy
import math


class AntColony_v2:
    def __init__(self, CustomerManager, Fleet):
        self.cm = deepcopy(CustomerManager)
        self.fleet = deepcopy(Fleet)
        self.pheromones = dict()

    def run(self, nAnt, evaporation=0.8, stop=20, alpha=1, beta=0,
         fitness='g', isPrint=False):

        n = self.cm.getCustomers()
        C1, C2 = self.fleet.maxCapacity()
        bestCost = math.inf

        if fitness=='g':
            FitnessFunc = self.GeometricFitness
        elif fitness=='a':
            FitnessFunc = self.ArithmeticFitness
        else:
            raise("Unknown fitness. Fitness can be 'g' or 'a'.")

        customers = list(range(1,n))
        allNodes = [0] + customers

        Nodes = [(i,j) for i in allNodes for j in allNodes if i!=j
            ]
        arcs = [(i, j) for i in Nodes for j in Nodes if i != j and
             i[1]==j[0]]



        for arc in arcs:
            self.pheromones[arc] = 1

        population = [deepcopy(self.fleet) for i in range(nAnt)]
        nonImp=0

        while(nonImp<stop):
            improved=False
```

```
            for ant in range(nAnt):
                remainings = customers.copy()
                curNode = random.choice([i for i in Nodes if i
                    [0]==0])

                route = [curNode[0], curNode[1]]
                remainings.remove(curNode[1])
                population[ant].assignVehicle(self.cm, [curNode
                    [1]])
                veh = population[ant].getAssignedVehicletoCustomer(
                    curNode[1])
                curC1 = C1 - self.cm.get_demands(curNode[1])[0]
                curC2 = C2 - self.cm.get_demands(curNode[1])[1]

                while(len(remainings)>0):
                    if route[-1] == 0:
                        S = [i for i in Nodes if i[0]==0 and i[1] in
                             remainings]
                        tmpNode = random.choices(S, weights=[
                            FitnessFunc(curNode,j,alpha,beta) for j
                            in S])[0]
                        curNode = tmpNode
                        route.append(curNode[1])
                        remainings.remove(curNode[1])
                        population[ant].assignVehicle(self.cm, [
                            curNode[1]])
                        veh = population[ant].
                            getAssignedVehicletoCustomer(curNode[1])
                        curC1 = C1 - self.cm.get_demands(curNode[1])
                            [0]
                        curC2 = C2 - self.cm.get_demands(curNode[1])
                            [1]
                    else:
                        eligibles = self.cm.getFeasibleSpace(curC1,
                            curC2, [0]+remainings)
                        S = [i for i in Nodes if i[0]==curNode[1]
                            and i[1] in eligibles]
                        tmpNode = random.choices(S, weights=[
                            FitnessFunc(curNode,j,alpha,beta) for j
                            in S])[0]
                        if tmpNode[1]!=0:
                            population[ant].vehicles[veh].
                                addCustomer(tmpNode[1], tmpNode[0] ,
                                self.cm, justAfter=True)
                            remainings.remove(tmpNode[1])
                        curNode = tmpNode
```

```
                    route.append(curNode[1])
                    curC1 = curC1 - self.cm.get_demands(curNode
                        [1])[0]
                    curC2 = curC2 - self.cm.get_demands(curNode
                        [1])[1]


            tmpCost = population[ant].totalCost()
            if bestCost > tmpCost:
                self.fleet = deepcopy(population[ant])
                bestCost = self.fleet.totalCost()
                improved=True


        for i in range(len(route)-2):
            self.pheromones[(route[i],route[i+1]), (route[i+1],
                route[i+2])] += 1 / math.log(tmpCost, n)

        self.pheromones.update({k: evaporation*v for k, v in
            self.pheromones.items()})

        if improved is False:
            nonImp += 1

    if isPrint:
        self.fleet.printVehicleRoutes()
    return self.fleet

def ArithmeticFitness(self, node1, node2, pAlpha, pBeta):
    return self.pheromones[node1, node2]*pAlpha+(1/self.cm.get
        _arc(node2[0],node2[1]))*pBeta

def GeometricFitness(self, node1, node2, pAlpha, pBeta):
    return self.pheromones[node1, node2]**pAlpha*(1/self.cm.
        get_arc(node2[0],node2[1]))**pBeta
```

A.7 DecomposedGenetic.py

```
import numpy as np
from copy import copy, deepcopy
import random

class DecomposedGenetic:
    def __init__(self, CustomerManager, Fleet):
        self.cm = deepcopy(CustomerManager)
        self.fleet = deepcopy(Fleet)
        self.clusters =dict()
        self.totalClusterDemands = dict()
        self.population = dict()
        self.fitnessValues = dict()

    def run(self, n, tournamentSize=3, eliteSize=1, stopCon = 50,
        isPrint=False):
        self.BuildClusters(n,tournamentSize, eliteSize, stopCon)
        self.BuildRoutes()
        if isPrint:
            self.printClusters()
            self.fleet.printVehicleRoutes()
        return self.fleet


    def BuildClusters(self, n, tournamentSize, eliteSize, stopCon)
        :

        self.population = self.GeneratePopulation(n=n)
        self.fitnessValues = dict()
        bestValue = 0
        nonImp = 0
        while(nonImp<stopCon):
            for key, value in self.population.items():
                self.fitnessValues[key] = self.FitnessFunction(
                    value)


            newGeneration = dict()
            sortedList = sorted(self.fitnessValues.keys(), key =
                lambda x:self.fitnessValues[x], reverse = True)
            for c in range(eliteSize):
                newGeneration[c] = self.population[sortedList[c]]

            if bestValue < self.fitnessValues[sortedList[0]]:
                bestValue = self.fitnessValues[sortedList[0]]
```

```
            nonImp = 0


        for c in range(eliteSize,n):

            child = self.Tournament(random.sample(list(self.
                population.keys()), tournamentSize))
            child = self.Mutation(child)
            newGeneration[c] = child

        self.population = newGeneration
        nonImp += 1

        for key, value in self.population.items():
            self.fitnessValues[key] = self.FitnessFunction(
                value)

        bestSolInd = sorted(self.fitnessValues.keys(), key =
            lambda x:self.fitnessValues[x], reverse = True)[0]
        bestSol = self.population[bestSolInd]

        for k in list(set(bestSol)):
            tW=0
            tV=0
            itemsIn = np.where(np.isin(bestSol,k))[0]
            for i in itemsIn:
                tW += self.cm.get_demands(i+1)[0]
                tV += self.cm.get_demands(i+1)[1]
            self.totalClusterDemands[k] = [tW, tV]


        for cust in range(len(bestSol)):
            self.clusters[cust+1] = bestSol[cust]

def GeneratePopulation(self, n):

    population = dict()
    customers = [i for i in range(1, self.cm.getCustomers())]
    W = self.fleet.weightCapacities()
    V = self.fleet.volumeCapacities()

    for i in range(n):
        vehicleWeights = [0 for k in customers]
        vehicleVolumes =[0 for k in customers]
        population[i] = [0 for k in customers]
        remainings = customers.copy()
```

```
        while(len(remainings)>0):
            tmpC = random.choice(remainings)
            remainings.remove(tmpC)
            vehList = list(range(1, max(population[i])+1))
            np.random.shuffle(vehList)
            vehList.append(max(population[i])+1)
            for j in vehList:
                if (vehicleWeights[j-1]+self.cm.get_demands(
                    tmpC)[0]<=W[-1]) \
                and (vehicleVolumes[j-1]+self.cm.get_demands(
                    tmpC)[1]<=V[-1]):
                    population[i][tmpC-1]=j
                    vehicleWeights[j-1] += self.cm.get_demands(
                        tmpC)[0]
                    vehicleVolumes[j-1] += self.cm.get_demands(
                        tmpC)[1]
                    break

    return population

def MaliyetHesapla(self, array):
    vehicleCount = max(array)
    fitnessValue = 0
    W = self.fleet.weightCapacities()
    V = self.fleet.volumeCapacities()
    F = self.fleet.FixedCosts()
    minItemCount = len(array)
    for j in range(1,vehicleCount+1):
        tW = 0
        tV = 0
        itemsIn = np.where(np.isin(array,j))[0]
        for i in itemsIn:
            tW += self.cm.get_demands(i+1)[0]
            tV += self.cm.get_demands(i+1)[1]

        for k in range(len(W)):
            if tW<=W[k] and tV<=V[k]:
                fitnessValue += F[k]
                break


    return fitnessValue
```

```
def FitnessFunction(self, array):
    vehicleCount = max(array)
    fitnessValue = 0
    W = self.fleet.weightCapacities()
    V = self.fleet.volumeCapacities()
    F = self.fleet.FixedCosts()
    minItemCount = len(array)
    for j in range(1,vehicleCount+1):
        tW = 0
        tV = 0
        itemsIn = np.where(np.isin(array,j))[0]
        for i in itemsIn:
            tW += self.cm.get_demands(i+1)[0]
            tV += self.cm.get_demands(i+1)[1]

        if len(itemsIn)< minItemCount:
            minItemCount=len(itemsIn)
            count = 1
        elif len(itemsIn) == minItemCount:
            count += 1

        for k in range(len(W)):
            if tW<=W[k] and tV<=V[k]:
                fitnessValue += 1/np.log(F[k])
                break

    fitnessValue += np.exp(-1*(minItemCount-0.1*count))

    return fitnessValue

def Tournament(self, array):
    ReqFits = { k: self.fitnessValues[k] for k in array }
    ind = max(ReqFits, key=ReqFits.get)
    return self.population[ind].copy()

'''
def CrossOver(self, parent1, parent2):
    N1 = max(parent1)

    tmpN = np.ceil(N1/2).astype(int)

    child = [0 for i in range(1, self.cm.getCustomers())]

    for k in range(tmpN):

'''
```

```
def Mutation(self, chromosome):
    customers = [i for i in range(1, self.cm.getCustomers())]
    vehicles = [i for i in range(1, max(chromosome)+1)]
    NewVeh = random.choice(vehicles)
    try:
        oldVeh = random.choices(vehicles, weights=[1/
            chromosome.count(j) for j in vehicles])[0]
    except:
        print(vehicles, chromosome)
    if NewVeh == oldVeh:
        return chromosome
    cust = random.choice(list(np.where(np.isin(chromosome,
        oldVeh)))[0])

    tW = 0
    tV = 0
    for i in np.where(np.isin(chromosome,NewVeh))[0]:
        tW += self.cm.get_demands(i+1)[0]
        tV += self.cm.get_demands(i+1)[1]

    if tW+self.cm.get_demands(cust)[0]<= self.fleet.
        maxCapacity()[0] \
    and tV+self.cm.get_demands(cust)[1]<= self.fleet.
        maxCapacity()[1]:
        chromosome[cust] = NewVeh

    if len(list(np.where(np.isin(chromosome,oldVeh)))[0]) ==
        0:
        chromosome = [i-1 if i>=oldVeh else i for i in
            chromosome]

    return chromosome


def printClusters(self):
    for key in self.totalClusterDemands:
        nodes = self.getCluster(key)
        print("Cluster #" +str(key) + " has " + str(self.
            totalClusterDemands[key][0]) + \
              " kg and " + str(self.totalClusterDemands[key
                  ][1]) + " liters of total demands, and for
                  customers: " \
              + str(nodes))

def getCluster(self, cluster):
```

```
        nodes = []
        for key2 in self.clusters:
            if(self.clusters[key2]==cluster):
                nodes.append(key2)
        return nodes


    def BuildRoutes(self):
        for key, demand in self.totalClusterDemands.items():
            customers = self.getCluster(key)

            for customer in customers:
                self.fleet.assignVehicle(self.cm, [customer])

            arcs = self.cm.sortArcs(customers)

            for arc in arcs:
                if self.fleet.isLast(customers[arc[0]]) & self.
                    fleet.isFirst(customers[arc[1]]):
                    vehicle1 = self.fleet.
                        getAssignedVehicletoCustomer(customers[arc
                        [0]])
                    vehicle2 = self.fleet.
                        getAssignedVehicletoCustomer(customers[arc
                        [1]])
                    if vehicle1 != vehicle2:
                        self.fleet.mergeVehicles(vehicle1, vehicle2,
                             customers[arc[0]], self.cm, reverse=
                            False)
                    del vehicle1, vehicle2
                elif self.fleet.isLast(customers[arc[1]]) & self.
                    fleet.isFirst(customers[arc[0]]):
                    vehicle1 = self.fleet.
                        getAssignedVehicletoCustomer(customers[arc
                        [1]])
                    vehicle2 = self.fleet.
                        getAssignedVehicletoCustomer(customers[arc
                        [0]])
                    if vehicle1 != vehicle2:
                        self.fleet.mergeVehicles(vehicle1, vehicle2,
                             customers[arc[1]], self.cm, reverse=
                            False)
                    del vehicle1, vehicle2
                elif self.fleet.isLast(customers[arc[0]]) & self.
                    fleet.isLast(customers[arc[1]]):
                    vehicle1 = self.fleet.
```

```
                    getAssignedVehicletoCustomer(customers[arc
                    [0]])
                vehicle2 = self.fleet.
                    getAssignedVehicletoCustomer(customers[arc
                    [1]])
                if vehicle1 != vehicle2:
                    self.fleet.mergeVehicles(vehicle1, vehicle2,
                         customers[arc[0]], self.cm, reverse=
                        True)
                del vehicle1, vehicle2
            elif self.fleet.isFirst(customers[arc[0]]) & self.
                fleet.isFirst(customers[arc[1]]):
                vehicle1 = self.fleet.
                    getAssignedVehicletoCustomer(customers[arc
                    [0]])
                vehicle2 = self.fleet.
                    getAssignedVehicletoCustomer(customers[arc
                    [1]])
                if vehicle1 != vehicle2:
                    self.fleet.mergeVehicles(vehicle1, vehicle2,
                         0, self.cm, reverse=True)
                del vehicle1, vehicle2
```

B Experiments

B.1 Cplex Solver

Sample	Cost	Time (secs)
N=20,fleet#1, sample#1	4,066.34	1000
N=20,fleet#1, sample#2	5,083.60	1000
N=20,fleet#1, sample#3	5,086.21	1000
N=20,fleet#2, sample#1	4,070.21	1000
N=20,fleet#2, sample#2	4,614.78	1000
N=20,fleet#2, sample#3	4,887.96	1000
N=20,fleet#3, sample#1	4,222.47	1000
N=20,fleet#3, sample#2	5,433.60	1000
N=20,fleet#3, sample#3	5,431.07	1000
N=50,fleet#1, sample#1	13,536.10	1000
N=50,fleet#1, sample#2	12,751.50	1000
N=50,fleet#1, sample#3	12,833.70	1000
N=50,fleet#2, sample#1	12,953.30	1000
N=50,fleet#2, sample#2	13,118.40	1000
N=50,fleet#2, sample#3	12,860.50	1000
N=50,fleet#3, sample#1	14,313.00	1000
N=50,fleet#3, sample#2	14,128.80	1000
N=50,fleet#3, sample#3	13,541.00	1000

B.2 Cluster-first-route-second

Sample (5 runs/each)	Average Time	Cost	Benchmark gap
N=20,fleet#1, sample#1	0.4826	5864.60	44.22%
N=20,fleet#1, sample#2	0.4682	5395.00	6.13%
N=20,fleet#1, sample#3	0.4993	5338.00	4.95%
N=20,fleet#2, sample#1	0.4878	5367.20	31.87%
N=20,fleet#2, sample#2	0.4864	5656.40	22.57%
N=20,fleet#2, sample#3	0.4898	5650.40	15.60%
N=20,fleet#3, sample#1	0.4731	6164.60	46.00%
N=20,fleet#3, sample#2	0.4880	5895.00	8.49%
N=20,fleet#3, sample#3	0.5031	5838.00	7.49%
N=50,fleet#1, sample#1	112.6255	15000.60	10.82%
N=50,fleet#1, sample#2	105.8944	14952.00	17.26%
N=50,fleet#1, sample#3	106.5856	14700.00	14.54%
N=50,fleet#2, sample#1	106.8393	15319.00	18.26%
N=50,fleet#2, sample#2	107.0264	15555.20	18.58%
N=50,fleet#2, sample#3	107.1966	15365.80	19.48%
N=50,fleet#3, sample#1	106.9952	15900.60	11.09%
N=50,fleet#3, sample#2	106.7409	16352.00	15.74%
N=50,fleet#3, sample#3	106.7802	15950.00	17.79%

B.3 Closest Neighbor Heuristic

sample (5 runs/each)	Average Time	Cost	Benchmark gap
N=20,fleet#1, sample#1	0.0553	4363	7.30%
N=20,fleet#1, sample#2	0.0415	5162.2	1.55%
N=20,fleet#1, sample#3	0.0529	5395	6.07%
N=20,fleet#2, sample#1	0.0636	4242.8	4.24%
N=20,fleet#2, sample#2	0.0529	5287.6	14.58%
N=20,fleet#2, sample#3	0.0686	5269.2	7.80%
N=20,fleet#3, sample#1	0.0537	4763	12.80%
N=20,fleet#3, sample#2	0.0414	5512.2	1.45%
N=20,fleet#3, sample#3	0.053	5895	8.54%
N=50,fleet#1, sample#1	0.1956	13871.4	2.48%
N=50,fleet#1, sample#2	0.203	12998	1.93%
N=50,fleet#1, sample#3	0.1869	12999	1.29%
N=50,fleet#2, sample#1	0.2387	13325.8	2.88%
N=50,fleet#2, sample#2	0.2306	12610.8	-3.87%
N=50,fleet#2, sample#3	0.2292	12567.4	-2.28%
N=50,fleet#3, sample#1	0.1938	15021.4	4.95%
N=50,fleet#3, sample#2	0.204	14198	0.49%
N=50,fleet#3, sample#3	0.1906	14199	4.86%

B.4 First Improvement Heuristic

Table B4.1: Sample: N=20,fleet#1, sample#1 over 5 runs

initial	*stopping*	*method*	mean(time)	mean(cost)	min(cost)	mean(gap)	best(gap)
Alg.1a	*15*	*mix*	0.4943	4894.51	4830.4	20.37%	18.79%
Alg.1a	*15*	*add/remove*	0.2674	4689.64	4320.2	15.33%	6.24%
Alg.1a	*15*	*Inter-switch*	1.0427	4961.84	4841.6	22.02%	19.07%
Alg.1a	*15*	*Intra-switch*	0.0005	5863.8	5861.6	44.20%	44.15%
Alg.1a	*30*	*mix*	1.0325	4889.32	4343.6	20.24%	6.82%
Alg.1a	*30*	*add/remove*	0.3001	4589.8	4329.6	12.87%	6.47%
Alg.1a	*30*	*Inter-switch*	1.7851	5045.36	4822.4	24.08%	18.59%
Alg.1a	*30*	*Intra-switch*	0.0004	5864.4	5863.6	44.22%	44.20%
Alg.1b	*15*	*mix*	0.4014	4292.4	4098	5.56%	0.78%
Alg.1b	*15*	*add/remove*	0.0668	4358.4	4357	7.18%	7.15%
Alg.1b	*15*	*Inter-switch*	1.0817	4336.4	4317	6.64%	6.16%
Alg.1b	*15*	*Intra-switch*	0.0006	4363	4363	7.30%	7.30%
Alg.1b	*30*	*mix*	1.3396	4275.28	4090.4	5.14%	0.59%
Alg.1b	*30*	*add/remove*	0.1444	4351.2	4342	7.01%	6.78%
Alg.1b	*30*	*Inter-switch*	1.96	4336.2	4332	6.64%	6.53%
Alg.1b	*30*	*Intra-switch*	0.0007	4363	4363	7.30%	7.30%

Table B4.2: Sample: N=20,fleet#2, sample#1 over 5 runs

initial	*stopping*	*method*	mean(time)	mean(cost)	min(cost)	mean(gap)	best(gap)
Alg. 1a	*15*	*mix*	1.2876	4139.00	3648.40	1.69%	-10.36%
Alg. 1a	*15*	*add/remove*	0.2944	4190.28	3823.00	2.95%	-6.07%
Alg. 1a	*15*	*Inter-switch*	2.0271	4602.44	4576.20	13.08%	12.43%
Alg. 1a	*15*	*Intra-switch*	0.0006	5367.20	5367.20	31.87%	31.87%
Alg. 1a	*30*	*mix*	2.0347	4179.96	3825.80	2.70%	-6.00%
Alg. 1a	*30*	*add/remove*	0.3812	4205.20	3835.20	3.32%	-5.77%
Alg. 1a	*30*	*Inter-switch*	2.5828	4400.32	4328.20	8.11%	6.34%
Alg. 1a	*30*	*Intra-switch*	0.0004	5367.00	5366.20	31.86%	31.84%
Alg. 1b	*15*	*mix*	2.233	3920.44	3885.00	-3.68%	-4.55%
Alg. 1b	*15*	*add/remove*	0.0554	4006.04	3987.80	-1.58%	-2.02%
Alg. 1b	*15*	*Inter-switch*	2.6058	3926.72	3890.80	-3.53%	-4.41%
Alg. 1b	*15*	*Intra-switch*	0.0006	4241.12	4240.00	4.20%	4.17%
Alg. 1b	*30*	*mix*	2.4511	3880.80	3855.40	-4.65%	-5.28%
Alg. 1b	*30*	*add/remove*	0.108	4012.12	3987.80	-1.43%	-2.02%
Alg. 1b	*30*	*Inter-switch*	4.4044	3958.96	3918.40	-2.73%	-3.73%
Alg. 1b	*30*	*Intra-switch*	0.0006	4241.40	4240.00	4.21%	4.17%

Table B4.3: Sample: N=50,fleet#2, sample#1 over 5 runs

initial	*stopping*	*method*	mean(time)	mean(cost)	min(cost)	mean(gap)	best(gap)
Alg. 1a	*15*	*mix*	2.5332	14299.00	14059.20	10.39%	8.54%
Alg. 1a	*15*	*add/remove*	1.0869	14321.68	14004.20	10.56%	8.11%
Alg. 1a	*15*	*Inter-switch*	3.2086	14396.44	14104.60	11.14%	8.89%
Alg. 1a	*15*	*Intra-switch*	0.0009	15316.76	15310.60	18.25%	18.20%
Alg. 1a	*30*	*mix*	4.2318	14141.92	13547.80	9.18%	4.59%
Alg. 1a	*30*	*add/remove*	2.1097	14199.96	13813.80	9.62%	6.64%
Alg. 1a	*30*	*Inter-switch*	5.6822	14292.64	14140.80	10.34%	9.17%
Alg. 1a	*30*	*Intra-switch*	0.0007	15316.76	15310.60	18.25%	18.20%
Alg. 1b	*15*	*mix*	1.5088	12945.4	12656.6	-0.06%	-2.29%
Alg. 1b	*15*	*add/remove*	0.4015	12983.16	12793.4	0.23%	-1.23%
Alg. 1b	*15*	*Inter-switch*	1.1265	13325.8	13325.8	2.88%	2.88%
Alg. 1b	*15*	*Intra-switch*	0.0008	13323	13318.8	2.85%	2.82%
Alg. 1b	*30*	*mix*	5.5605	12759.2	12662.8	-1.50%	-2.24%
Alg. 1b	*30*	*add/remove*	1.1424	13009.96	12945.6	0.44%	-0.06%
Alg. 1b	*30*	*Inter-switch*	2.4393	13325.8	13325.8	2.88%	2.88%
Alg. 1b	*30*	*Intra-switch*	0.0008	13325.8	13325.8	2.88%	2.88%

B.5 Best Improvement Heuristic

Table B5.1: Sample: N=20, fleet#1, sample#1 over 5 runs

initial	*stopping*	*method*	mean(time)	mean(cost)	min(cost)	mean(gap)	best(gap)
Alg. 1a	*15*	*mix*	0.6482	4781.64	4604.20	17.59%	13.23%
Alg. 1a	*15*	*add/remove*	0.1982	4438.40	4320.60	9.15%	6.25%
Alg. 1a	*15*	*Inter-switch*	1.1773	5167.24	5089.20	27.07%	25.15%
Alg. 1a	*15*	*Intra-switch*	0.0053	5860.60	5860.60	44.12%	44.12%
Alg. 1a	*30*	*mix*	1.1225	4639.84	4315.40	14.10%	6.12%
Alg. 1a	*30*	*add/remove*	0.3523	4675.28	4325.00	14.98%	6.36%
Alg. 1a	*30*	*Inter-switch*	1.5816	5101.00	5089.20	25.44%	25.15%
Alg. 1a	*30*	*Intra-switch*	0.0059	5860.60	5860.60	44.12%	44.12%
Alg. 1b	*15*	*mix*	0.7519	4280.2	4098	5.26%	0.78%
Alg. 1b	*15*	*add/remove*	0.0727	4357.6	4357	7.16%	7.15%
Alg. 1b	*15*	*Inter-switch*	1.2681	4341.8	4341	6.77%	6.75%
Alg. 1b	*15*	*Intra-switch*	0.006	4363	4363	7.30%	7.30%
Alg. 1b	*30*	*mix*	1.2033	4227.6	4098	3.97%	0.78%
Alg. 1b	*30*	*add/remove*	0.1387	4350	4342	6.98%	6.78%
Alg. 1b	*30*	*Inter-switch*	2.0967	4341.2	4341	6.76%	6.75%
Alg. 1b	*30*	*Intra-switch*	0.0121	4363	4363	7.30%	7.30%

Table B5.2: Sample: N=20, fleet#2, sample#1 over 5 runs

initial	*stopping*	*method*	mean(time)	mean(cost)	min(cost)	mean(gap)	best(gap)
Alg. 1a	*15*	*mix*	1.2417	4138.16	3835.40	1.67%	-5.77%
Alg. 1a	*15*	*add/remove*	0.2588	4282.48	4098.80	5.22%	0.70%
Alg. 1a	*15*	*Inter-switch*	1.9348	4484.08	4346.20	10.17%	6.78%
Alg. 1a	*15*	*Intra-switch*	0.0078	5366.20	5366.20	31.84%	31.84%
Alg. 1a	*30*	*mix*	1.6518	4320.40	3814.00	6.15%	-6.29%
Alg. 1a	*30*	*add/remove*	0.5140	4222.92	3821.40	3.75%	-6.11%
Alg. 1a	*30*	*Inter-switch*	2.5085	4443.88	4350.40	9.18%	6.88%
Alg. 1a	*30*	*Intra-switch*	0.0099	5366.20	5366.20	31.84%	31.84%
Alg. 1b	*15*	*mix*	2.1633	3887.8	3851.2	-4.48%	-5.38%
Alg. 1b	*15*	*add/remove*	0.0831	3999.96	3987.8	-1.73%	-2.02%
Alg. 1b	*15*	*Inter-switch*	2.5641	3922.16	3918.4	-3.64%	-3.73%
Alg. 1b	*15*	*Intra-switch*	0.0247	4185.4	4185.4	2.83%	2.83%
Alg. 1b	*30*	*mix*	2.8736	3876.84	3858.2	-4.75%	-5.21%
Alg. 1b	*30*	*add/remove*	0.1492	3993.88	3987.8	-1.88%	-2.02%
Alg. 1b	*30*	*Inter-switch*	5.4419	3924.04	3918.4	-3.59%	-3.73%
Alg. 1b	*30*	*Intra-switch*	0.0499	4185.4	4185.4	2.83%	2.83%

Table B5.3: Sample: N=50,fleet#2, sample#1 over 5 runs

initial	*stopping*	*method*	mean(time)	mean(cost)	min(cost)	mean(gap)	best(gap)
Alg. 1a	*15*	*mix*	2.0725	14121.68	13436.20	9.02%	3.73%
Alg. 1a	*15*	*add/remove*	0.8322	14484.16	14269.60	11.82%	10.16%
Alg. 1a	*15*	*Inter-switch*	3.3417	14475.64	14324.40	11.75%	10.58%
Alg. 1a	*15*	*Intra-switch*	0.0966	15294.36	15292.40	18.07%	18.06%
Alg. 1a	*30*	*mix*	3.7301	14032.80	13488.00	8.33%	4.13%
Alg. 1a	*30*	*add/remove*	1.8977	14185.80	13532.80	9.51%	4.47%
Alg. 1a	*30*	*Inter-switch*	5.3971	14403.68	14304.20	11.20%	10.43%
Alg. 1a	*30*	*Intra-switch*	0.0171	15292.40	15292.40	18.06%	18.06%
Alg. 1b	*15*	*mix*	2.0641	13073.32	12954.2	0.93%	0.01%
Alg. 1b	*15*	*add/remove*	0.3924	13038.76	12983.4	0.66%	0.23%
Alg. 1b	*15*	*Inter-switch*	1.0873	13325.8	13325.8	2.88%	2.88%
Alg. 1b	*15*	*Intra-switch*	0.0186	13261.4	13246	2.38%	2.26%
Alg. 1b	*30*	*mix*	6.5069	12786.2	12638.6	-1.29%	-2.43%
Alg. 1b	*30*	*add/remove*	1.0082	12887	12776.6	-0.51%	-1.36%
Alg. 1b	*30*	*Inter-switch*	2.9829	13325.8	13325.8	2.88%	2.88%
Alg. 1b	*30*	*Intra-switch*	0.0477	13245.16	13244.6	2.25%	2.25%

B.6 Simulated Annealing

Table B6.1: Sample: N=20, fleet#1, sample#1 over 5 runs

initial	*cooling*	*Epoch*	*method*	mean(time)	mean(cost)	min(cost)	mean(gap)	best(gap)
Alg. 1b	*0.7*	*5*	*mix*	0.2651	4369.24	4165.2	7.35%	2.33%
Alg. 1b	*0.7*	*5*	*add/remove*	0.1872	4338.4	4111	6.59%	1.00%
Alg. 1b	*0.7*	*5*	*Inter-switch*	0.5812	4427.6	4381	8.78%	7.64%
Alg. 1b	*0.7*	*5*	*Intra-switch*	0.01	4385.6	4371	7.75%	7.39%
Alg. 1b	*0.7*	*10*	*mix*	0.8875	4192.6	4133	3.01%	1.54%
Alg. 1b	*0.7*	*10*	*add/remove*	0.6276	4202.48	4190.4	3.25%	2.95%
Alg. 1b	*0.7*	*10*	*Inter-switch*	1.9075	4254.88	4142.6	4.54%	1.78%
Alg. 1b	*0.7*	*10*	*Intra-switch*	0.0322	4370.8	4363	7.39%	7.19%
Alg. 1b	*0.9*	*5*	*mix*	0.4454	4412.4	4396	8.41%	8.00%
Alg. 1b	*0.9*	*5*	*add/remove*	0.3518	4330.56	4166.2	6.40%	2.36%
Alg. 1b	*0.9*	*5*	*Inter-switch*	1.0374	4397.6	4369	8.04%	7.34%
Alg. 1b	*0.9*	*5*	*Intra-switch*	0.0182	4369	4363	7.34%	7.19%
Alg. 1b	*0.9*	*10*	*mix*	1.6216	4145	4103.4	1.84%	0.82%
Alg. 1b	*0.9*	*10*	*add/remove*	1.143	4243.88	4197.8	4.27%	3.13%
Alg. 1b	*0.9*	*10*	*Inter-switch*	3.6154	4152.12	4114.6	2.01%	1.09%
Alg. 1b	*0.9*	*10*	*Intra-switch*	0.06	4372	4363	7.41%	7.19%

Table B6.2: Sample: N=20,fleet#2, sample#1 over 5 runs

initial	*cooling*	*Epoch*	*method*	mean(time)	mean(cost)	min(cost)	mean(gap)	best(gap)
Alg. 1b	*0.7*	*5*	*mix*	0.5222	4055.32	4005.8	-0.37%	-1.58%
Alg. 1b	*0.7*	*5*	*add/remove*	0.375	4062.24	4031.6	-0.20%	-0.95%
Alg. 1b	*0.7*	*5*	*Inter-switch*	0.9139	4079.48	4045.6	0.23%	-0.60%
Alg. 1b	*0.7*	*5*	*Intra-switch*	0.0115	4235.24	4199.4	4.05%	3.17%
Alg. 1b	*0.7*	*10*	*mix*	1.2675	4006.56	3995	-1.56%	-1.85%
Alg. 1b	*0.7*	*10*	*add/remove*	0.9415	4057.24	4004.8	-0.32%	-1.61%
Alg. 1b	*0.7*	*10*	*Inter-switch*	2.8021	4018.28	3965.2	-1.28%	-2.58%
Alg. 1b	*0.7*	*10*	*Intra-switch*	0.0368	4192.96	4188.2	3.02%	2.90%
Alg. 1b	*0.9*	*5*	*mix*	0.7055	4016.32	3960.2	-1.32%	-2.70%
Alg. 1b	*0.9*	*5*	*add/remove*	0.4963	4039.16	3923.2	-0.76%	-3.61%
Alg. 1b	*0.9*	*5*	*Inter-switch*	1.5671	4060.92	4043.4	-0.23%	-0.66%
Alg. 1b	*0.9*	*5*	*Intra-switch*	0.0237	4214.24	4193.8	3.54%	3.04%
Alg. 1b	*0.9*	*10*	*mix*	2.4754	3955.32	3919.4	-2.82%	-3.71%
Alg. 1b	*0.9*	*10*	*add/remove*	1.7746	4075.72	4019	0.14%	-1.26%
Alg. 1b	*0.9*	*10*	*Inter-switch*	5.0127	4027.12	3979.6	-1.06%	-2.23%
Alg. 1b	*0.9*	*10*	*Intra-switch*	0.0665	4185.4	4185.4	2.83%	2.83%

Table B6.3: Sample: N=50,fleet#2, sample#1 over 5 runs

initial	*cooling*	*Epoch*	*method*	mean(time)	mean(cost)	min(cost)	mean(gap)	best(gap)
Alg. 1b	*0.7*	*5*	*mix*	0.4909	13436.76	13232	3.73%	2.15%
Alg. 1b	*0.7*	*5*	*add/remove*	0.3634	13100.4	12867.4	1.14%	-0.66%
Alg. 1b	*0.7*	*5*	*Inter-switch*	1.2217	13711.6	13526.8	5.85%	4.43%
Alg. 1b	*0.7*	*5*	*Intra-switch*	0.0182	13404.76	13367.8	3.49%	3.20%
Alg. 1b	*0.7*	*10*	*mix*	1.7455	13265.32	13060	2.41%	0.82%
Alg. 1b	*0.7*	*10*	*add/remove*	1.1849	13189.28	13063.2	1.82%	0.85%
Alg. 1b	*0.7*	*10*	*Inter-switch*	4.1582	13713.08	13447.6	5.87%	3.82%
Alg. 1b	*0.7*	*10*	*Intra-switch*	0.0434	13269.8	13261.4	2.44%	2.38%
Alg. 1b	*0.9*	*5*	*mix*	0.9844	13203.4	13045	1.93%	0.71%
Alg. 1b	*0.9*	*5*	*add/remove*	0.6452	13147.88	13005.8	1.50%	0.41%
Alg. 1b	*0.9*	*5*	*Inter-switch*	2.2004	13680.32	13415.8	5.61%	3.57%
Alg. 1b	*0.9*	*5*	*Intra-switch*	0.0245	13330.28	13309	2.91%	2.75%
Alg. 1b	*0.9*	*10*	*mix*	3.2227	13177.44	13019.4	1.73%	0.51%
Alg. 1b	*0.9*	*10*	*add/remove*	2.3013	13301.56	13226.2	2.69%	2.11%
Alg. 1b	*0.9*	*10*	*Inter-switch*	7.694	13604.76	13337	5.03%	2.96%
Alg. 1b	*0.9*	*10*	*Intra-switch*	0.0809	13265.04	13246	2.41%	2.26%

B.7 Node-to-node Ant Colony

Table B7.1: N=20, fleet#1, sample#1 over 5 runs

α	β	***weights***	mean(time)	mean(cost)	min(cost)	mean(gap)	best(gap)
1	*1*	*Arithmetic*	8.3672	4386.07	4251	7.86%	4.54%
1	*0*	*Arithmetic*	9.1785	4444.53	4272.6	9.30%	5.07%
1.5	*1*	*Geometric*	9.1406	4641.4	4434	14.14%	9.04%
1.5	*1*	*Arithmetic*	8.0676	4491.53	4348.6	10.46%	6.94%
1.5	*1.5*	*Geometric*	9.3343	4686.6	4182.8	15.25%	2.86%

Table B7.2: Sample: N=20,fleet#2, sample#1 over 5 runs

α	β	***weights***	mean(time)	mean(cost)	min(cost)	mean(gap)	best(gap)
1	*1*	*Arithmetic*	10.1033	4416.33	4200.6	8.50%	3.20%
1	*0*	*Arithmetic*	10.5056	4331.47	4172.4	6.42%	2.51%
1.5	*1*	*Geometric*	12.3064	4357.67	4081.8	7.06%	0.28%
1.5	*1*	*Arithmetic*	10.5825	4326.07	4205.4	6.29%	3.32%
1.5	*1.5*	*Geometric*	12.2199	4514.13	4385.6	10.91%	7.75%

Table B7.3: Sample: N=50,feet#2, sample#1 over 5 runs

α	β	***weights***	mean(time)	mean(cost)	min(cost)	mean(gap)	best(gap)
1	*1*	*Arithmetic*	32.4286	14406.53	14213.6	11.22%	9.73%
1	*0*	*Arithmetic*	34.6678	14224	14058	9.81%	8.53%
1.5	*1*	*Geometric*	36.2459	14114.27	13780.2	8.96%	6.38%
1.5	*1*	*Arithmetic*	36.3392	14467.07	14281.4	11.69%	10.25%
1.5	*1.5*	*Geometric*	35.1925	13875.4	13612	7.12%	5.09%

B.8 Couple-to-couple Ant Colony

Table B8.1: N=20, fleet#1, sample#1 over 5 runs

α	β	***weights***	**mean(time)**	**mean(cost)**	**min(cost)**	**mean(gap)**	**best(gap)**
1	*1*	*Arithmetic*	8.355	4573.33	4552	12.47%	11.94%
1	*0*	*Arithmetic*	8.5601	4441.6	4300.8	9.23%	5.77%
1.5	*1*	*Geometric*	8.8176	4465.33	4448	9.81%	9.39%
1.5	*1*	*Arithmetic*	7.7592	4486.6	4337.8	10.34%	6.68%
1.5	*1.5*	*Geometric*	8.8894	4371.07	4228.2	7.49%	3.98%

Table B8.2: Sample: N=20,fleet#2, sample#1 over 5 runs

α	β	***weights***	**mean(time)**	**mean(cost)**	**min(cost)**	**mean(gap)**	**best(gap)**
1	*1*	*Arithmetic*	9.5268	4339.73	4192	6.62%	2.99%
1	*0*	*Arithmetic*	9.6801	4335.87	4153	6.53%	2.03%
1.5	*1*	*Geometric*	10.6295	4294.27	4087.8	5.50%	0.43%
1.5	*1*	*Arithmetic*	9.8966	4215.53	4177	3.57%	2.62%
1.5	*1.5*	*Geometric*	11.0857	4340.6	4128.8	6.64%	1.44%

Table B8.3: N=50,feet#2, sample#1 over 5 runs

α	β	***weights***	**mean(time)**	**mean(cost)**	**min(cost)**	**mean(gap)**	**best(gap)**
1	*1*	*Arithmetic*	39.1198	14414.2	14324.2	11.28%	10.58%
1	*0*	*Arithmetic*	39.8126	14375.13	14293.4	10.98%	10.35%
1.5	*1*	*Geometric*	41.4604	14201.2	14014.2	9.63%	8.19%
1.5	*1*	*Arithmetic*	41.4933	14473.07	14384.6	11.73%	11.05%
1.5	*1.5*	*Geometric*	40.5064	14162.6	14099.6	9.34%	8.85%

References

[1] Roberto Baldacci, Maria Battarra, and Daniele Vigo. *Routing a Heterogeneous Fleet of Vehicles*. 2008.

[2] Bruce Golden, Arjang Assad, Larry Levy, and FilipGheysens. The fleet size and mix vehicle routing problem. 1984.

[3] Gerhard Hiermann, Jakob Puchinger, Stefan Ropke, and Richard F Hartl. The electric fleet size and mix vehicle routing problem with time windows and recharging stations. 2016.

[4] Shaohui Hong, Defu Zhang, Hoong Chuin Lau, XiangXiang Zeng, and Yain-Whar Si. A hybrid heuristic algorithm for the 2d variable-sized bin packing problem. 2014.

[5] F-H Liu and S-Y Shen. The fleet size and mix vehicle routing problem with time windows. 1999.

[6] Andrea Lodi, Silvano Martello, and Daniele Vigo. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. 1999.

[7] Artur Pessoa, Ruslan Sadykov, and Eduardo Uchoa. Solving bin packing problems using vrpsolver models. 2021.

[8] Michael Schneider, Andreas Stenger, and Dominik Goeke. The electric vehicle-routing problem with time windows and recharging stations. 2014.

[9] Paolo Toth and Daniele Vigo. *Vehicle Routing: Problems, Methods, and Applications*. second edition, 2014.

[10] Lijun Wei, Wee-Chong Oon, Wenbin Zhu, and Andrew Lim. A goal-driven approach to the 2d bin packing and variable-sized bin packing problems. 2013.