Studies of How Clustering Formulations Influence the Performance of an Existing Inventory Routing Problem Algorithm

Zhongjie Wu, Zehua Qiu, Xiao Lu, Tianshu Ni
Oct 2022

1. Introduction

One of the essential concepts in supply chain management is to replace sequential decision-making with global decision-making and apply systematic thinking to decision-making. Implementing the vendor management inventory (VMI) strategy enables the sharing of information between the supply and demand sides, thus making it possible to effectively integrate the two links of inventory and transportation in the logistics distribution system with conflicting interests. The goal of the VMI strategy is to ultimately minimize logistical costs by coordinating inventory and shipping activities. Moreover, such an optimization problem of coordinating inventory and transportation activities is the inventory routing problem, which is the core problem in implementing the VMI strategy and a new research field that will be produced with the continuous deepening of supply chain management research. Driven by the commercial practice of VMI, the optimization of inventory and transportation integration has received more and more attention.

Inventory routing problem, a variant of the famous vehicle routing program, also falls into the domain of NP-hard, which implies the difficulty of obtaining the optimal solution given a large-scale problem instance. At present, there is much research on the integration and optimization of inventory and transportation, and some achievements have been accomplished. Some breakthroughs have been made in the establishment of mathematical models and algorithm systems, and good benefits have also been achieved in an application. Practices have proven that significant cost savings can be achieved when inventory and shipping are integrated. Since the 1980s, many scholars have devoted themselves to discussing the IRP problem, and proposed many models and algorithms to solve it. The purpose of this paper is to provide an analysis of one of these matheuristic algorithms for the multi-depot inventory routing problem, along with some of our discussion of its clustering stage, which would be helpful for future research.

Why do we need to solve IRP?

IRP problems are crucial in real life in terms of emergency goods allocation as well as furniture delivery as examples. The main goal of IRP is to deliver the optimal amount of goods in the shortest time with the optimal vehicle routing at the same time. Hence, it is significant to find the balance between the customers' interest and how to minimize the total cost. Different goods to be delivered might have different preferences and will lead to completely different models to solve the IRP problem. For instance, the space in each vehicle for huge furniture delivery tends to be limited hence the effectiveness problem of the delivery, utilities delivery after natural disasters prefer more to optimize the time, and fuel delivery needs specific vehicles which increases the costs. As a result, many sub-branches of these types of questions are proposed. Since the complicated nature of this kind of problem and the application of the companies over the years, the market is always seeking a more comprehensive algorithm that could reduce the cost. A more comprehensive algorithm is extremely helpful in both the academic and real world.

IRP problem description and formulation

Under the VMI strategy, within a certain planning period, multiple warehouses provide goods distribution services to multiple customers. In this process, it is necessary to determine the quantity and time of replenishment of inventory to each customer, as well as the driving path of the vehicle. When certain constraints are met (demand quantity of goods, demand time, inventory level, etc.), the total operating cost of the system is minimized, or the total revenue is maximized. Its purpose is to determine the vehicles assigned to each site, the customers each site serves, the number of deliveries to each customer, and the delivery routes travelled during each time period. This problem is known as the Multi-Warehouse Inventory Routing Problem (MDIRP). In fact, the essence of the problem is to integrate the many-to-many distribution problem of inventory and transportation, which is related to the coordination relationship between inventory replenishment and itinerary arrangement.

In the mathematical expression, we consider a complete undirected graph G = (V, E), where V is the set of vertices and E is the set of edges. We split V into $V = P \cup I$, where P = {1, 2, 3, …, m} is a set of depots delivering products to a set of customers I = {m+1, m+2, …, m+n} over a finite and discrete time horizon H. Let T = {1, 2, 3, ..., H} be the corresponding set of time periods. We define the edge as a pair of vertices (i, j) where i denotes a depot and j denotes a customer. Let c_{ij} be the cost of edge from i to j. A set of K = {1, 2, 3, ..., M} vehicles of

capacity C are available for delivery. Each vehicle $k \in K$ can be assigned to a depot $p \in P$ for each time period t. Each customer $i \in I$ has a demand d_{it} in time period t which can be served by different vehicles from different sites. Two types of batch deliveries are allowed for the same period: two vehicles at the same site or two vehicles at different sites visiting the customer. Both of these situations usually only occur when the capacity of each vehicle is small. Furthermore, the highest inventory level U_i and a given starting inventory level I_{i0} are associated with each customer i. We assume that both U_i and I_{i0} are integers. The inventory level of each customer *i* cannot be negative in each time period, i.e. no stockouts are allowed. Our goal is to determine:

- the set of vehicles assigned to each depot at each time period,
- the number of products delivered to each customer using each vehicle in each time period,
- set of routes travelled in each time period,

and minimize the total routing cost within the time horizon.

A mathematical formula was provided by Bertazzi et al [1], which can be solved by branch-and-cut. In this way, they were able to compare the value of the solution obtained using a mathematical algorithm with the value of the solution found by branch-and-cut. To this end, they propose edge-based formulations related to routing problems, where binary variables are associated with graph edges for each vehicle and each epoch. The following notations are introduced. $\delta(S)$ is the set of (i, i'), where $i \in S \subset V$ and $i' \notin S$, which S represents the edge cutset of E. $E(u)$ is the set of edges (i, j) such that $i, j \in U$, where $U \subseteq I$ is a subset of customers. Their mathematical formulation is based on the following variables:

- I_{it}: inventory level of customer i at the end of period t;
- x_{ijktp}: binary variable equal to 1 if vehicle k travels directly from vertex i to vertex j in cycle t starting from depot p, $(i, j) \in E$;
- y_{iktp}: the number of vehicles k delivered from warehouse p to customer i during period t;
- z_{iktp}: binary variable equal to 1 if vehicle k visits vertex i from site p in cycle t.

The mathematical formula is described as:

$$\min \sum_{(i,j)\in E}\sum_{k\in K}\sum_{t\in T}\sum_{p\in P} c_{ij}x_{ijktp} \tag{1}$$

$$s.t.\ I_{it} = I_{it-1} + \sum_{k\in K}\sum_{p\in P} y_{ikt-1p} - d_{it-1} \qquad \forall t \in T \cup \left\{H+1\right\}, \quad \forall i \in I \tag{2}$$

$$I_{it} + \sum_{p\in P}\sum_{k\in K} y_{iktp} \leqslant U_i \qquad \forall t \in T, \quad \forall i \in I \tag{3}$$

$$\sum_{i\in I} y_{iktp} \leqslant C z_{pktp} \qquad \forall t \in T, \quad \forall p \in P, \quad \forall k \in K \tag{4}$$

$$\sum_{i\in I} y_{iktp} \geqslant z_{pktp} \qquad \forall t \in T, \quad \forall p \in P, \quad \forall k \in K \tag{5}$$

$$y_{iktp} \leqslant C z_{iktp} \qquad \forall i \in I, \quad \forall t \in T, \quad \forall p \in P, \quad \forall k \in K \tag{6}$$

$$\sum_{p\in P} z_{pktp} \leqslant 1 \qquad \forall t \in T, \quad \forall k \in K \tag{7}$$

$$\sum_{j\in I:(j,i)\in E} x_{jiktp} + \sum_{j\in I:(i,j)\in E} x_{ijktp} = 2z_{iktp} \qquad \forall i \in I, \quad \forall t \in T, \quad \forall p \in P, \quad \forall k \in K \tag{8}$$

$$\sum_{(i,j)\in E(S)} x_{ijktp} \leqslant \sum_{i\in S} z_{iktp} - z_{uktp} \qquad \forall S \subseteq I, \left|S\right| \geqslant 2, \quad \forall t \in T, \quad \forall k \in K, \quad \forall p \in P, u \in S \tag{9}$$

$$x_{ijktp}, x_{pjktp}, x_{jpktp} \in \{0,1\} \qquad \forall i,j \in I, \quad \forall t \in T, \quad \forall p \in P, \quad \forall k \in K \tag{10}$$

$$I_{it} \geqslant 0 \qquad \forall i \in I, \quad \forall t \in T \cup \{H+1\} \tag{11}$$

$$y_{iktp} \geqslant 0 \qquad \forall i \in I, \quad \forall t \in T, \quad \forall p \in P, \quad \forall k \in K \tag{12}$$

$$z_{iktp} \in \{0,1\} \qquad \forall i \in I, \quad \forall t \in T, \quad \forall p \in P, \quad \forall k \in K. \tag{13}$$

Fig 1. The mathematical formulation

Literature Review

Numerous works have been done on solving IRP. Son and Chi Tao [2] introduced an approach for solving more significant dimension inventory routing problems with logistic ration based DC programming. Ma et al. [3] proposed a multi-objective optimization model to solve the

multi-depot vehicle routing problem of hazardous material transportation, which is solved by a two-stage method (TSM) and a hybrid multi-objective genetic algorithm (HMOGA). Their experimental results show that both methods can obtain a Pareto solution set, and the hybrid multi-objective genetic algorithm can find better vehicle routes throughout the transportation network.

In terms of the variant of IRP, Johnny [4] showed multiple solutions to the order cutoff assignment problem. The order cutoff is common in the reality that if customers make orders before some time then the order will be delivered tomorrow, otherwise, it will be delivered the day after tomorrow. Therefore, an order cutoff assignment problem is to acquire an optimal time of order cutoff for retailers. One way to solve this problem is by the TS algorithm, which goes through every subset of the retailers and checks if the demand exceeds the vehicle's maximum capacity. Then it checks if the cutoff is valid - it needs to be valid for the route, otherwise it will check the next available cutoff until it finds a feasible one. After choosing the best cutoff, the route will be stored and used for later partitions. Lastly, he used mathematical programming formulations to solve the problem as follows:

$$\text{Max } z = \sum_i p_i x_i \qquad (1)$$

subject to

$$\sum_i a_{ij} x_i = 1 \qquad \forall j \qquad (2)$$

$$\sum_i x_i \leq M \qquad (3)$$

$$x_i \in \{0,1\} \qquad \forall i \qquad (4)$$

where z is the everyday demand, i is the route, p_i is the demand of route i in one day, x_i is 1 if route i is chosen, and 0 otherwise, and a_{ij} is 1 if route i serves retailer j and 0 otherwise.

In addition, IRP was indispensable in various types and aspects, which can be found in different literature. Vidović, Popović and Branislava [5] focused on an innovative approach to the multi-product multi-period IRP in fuel delivery, which aimed to find the optimal solution for a homogeneous fleet of vehicles from one depot to petrol stations for fuel delivery. This literature provided a Mixed Integer Programming model and a heuristic approach to assess how the fleet size cost will affect the solution by proposing models both with and without this cost. For the

heuristic approach, both Variable Neighbourhood Descent (VND) search types, local intra-period search and large inter-period neighborhood search with various example data testing.

Goal

In existing algorithms, the initial steps are basically the clustering stage. Clustering customers in different ways and levels is an important stage in optimizing route planning for later and reducing travel costs. We aim to look at how the influence of different clustering formulations on the overall performance of the matheuristic algorithm presented in [1], focusing mainly on the constraint that is about whether to enforce the maximum value TC of the average critical level for each cluster computed for CC.

2. Methodology

This section will go through the steps we have taken to study the inventory routing problem. The matheuristic algorithm given in [1] has three major phases: Clustering, Route construction, and Optimization.

2.1 Clustering Phase

In this phase, the group of customers will be partitioned into different groups. Each group contains one depot, which will perform the delivery for all the customers it incorporates. This clustering problem is formed as an integer linear program. We use the python library PULP as the linear program solver in our implementation. The linear program, i.e., equations (22)-(26) in [1], has its implementation as follows:

```
#Define Linear Program
prob = LpProblem("Clustering", LpMinimize)

#Objective Function (22)
constraint22 = lpDot(flat_b_pi, flat_c_pi)
prob += constraint22
# print(constraint22)

#Constraint (23)
for p in range(n_depots):
  prob += lpSum(b_pi[p]) <= CC

#Constraint (24)
for i in range(n_customers):
  prob += b_pi[0][i] + b_pi[1][i] == 1

#Constraint(25)
CL_div_CC = []
for num in CL:
  CL_div_CC.append(num/CC)
for p in range(n_depots):
  prob += lpDot(b_pi[p], CL_div_CC) <= TC
```

```
def make_var_b(p, i):
    return LpVariable(f"b_{p}_{i+n_depots}", lowBound=0, upBound=1, cat='Integer') # Satisfy constraint (26) already
```

2.2 Route Construction phase

In this phase, an initial set of delivery routes will be generated based on different replenishment policies and vehicle capacities. First, we will determine the direct routes from the depot in each cluster to their customers. In our implementation, the routes are generated using Dijkstra's algorithm from `dijkstar` library.

```
graph = Graph()

for i, d in enumerate(d_list):
  for j, cost in enumerate(d):
    if i != j:
      graph.add_edge(i, j, d_list[i, j])

R_l= []
for p, C in enumerate(cluster):
  for customer in C:
    path_info = find_path(graph, p, customer)
    print(path_info)
    path = getattr(path_info, 'nodes')
    R_l.append(path)
```

3. Experiment Setting and Data Generation

3.1 Data Generation

We randomly generated two datasets, one with size 10 and the other with size 100 by the random function using Python, as well as calculating the distances for each point to the others. The data was processed later using Python lists and further generated to arrays as needed.

3.2 Critical Parameters Calculation in Clustering Phase

In the clustering phase, we refer to the algorithm Bertazzi el at [1] that constructed cluster clusters based on two main characteristics: a limited number of customers and the highest average critical level. The critical level CL_i corresponding to each customer is calculated as: $CL_i = \alpha R_i + (1-\alpha)M_i$, where αtakes value between 0 and 1. R_i and M_i are the minimum quantity delivered to customer i in the time frame and the average delivery cost per customer i for all warehouses, respectively. These parameters are obtained by calculating:

$$\widehat{R}_i = \frac{\max\{0, \sum_{t\in T} d_{it} - I_{i0}\}}{U_i}$$

$$\widehat{M}_i = \frac{\sum_{p\in P} DC_i^p}{|P|},$$

Fig 2. Parameters of the clustering stage [1]

where $DC_i^p = c_{TST}^p - c_{TST\setminus i}^p$ represents the estimated traveling cost from depot p to serve customer i. The difference between the optimal transport cost TSP from the depot to serve all customers in I travel cost $p \in P(c_{TST}^p)$ and the optimal TSP travel cost to serve all customers $I \setminus i$ from depot $p \in P(c_{TST\setminus i}^p)$.

```
# define a function to remove some points in dataset
def mis(i):
    tsplib_file = "/content/a10.tsp"
    # construct distance matrix
    distance_matrix = tsplib_distance_matrix(tsplib_file)

    # exclude ith customer
    distance_matrix = np.delete(distance_matrix, i, axis=1)
    distance_matrix = np.delete(distance_matrix, i, axis=0)
    # run the SA with a 2-opt and follow it by a LS with PS3
    permutation, distance = solve_tsp_simulated_annealing(distance_matrix)
    permutation, distance = solve_tsp_local_search(distance_matrix,
                                                   x0 = permutation,
                                                   perturbation_scheme = "ps3")

    return [permutation, distance]

print("Total distance from depot 1 is", mis(1)[1])
print("Total distance from depot 2 is", mis(0)[1])

# compute DC_i of  depot 1 and depot 2 from group P
x = [0]*8
y = [0]*8
for i in range(8):
    k = [1, i+2]
    s = [0, i+1]
    x[i] = mis(k)[1]
    y[i] = mis(s)[1]
```

When solving the TSP problem, we use the python-tsp library. We implemented the algorithm Simulated Annealing (SA) with a 2-opt and followed it by a Local Search (LA) with the PS3 perturbation scheme that comes with the library, and the gap of optimum is less than 10%. Finally, we brought the calculated DC_i^p into the formulas in Fig 2 to calculate R_i and M_i respectively, and CL_i is calculated from this. Here, we set the α equal to 0.5.

```
alpha = 0.5

for i in range(len(I)):
    max_inventory = I[i][2]
    initial_inventory = I[i][3][0]
    sum = 0
    for t in T:
        sum += I[i][4][t] - initial_inventory
    if sum > 0:
        R[i] = sum / max_inventory
    else:
        R[i] = 0

TSP_all = [3166, 3227]
TSP = [[2457, 2983, 3050, 3101, 3076, 3121, 3128, 2851], [3137, 3023, 3044, 3111, 3162, 2873, 3182, 3133]]
cost = 0
for i in range(len(I)):
    for p in range(len(P)):
        cost += TSP_all[p] - TSP[p][i]
    M[i] = cost / len(P)
```

```
for i in range(len(I)):
    CL[i] = alpha * R[i] + (1 - alpha) * M[i]



for i in range(8):
    print(R[i], end=', ')
print('\n')
for i in range(8):
    print(M[i], end=', ')
print('\n')
for i in range(8):
    print(CL[i], end=', ')
```

The final CL we calculated for each customer i is:

```
199.75, 296.5, 371.25, 416.5, 455.8, 555.47, 576.05, 678.0
```

3.3 Critical Parameters Calculation in Routing Construction Phase

For routing within the resulting cluster, we refer to the four replenishment policies of Bertazzi el at [1]

- Order-up-to-level policy: allows the customer's inventory level sufficient to meet the customer's demand during certain time periods. Its calculation formula is:

$$\hat{y}_{it}\left(g_1\right)=\begin{cases} U_i-\hat{I}_{it-1}\left(g_1\right), & \text{if } d_{it}\geqslant \hat{I}_{it-1}\left(g_1\right) \\ 0, & \text{otherwise} \end{cases}$$

$$\text{where } \hat{I}_{it-1}\left(g_1\right)=I_{i0}+\sum_{h=1}^{t-2}\hat{y}_{ih}\left(g_1\right)-\sum_{h=1}^{t-2}d_{ih}.$$

Fig 3. Formula for oder-up-to-level policy [1]

Corresponding code:

```
# g1 policy
g1_clus1 = np.empty((len(cluster_1), len(T)))
for i in range(len(cluster_1)):
    for j in range(len(T)):
        yih_g1 = 0
        d_ih = 0
        Iit_g1 = data[cluster_1[i]][3][0] + yih_g1 - d_ih
        if data[cluster_1[i]][4][j] >= Iit_g1:
            yit_g1 = data[cluster_1[i]][2] - Iit_g1
        else:
            yit_g1 = 0
        g1_clus1[i, j] = yit_g1
g1_clus1 = g1_clus1.astype(int)
print(g1_clus1) #[[yi0(g1), yi1(g1), yi2(g1)],[cust 2], [cust 3], ...]

g1_clus2 = np.empty((len(cluster_2), len(T)))
for i in range(len(cluster_2)):
    for j in range(len(T)):
        yih_g1 = 0
        d_ih = 0
        Iit_g1 = data[cluster_1[i]][3][0] + yih_g1 - d_ih
        if data[cluster_2[i]][4][j] >= Iit_g1:
            yit_g1 = data[cluster_2[i]][2] - Iit_g1
        else:
            yit_g1 = 0
        g1_clus2[i, j] = yit_g1
g1_clus2 = g1_clus2.astype(int)
print(g1_clus2)
```

- Demand level policy: allows inventory levels equal to demand as long as demand is greater than or equal to current inventory levels. The calculation principle is to let the customer's inventory level equal to 0 after the customer's demand is satisfied:

$$\hat{y}_{it}\left(g_2\right) = \begin{cases} d_{it} - \hat{I}_{it-1}\left(g_2\right), & \text{if } d_{it} \geqslant \hat{I}_{it-1}\left(g_2\right) \\ 0, & \text{otherwise} \end{cases}$$

$$\text{where } \hat{I}_{it-1}\left(g_2\right) = I_{i0} + \sum_{h=1}^{t-2} \hat{y}_{ih}\left(g_2\right) - \sum_{h=1}^{t-2} d_{ih}.$$

Fig 4. Formula for demand level policy [1]

Corresponding code:

```
# g2 policy
g2_clus1 = np.empty((len(cluster_1), len(T)))
for i in range(len(cluster_1)):
    for j in range(len(T)):
        yih_g2 = 0
        d_ih = 0
        Iit_g2 = data[cluster_1[i]][3][0] + yih_g2 - d_ih
        if data[cluster_1[i]][4][j] >= Iit_g2:
            yit_g2 = data[cluster_1[i]][4][j] - Iit_g2
        else:
            yit_g2 = 0
        g2_clus1[i, j] = yit_g2
g2_clus1 = g2_clus1.astype(int)
print(g2_clus1) #[[yi0(g2), yi1(g2), yi2(g2)],[cust 2], [cust 3], ...]

g2_clus2 = np.empty((len(cluster_2), len(T)))
for i in range(len(cluster_2)):
    for j in range(len(T)):
        yih_g2 = 0
        d_ih = 0
        Iit_g2 = data[cluster_1[i]][3][0] + yih_g2 - d_ih
        if data[cluster_2[i]][4][j] >= Iit_g2:
            yit_g2 = data[cluster_2[i]][4][j] - Iit_g2
        else:
            yit_g2 = 0
        g2_clus2[i, j] = yit_g2
g2_clus2 = g2_clus2.astype(int)
print(g2_clus2)
```

- Initial-inventory-level policy: the maximum value between restoring the initial inventory level I_i0 and the demand d_it in each time period, as long as the current inventory level is lower than the demand:

$$\hat{y}_{it}\left(g_3\right) = \begin{cases} \max\left\{I_{i0} - \hat{I}_{it-1}\left(g_3\right), d_{it} - \hat{I}_{it-1}\left(g_3\right)\right\}, & \text{if } d_{it} \geqslant \hat{I}_{it-1}\left(g_3\right) \\ 0, & \text{otherwise} \end{cases}$$

$$\text{where } \hat{I}_{it-1}\left(g_3\right) = I_{i0} + \sum_{h=1}^{t-2} \hat{y}_{ih}\left(g_3\right) - \sum_{h=1}^{t-2} d_{ih}.$$

Fig 5. Formula for initial-inventory-level policy [1]

Corresponding code:

```
# g3 policy
g3_clus1 = np.empty((len(cluster_1), len(T)))
for i in range(len(cluster_1)):
    for j in range(len(T)):
        yih_g3 = 0
        d_ih = 0
        Iit_g3 = data[cluster_1[i]][3][0] + yih_g3 - d_ih
        if data[cluster_1[i]][4][j] >= Iit_g3:
            yit_g3 = max(data[cluster_1[i]][4][j] - Iit_g3, data[cluster_1[i]][3][0] - Iit_g3)
        else:
            yit_g3 = 0
        g3_clus1[i, j] = yit_g3
g3_clus1 = g3_clus1.astype(int)
print(g3_clus1) #[[yi0(g3), yi1(g3), yi2(g3)],[cust 2], [cust 3], ...]

g3_clus2 = np.empty((len(cluster_2), len(T)))
for i in range(len(cluster_2)):
    for j in range(len(T)):
        yih_g3 = 0
        d_ih = 0
        Iit_g3 = data[cluster_2[i]][3][0] + yih_g3 - d_ih
        if data[cluster_2[i]][4][j] >= Iit_g3:
            yit_g3 = max(data[cluster_2[i]][4][j] - Iit_g3, data[cluster_2[i]][3][0] - Iit_g3)
        else:
            yit_g3 = 0
        g3_clus2[i, j] = yit_g3
g3_clus2 = g3_clus2.astype(int)
print(g3_clus2)
```

- Critical-customers-level policy: for customers who do not have sufficient inventory levels to meet demand, make sure to deliver goods twice its demand value, depending on the value of the critical level CL:

$$\hat{y}_{it}\left(g_4\right) = \begin{cases} \min\{2d_{it}, U_i\}, & \text{if } d_{it} \geqslant \hat{I}_{it-1}\left(g_4\right), \text{ and } CL_i \geqslant 6 \\ d_{it}, & \text{if } d_{it} \geqslant \hat{I}_{it-1}\left(g_4\right), \text{ and } CL_i < 6 \\ 0, & \text{otherwise} \end{cases}$$

$$\text{where } \hat{I}_{it-1}\left(g_4\right) = I_{i0} + \sum_{h=1}^{t-2} \hat{y}_{ih}\left(g_4\right) - \sum_{h=1}^{t-2} d_{ih}.$$

Fig 6. Formula for critical-customers-level policy [1]

Here, we set the cutoff value to be the average CL for all customers since we did not scale the CL from 0 to 10.

Corresponding code:

```
# g4 policy
cl_avg = np.sum(CL) / len(CL)
g4_clus1 = np.empty((len(cluster_1), len(T)))
for i in range(len(cluster_1)):
    for j in range(len(T)):
        yih_g4 = 0
        d_ih = 0
        Iit_g4 = data[cluster_1[i]][3][0] + yih_g4 - d_ih
        if (data[cluster_1[i]][4][j] >= Iit_g4 and CL[(np.array(cluster_1) - 2)[i]] >= cl_avg):
            yit_g4 = min(2*data[cluster_1[i]][4][j], data[cluster_1[i]][2])
        elif (data[cluster_1[i]][4][j] >= Iit_g4 and CL[(np.array(cluster_1) - 2)[i]] < cl_avg):
            yit_g4 = data[cluster_1[i]][4][j]
        else:
            yit_g4 = 0
        g4_clus1[i, j] = yit_g4
g4_clus1 = g4_clus1.astype(int)
print(g4_clus1) #[[yi0(g4), yi1(g4), yi2(g4)],[cust 2], [cust 3], ...]

g4_clus2 = np.empty((len(cluster_2), len(T)))
for i in range(len(cluster_2)):
    for j in range(len(T)):
        yih_g4 = 0
        d_ih = 0
        Iit_g4 = data[cluster_1[i]][3][0] + yih_g4 - d_ih
        if (data[cluster_2[i]][4][j] >= Iit_g4 and CL[(np.array(cluster_2) - 2)[i]] >= cl_avg):
            yit_g4 = min(2*data[cluster_2[i]][4][j], data[cluster_2[i]][2])
        elif (data[cluster_2[i]][4][j] >= Iit_g4 and CL[(np.array(cluster_2) - 2)[i]] < cl_avg):
            yit_g4 = data[cluster_2[i]][4][j]
        else:
            yit_g4 = 0
        g4_clus2[i, j] = yit_g4
g4_clus2 = g4_clus2.astype(int)
print(g4_clus2)
```

4. Future Work

In the future, we are planning to expand the size of the generated dataset and test the implemented algorithm on a larger scale with varying conditions of the clustering formulation. It is predictable that a larger dataset challenges the ability to produce a valid output, and therefore we must optimize our code in terms of efficiency and complexity. In addition, more problem instances should be initiated and tested with our code so that correctness and accuracy can be guaranteed to some extent. Given different clustering strategies and problem instances, we should perform a detailed analysis based on the comparison between them, so we can have a better understanding of what our code is best for - in which situation we should apply the algorithm and solve the case most efficiently.

5. References

[1] Luca Bertazzi, Leandro C. Coelho, Annarita De Maio, Demetrio Laganà,
A matheuristic algorithm for the multi-depot inventory routing problem,
Transportation Research Part E: Logistics and Transportation Review, Volume 122, 2019, Pages 524-544, ISSN 1366-5545, https://doi.org/10.1016/j.tre.2019.01.005.

[2] T. A. Son and N. Chi Thao, "Solving Inventory Routing Problem with Logistic Ratio via DC programming," 2020 12th International Conference on Knowledge and Systems Engineering (KSE), 2020, pp. 258-262, doi: 10.1109/KSE50997.2020.9287855.

[3] Ma, C.; Mao, B.; Xu, Q.; Hua, G.; Zhang, S.; Zhang, T. Multi-Depot Vehicle Routing Optimization Considering Energy Consumption for Hazardous Materials Transportation. Sustainability 2018, 10, 3519. https://doi.org/10.3390/su10103519

[4] Tam, J. W.-Y. (2011). Vehicle Routing Approaches for Solving an Order Cutoff Assignment Problem. Thesis (M.A.Sc.)--University of Toronto, 2011.

[5] Milorad Vidović, Dražen Popović, Branislava Ratković. Mixed integer and heuristics model for the inventory routing problem in fuel delivery, International Journal of Production Economics, Volume 147, Part C, 2014, Pages 593-604, ISSN 0925-5273, https://doi.org/10.1016/j.ijpe.2013.04.034.