Name: **Kunal Sinha** Batch: **Swarm R1- G2** Roll no: **2K17/CO/164**

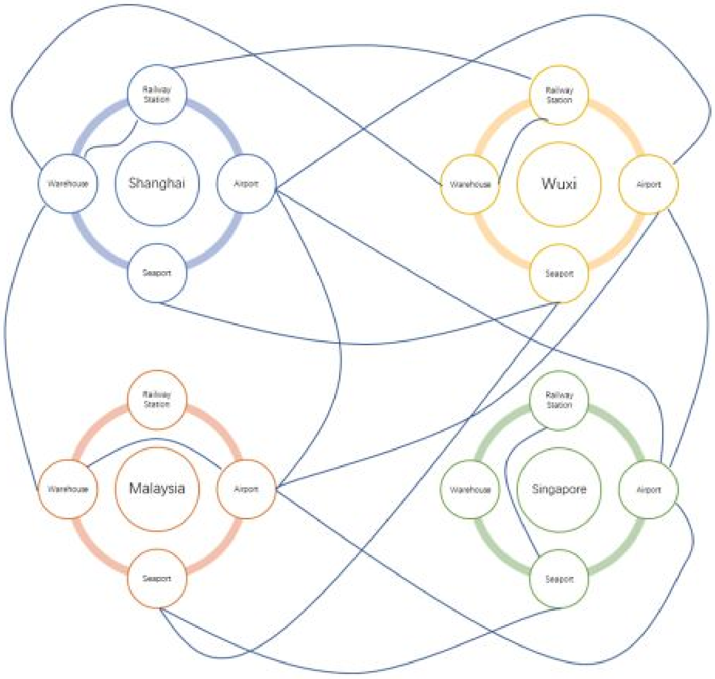
Experiment 8

Aim: Write a program to implement any kind of optimization on a multimodal dataset

Theory:

In this experiment, we would be using swarm algorithms to solve multi-modal transportation cost minimization in goods delivery and do the required optimization for better results.

In our simulated case, there are 8 goods, 4 cities/countries (Shanghai, Wuxi, Singapore, Malaysia), 16 ports and 4 transportation tools. The 8 goods originate from different cities and have different destinations. Each city/country has 4 ports, the airport, railway station, seaport and warehouse. There are in total 50 direct routes connecting different ports. Each route has a specific transportation tool, transportation cost, transit time and weekly schedule. Warehouses in each city allow goods to be deposited for a period of time so as to fit certain transportation schedules or wait for other goods to be transported together. All goods might have different order dates and different delivery deadlines. With all these criteria, how can we find out solution routes for all goods that minimize the overall cost?



Optimizing of routes using various swarm algorithms

Algorithm:

In order to make the criteria logic clearer and the calculation more efficient, we use the concept of the matrix to build the necessary components in the model. In our case, there are totally 4 dimensions:

1. Start Port: i

Indicating the start port of a direct transport route. The dimension length equals the total number of ports in the data.

2. End Port: j

Indicating the end port of a direct transport route. The dimension length equals the total number of ports in the data.

3. Time: t

Indicating the departure time of direct transport. The dimension length equals the total number of days between the earliest order date and the latest delivery deadline date of all goods in the data.

4. Goods: k

Indicating the goods to be transported. The dimension length equals the total number of goods in the data. All the variable or parameter matrices to be introduced in the later parts will have one or more of these 4 dimensions.

The objective of the model is to minimize the overall cost, which includes 3 parts, transportation cost, warehouse cost and tax cost. Firstly, the transportation cost includes container cost and route fixed cost. Container cost equals the number of containers used in each route times per container cost while route fixed cost equals the sum of the fixed cost of all routes.

Source Code:

multi.py

```
from
docplex.mp.model
import Model
from itertools import product
import numpy as np
import cvxpy as cp
import pandas as pd
import json
class MMT:
 '''a Model class that solves the multi-model transportation optimization problem.'''
 def __init__(self, framework='DOCPLEX'):
        self.portSpace = None
        self.dateSpace = None
        self.goods = None
        self.indexPort = None
        self.portIndex = None
        self.maxDate = None
        self.minDate = None
        self.tranCost = None
        self.tranFixedCost = None
        self.tranTime = None
        self.ctnVol = None
        self.whCost = None
        self.kVol = None
        self.kValue = None
        self.kDDL = None
        self.kStartPort = None
        self.kEndPort = None
        self.kStartTime = None
        self.taxPct = None
        self.transitDuty = None
        self.route_num = None
        self.available_routes = None
        # decision variables
        self.var = None
        self.x = None
        self.var_2 = None
        self.y = None
        self.var_3 = None
        self.z = None
        # result & solution
        self.xs = None
        self.ys = None
        self.zs = None
        self.whCostFinal = None
        self.transportCost = None
        self.taxCost = None
        self.solution_ = None
        self.arrTime_ = None
        self.objective_value = None
        # helping variables
        self.var_location = None
        self.var_2_location = None
        self.var_3_location = None

       if framework not in ['CVXPY', 'DOCPLEX']:
```

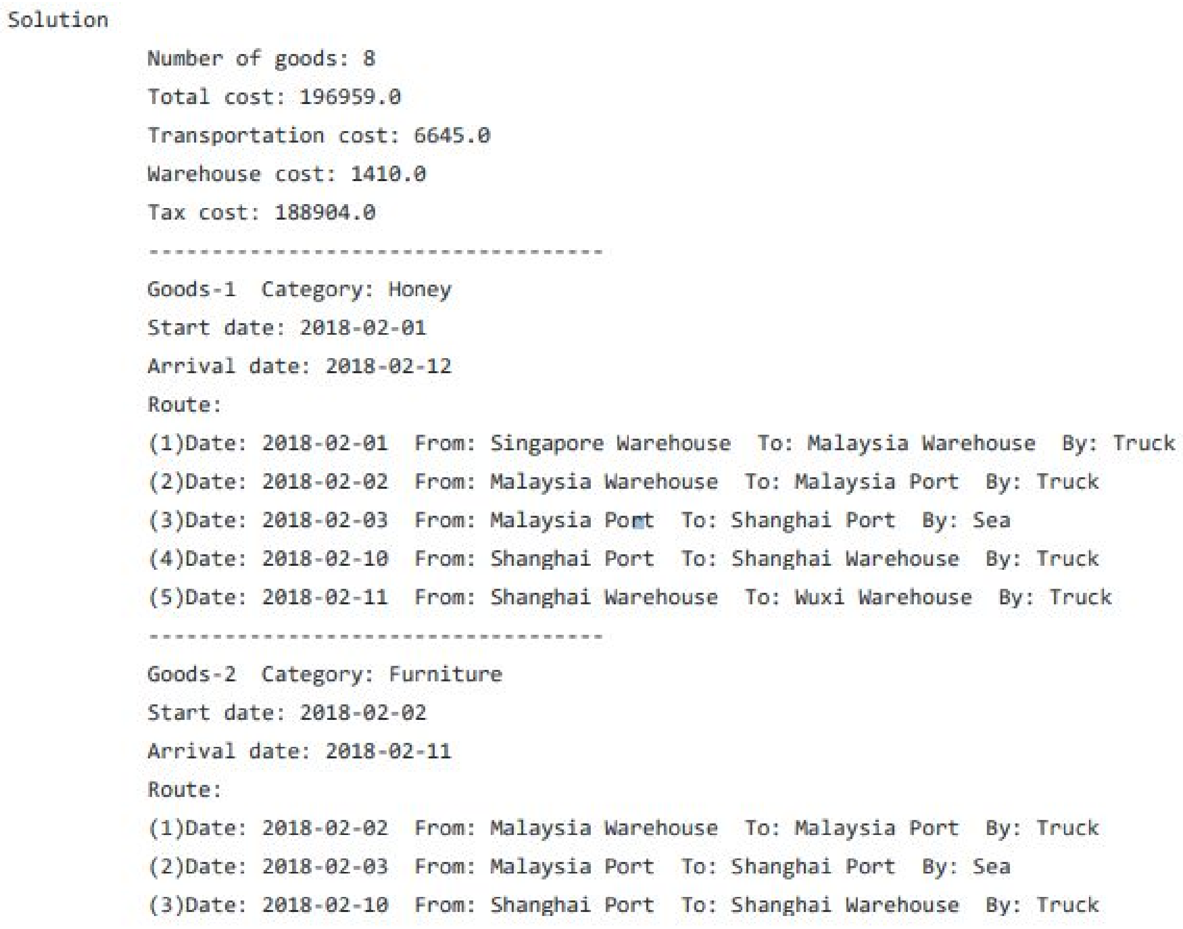
```
            raise ValueError('Framework not supported, the model only supports CVXPY and
DOCPLEX')
        else:
            self.framework = framework

    def set_param(self, route, order):
    '''set model parameters based on the read-in route and order information.'''
        bigM = 100000
        route = route[route['Feasibility'] == 1]
        route['Warehouse Cost'][route['Warehouse Cost'].isnull()] = bigM
        route = route.reset_index()
        portSet = set(route['Source']) | set(route['Destination'])
        self.portSpace = len(portSet)
        self.portIndex = dict(zip(range(len(portSet)), portSet))
        self.indexPort = dict(zip(self.portIndex.values(), self.portIndex.keys()))
        self.maxDate = np.max(order['Required Delivery Date'])
        self.minDate = np.min(order['Order Date'])
        self.dateSpace = (self.maxDate - self.minDate).days
        startWeekday = self.minDate.weekday() + 1
        weekday = np.mod((np.arange(self.dateSpace) + startWeekday), 7)
        weekday[weekday == 0] = 7
        weekdayDateList = {i: [] for i in range(1, 8)}
        for i in range(len(weekday)):
            weekdayDateList[weekday[i]].append(i)
        for i in weekdayDateList:
            weekdayDateList[i] = json.dumps(weekdayDateList[i])

        source = list(route['Source'].replace(self.indexPort))
        destination = list(route['Destination'].replace(self.indexPort))
        DateList =list(route['Weekday'].replace(weekdayDateList).apply(json.loads))
        self.goods = order.shape[0]
        self.tranCost = np.ones([self.portSpace, self.portSpace, self.dateSpace])* bigM
        self.tranFixedCost = np.ones([self.portSpace, self.portSpace, self.dateSpace]) * bigM
        self.tranTime = np.ones([self.portSpace, self.portSpace, self.dateSpace])* bigM
        for i in range(route.shape[0]):
            self.tranCost[source[i], destination[i], DateList[i]] = route['Cost'][i]
            self.tranFixedCost[source[i], destination[i], DateList[i]] =route['Fixed Freight
Cost'][i]
            self.tranTime[source[i], destination[i], DateList[i]] = route['Time'][i]
        self.transitDuty = np.ones([self.portSpace, self.portSpace]) * bigM
        self.transitDuty[source, destination] = route['Transit Duty']
        # make the container size of infeasible routes to be small enough, similar to bigM
        self.ctnVol = np.ones([self.portSpace, self.portSpace]) * 0.1
        self.ctnVol[source, destination] = route['Container Size']
        self.ctnVol = self.ctnVol.reshape(self.portSpace, self.portSpace, 1)
        self.whCost = route[['Source', 'Warehouse Cost']].drop_duplicates()
        self.whCost['index'] = self.whCost['Source'].replace(self.indexPort)
        self.whCost = np.array(self.whCost.sort_values(by='index')['WarehouseCost'])
        self.kVol = np.array(order['Volume'])
        self.kValue = np.array(order['Order Value'])
        self.kDDL = np.array((order['Required Delivery Date'] - self.minDate).dt.days)
        self.kStartPort = np.array(order['Ship From'].replace(self.indexPort))
        self.kEndPort = np.array(order['Ship To'].replace(self.indexPort))
        self.kStartTime = np.array((order['Order Date'] - self.minDate).dt.days)
        self.taxPct = np.array(order['Tax Percentage'])

        # add available route indexes
        self.route_num = route[['Source','Destination']].drop_duplicates().shape[0]
        routes = route[['Source','Destination']].drop_duplicates().replace(self.indexPort)
        self.available_routes = list(zip(routes['Source'], routes['Destination']))

        # localization variables of decision variables in the matrix
        var_location = product(self.available_routes, range(self.dateSpace),range(self.goods))
        var_location = [(i[0][0], i[0][1], i[1], i[2]) for i in var_location]
        self.var_location = tuple(zip(*var_location))
        var_2_location = product(self.available_routes, range(self.dateSpace))
        var_2_location = [(i[0][0], i[0][1], i[1]) for i in var_2_location]
        self.var_2_location = tuple(zip(*var_2_location))
        self.var_3_location = self.var_2_location

    def build_model(self):
    '''overall function to build up model objective and constraints'''
        if self.framework == 'CVXPY':
            self.cvxpy_build_model()
        elif self.framework == 'DOCPLEX':
            self.cplex_build_model()

    def cvxpy_build_model(self):
    '''build up the mathematical programming model's objective and constraints using CVXPY
    framework.'''
    # 4 dimensional binary decision variable matrix
        self.var = cp.Variable(self.route_num * self.dateSpace * self.goods, boolean=True,
    name='x')
```

```
        self.x = np.zeros((self.portSpace, self.portSpace, self.dateSpace,
self.goods)).astype('object')
        self.x[self.var_location] = list(self.var)

        # 3 dimensional container number matrix
        self.var_2 = cp.Variable(self.route_num * self.dateSpace, integer=True, name='y')
        self.y = np.zeros((self.portSpace, self.portSpace, self.dateSpace)).astype('object')
        self.y[self.var_2_location] = list(self.var_2)

        self.var_3 = cp.Variable(self.route_num * self.dateSpace, boolean=True, name='z')
        self.z = np.zeros((self.portSpace, self.portSpace, self.dateSpace)).astype('object')
        self.z[self.var_3_location] = list(self.var_3)
        # warehouse related cost
        warehouseCost, arrTime, stayTime = self.warehouse_fee(self.x)
        transportCost = np.sum(self.y * self.tranCost) + np.sum(self.z *self.tranFixedCost)
        transitDutyCost = np.sum(np.sum(np.dot(self.x, self.kValue), axis=2) * self.transitDuty)
        taxCost = np.sum(self.taxPct * self.kValue) + transitDutyCost
        objective = cp.Minimize(transportCost + warehouseCost + taxCost)

        constraints = []
        constraints += [np.sum(self.x[self.kStartPort[k], :, :, k]) == 1 for k in range(self.goods)]
        constraints += [np.sum(self.x[:, self.kEndPort[k], :, k]) == 1 for k in range(self.goods)]

        constraints += [np.sum(self.x[:, self.kStartPort[k], :, k]) == 0 for k in range(self.goods)]
        constraints += [np.sum(self.x[self.kEndPort[k], :, :, k]) == 0 for k in range(self.goods)]

        for k in range(self.goods):
                for j in range(self.portSpace):
                        if (j != self.kStartPort[k]) & (j != self.kEndPort[k]):
                                constraints.append(np.sum(self.x[:, j, :, k]) == np.sum(self.x[j, :, :,
k]))

        constraints += [np.sum(self.x[i, :, :, k]) <= 1 for k in range(self.goods)
        for i in range(self.portSpace)]
                constraints += [np.sum(self.x[:, j, :, k]) <= 1 for k in range(self.goods)
        for j in range(self.portSpace)]
                constraints += [stayTime[j, k] >= 0 for j in range(self.portSpace) for k in
range(self.goods)]

        numCtn = np.dot(self.x, self.kVol) / self.ctnVol
        constraints += [self.y[i, j, t] - numCtn[i, j, t] >= 0 \
                                        for i in range(self.portSpace) for j in
        range(self.portSpace) for t in
                                        range(self.dateSpace) if not isinstance(self.y[i, j, t] -
        numCtn[i, j, t] >= 0, bool)]

        constraints += [self.z[i, j, t] >= (np.sum(self.x[i, j, t, :]) * 10e-5) \
         for i in range(self.portSpace)
                for j in range(self.portSpace)
                        for t in range(self.dateSpace)
                                if not isinstance(self.z[i, j, t] >= (np.sum(self.x[i, j, t, :]) * 10e-5),
bool)]
                                        constraints += [np.sum(arrTime[:, self.kEndPort[k], :, k])
<= self.kDDL[k]
        for k in range(self.goods)
         if not isinstance(np.sum(arrTime[:, self.kEndPort[k], :, k]) <= self.kDDL[k], bool)]
                model = cp.Problem(objective, constraints)
         self.objective = objective
         self.constraints = constraints
         self.model = model

 def cplex_build_model(self):
 '''build up the mathematical programming model's objective and constraints using DOCPLEX
framework.'''
        model = Model()

         self.var = model.binary_var_list(self.route_num * self.dateSpace *self.goods, name='x')
         self.x = np.zeros((self.portSpace, self.portSpace,
self.dateSpace,self.goods)).astype('object')
         self.x[self.var_location] = self.var

         # 3 dimensional container number matrix
         self.var_2 = model.integer_var_list(self.route_num * self.dateSpace,name='y')
         self.y = np.zeros((self.portSpace, self.portSpace,self.dateSpace)).astype('object')
         self.y[self.var_2_location] = self.var_2

         self.var_3 = model.binary_var_list(self.route_num * self.dateSpace,name='z')
         self.z = np.zeros((self.portSpace, self.portSpace,self.dateSpace)).astype('object')
         self.z[self.var_3_location] = self.var_3
         warehouseCost, arrTime, stayTime = self.warehouse_fee(self.x)
         transportCost = np.sum(self.y * self.tranCost) + np.sum(self.z *self.tranFixedCost)
         transitDutyCost = np.sum(np.sum(np.dot(self.x, self.kValue), axis=2) *self.transitDuty)
         taxCost = np.sum(self.taxPct * self.kValue) + transitDutyCost
         model.minimize(transportCost + warehouseCost + taxCost)
         model.add_constraints(np.sum(self.x[self.kStartPort[k], :, :, k]) == 1 for k in
range(self.goods))
                model.add_constraints(np.sum(self.x[:, self.kEndPort[k], :, k]) == 1 for k in
range(self.goods))
                model.add_constraints(np.sum(self.x[:, self.kStartPort[k], :, k]) == 0 for k in
range(self.goods))
                model.add_constraints(np.sum(self.x[self.kEndPort[k], :, :, k]) == 0 for k in
range(self.goods))

        for k in range(self.goods):
         for j in range(self.portSpace):
                if (j != self.kStartPort[k]) & (j != self.kEndPort[k]):
                        model.add_constraint(np.sum(self.x[:, j, :, k]) ==np.sum(self.x[j, :, :, k]))

         model.add_constraints(np.sum(self.x[i, :, :, k]) <= 1 for k in range(self.goods) for i in
range(self.portSpace))
         model.add_constraints(np.sum(self.x[:, j, :, k]) <= 1 for k in  range(self.goods) for j in
range(self.portSpace))
         # 5.transition-out should be after transition-in
         model.add_constraints(stayTime[j, k] >= 0 for j in range(self.portSpace)
        for k in range(self.goods))
         # 6.constraint for number of containers used
         numCtn = np.dot(self.x, self.kVol) / self.ctnVol
         model.add_constraints(self.y[i, j, t] - numCtn[i, j, t] >= 0 \
         for i in range(self.portSpace) for j in
        range(self.portSpace) for t in
         range(self.dateSpace) if not isinstance(self.y[i, j,
        t] - numCtn[i, j, t] >= 0, bool))
         # 7. constraint to check whether a route is used
         model.add_constraints(self.z[i, j, t] >= (np.sum(self.x[i, j, t, :]) *
        10e-5) \
         for i in range(self.portSpace) for j in range(self.portSpace) for t in range(self.dateSpace)
         if not isinstance(self.z[i, j, t] >= (np.sum(self.x[i, j, t, :]) * 10e-5), bool))
                 model.add_constraints(np.sum(arrTime[:, self.kEndPort[k], :, k])
<=self.kDDL[k] for k in range(self.goods)
         if not isinstance(np.sum(arrTime[:,self.kEndPort[k], :, k]) <= self.kDDL[k], bool))
         self.objective = model.objective_expr
         self.constraints = list(model.iter_constraints())
         self.model = model

 def solve_model(self, solver=cp.CBC):
```

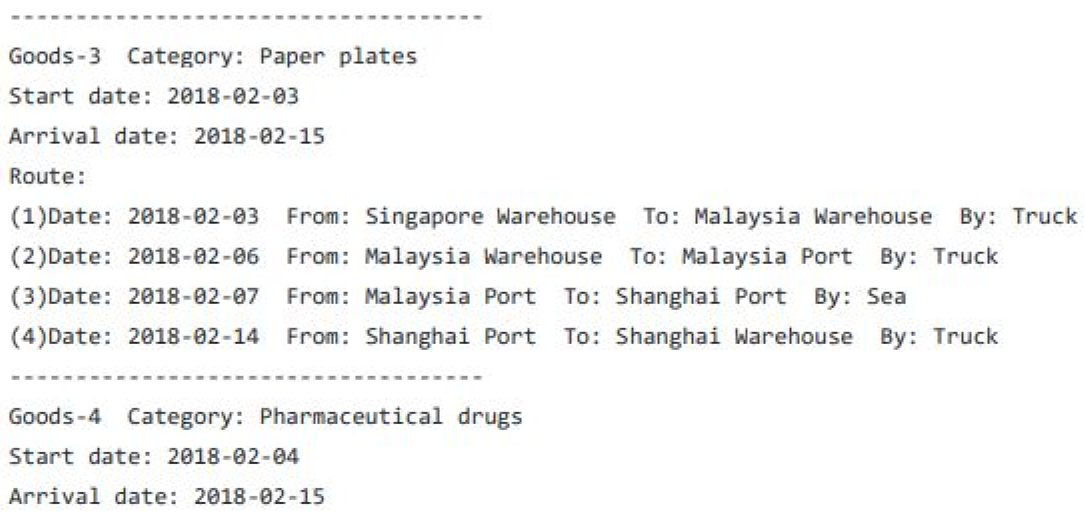
```
 try:
 if self.framework == 'CVXPY':
 self.objective_value = self.model.solve(solver)
 self.xs = np.zeros((self.portSpace, self.portSpace,
self.dateSpace, self.goods))
 self.xs[self.var_location] = self.var.value
 self.ys = np.zeros((self.portSpace, self.portSpace,
self.dateSpace))
 self.ys[self.var_2_location] = self.var_2.value
 self.zs = np.zeros((self.portSpace, self.portSpace,
self.dateSpace))
 self.zs[self.var_3_location] = self.var_3.value
 elif self.framework == 'DOCPLEX':
 ms = self.model.solve()
 self.objective_value = self.model.objective_value
 self.xs = np.zeros((self.portSpace, self.portSpace,
self.dateSpace, self.goods))
 self.xs[self.var_location] = ms.get_values(self.var)
 self.ys = np.zeros((self.portSpace, self.portSpace,
self.dateSpace))
 self.ys[self.var_2_location] = ms.get_values(self.var_2)
 self.zs = np.zeros((self.portSpace, self.portSpace,
self.dateSpace))
 self.zs[self.var_3_location] = ms.get_values(self.var_3)
 except:
 raise Exception('Model is not solvable, no solution will be provided')
 nonzeroX = list(zip(*np.nonzero(self.xs)))
 nonzeroX = sorted(nonzeroX, key=lambda x: x[2])
 nonzeroX = sorted(nonzeroX, key=lambda x: x[3])
 nonzeroX = list(map(lambda x: (self.portIndex[x[0]], self.portIndex[x[1]],
\
 (self.minDate + pd.to_timedelta(x[2],
unit='days')).date().isoformat(),
 x[3]), nonzeroX))
 self.whCostFinal, arrTime, _ = self.warehouse_fee(self.xs)
 self.transportCost = np.sum(self.ys * self.tranCost) + np.sum(self.zs *
self.tranFixedCost)
 self.taxCost = np.sum(self.taxPct * self.kValue) + \
 np.sum(np.sum(np.dot(self.xs, self.kValue), axis=2) *
self.transitDuty)
 self.solution_ = {}
 self.arrTime_ = {}
```

```
         for i in range(self.goods):
         self.solution_['goods-' + str(i + 1)] = list(filter(lambda x: x[3] ==
        i, nonzeroX))
         self.arrTime_['goods-' + str(i + 1)] = (self.minDate + pd.to_timedelta
        \
         (np.sum(arrTime[:, self.kEndPort[i], :, i]),
        unit='days')).date().isoformat()

 def get_output_(self):
        return self.objective_value, self.solution_, self.arrTime_

 def warehouse_fee(self, x):
         startTime = np.arange(self.dateSpace).reshape(1, 1, self.dateSpace, 1) * x
         arrTimeMtrx = startTime + self.tranTime.reshape(self.portSpace, \ self.portSpace,
self.dateSpace, 1) * x
         arrTime = arrTimeMtrx.copy()
         arrTimeMtrx[:, self.kEndPort.tolist(), :, range(self.goods)] = 0
         stayTime = np.sum(startTime, axis=(1, 2)) - np.sum(arrTimeMtrx, axis=(0, 2))
         stayTime[self.kStartPort.tolist(), range(self.goods)] -= self.kStartTime
         warehouseCost = np.sum(np.sum(stayTime * self.kVol, axis=1) * self.whCost)
         return warehouseCost, arrTime, stayTime

 def txt_solution(self, route, order):
         '''transform the cached results to text.'''
         travelMode = dict(zip(zip(route['Source'], route['Destination']), route['Travel Mode']))
         txt = "Solution"
         txt += "\nNumber of goods: " + str(order['Order Number'].count())
         txt += "\nTotal cost: " + str(self.transportCost + self.whCostFinal + self.taxCost)
         txt += "\nTransportation cost: " + str(self.transportCost)
         txt += "\nWarehouse cost: " + str(self.whCostFinal)
         txt += "\nTax cost: " + str(self.taxCost)
         for i in range(order.shape[0]):
         txt += "\n------------------------------------"
         txt += "\nGoods-" + str(i + 1) + " Category: " + order['Commodity'][i]
         txt += "\nStart date: " + pd.to_datetime(order['Order Date']) \ .iloc[i].date().isoformat()
         txt += "\nArrival date: " + str(self.arrTime_['goods-' + str(i + 1)])
         txt += "\nRoute:"
         solution = self.solution_['goods-' + str(i + 1)]
         route_txt = ''
         a = 1
         for j in solution:
                 route_txt += "\n(" + str(a) + ")Date: " + j[2]
        route_txt += " From: " + j[0]
        route_txt += " To: " + j[1]
        route_txt += " By: " + travelMode[(j[0], j[1])]
        a += 1
        txt += route_txt
    return txt

def transform(filePath):
    order = pd.read_excel(filePath, sheet_name='Order Information')
    route = pd.read_excel(filePath, sheet_name='Route Information')
    order['Tax Percentage'][order['Journey Type'] == 'Domestic'] = 0
    route['Cost'] = route[route.columns[7:12]].sum(axis=1)
    route['Time'] = np.ceil(route[route.columns[14:18]].sum(axis=1) / 24)
    route = route[list(route.columns[0:4]) +
            ['Fixed Freight Cost', 'Time', \ 'Cost', 'Warehouse Cost', 'Travel Mode',
'Transit Duty'] + list(
    route.columns[-9:-2])]
    route = pd.melt(route, id_vars=route.columns[0:10], value_vars=route.columns[-7:] \ ,
var_name='Weekday', value_name='Feasibility')
    route['Weekday'] = route['Weekday'].replace({'Monday': 1, 'Tuesday': 2,'Wednesday': 3,
\ 'Thursday': 4, 'Friday': 5,'Saturday': 6, 'Sunday': 7})
    return order, route

    if __name__ == '__main__':
    order, route = transform("model data.xlsx")
    m = MMT()
    m.set_param(route, order)
    m.build_model()
    m.solve_model()
    txt = m.txt_solution(route, order)
    with open("Solution.txt", "w") as text_file:
    text_file.write(txt)
```

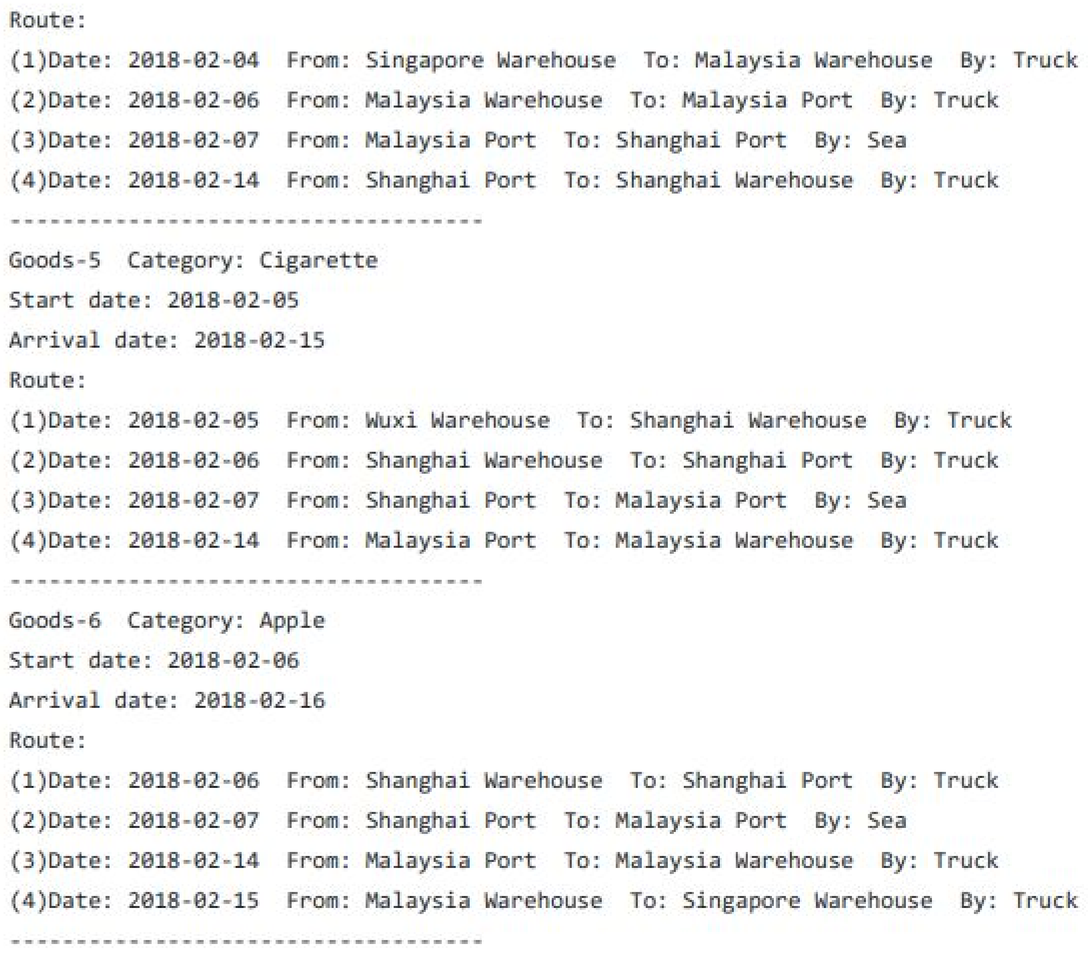
Output:



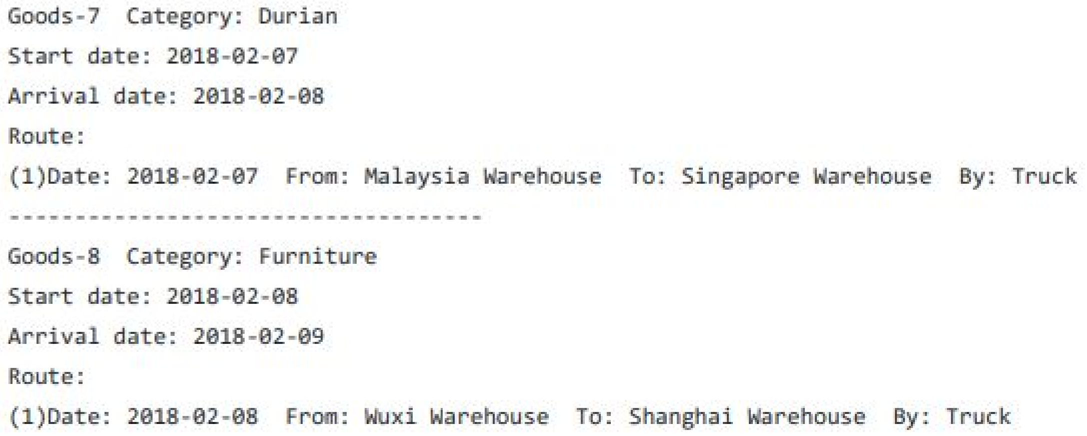
```
Solution
          Number of goods: 8
          Total cost: 196959.0
          Transportation cost: 6645.0
          Warehouse cost: 1410.0
          Tax cost: 188904.0
          ------------------------------------
          Goods-1  Category: Honey
          Start date: 2018-02-01
          Arrival date: 2018-02-12
          Route:
          (1)Date: 2018-02-01  From: Singapore Warehouse  To: Malaysia Warehouse  By: Truck
          (2)Date: 2018-02-02  From: Malaysia Warehouse  To: Malaysia Port  By: Truck
          (3)Date: 2018-02-03  From: Malaysia Port  To: Shanghai Port  By: Sea
          (4)Date: 2018-02-10  From: Shanghai Port  To: Shanghai Warehouse  By: Truck
          (5)Date: 2018-02-11  From: Shanghai Warehouse  To: Wuxi Warehouse  By: Truck
          ------------------------------------
          Goods-2  Category: Furniture
          Start date: 2018-02-02
          Arrival date: 2018-02-11
          Route:
          (1)Date: 2018-02-02  From: Malaysia Warehouse  To: Malaysia Port  By: Truck
          (2)Date: 2018-02-03  From: Malaysia Port  To: Shanghai Port  By: Sea
          (3)Date: 2018-02-10  From: Shanghai Port  To: Shanghai Warehouse  By: Truck
```



```
------------------------------------
Goods-3  Category: Paper plates
Start date: 2018-02-03
Arrival date: 2018-02-15
Route:
(1)Date: 2018-02-03  From: Singapore Warehouse  To: Malaysia Warehouse  By: Truck
(2)Date: 2018-02-06  From: Malaysia Warehouse  To: Malaysia Port  By: Truck
(3)Date: 2018-02-07  From: Malaysia Port  To: Shanghai Port  By: Sea
(4)Date: 2018-02-14  From: Shanghai Port  To: Shanghai Warehouse  By: Truck
------------------------------------
Goods-4  Category: Pharmaceutical drugs
Start date: 2018-02-04
Arrival date: 2018-02-15
```



```
Route:
(1)Date: 2018-02-04  From: Singapore Warehouse  To: Malaysia Warehouse  By: Truck
(2)Date: 2018-02-06  From: Malaysia Warehouse  To: Malaysia Port  By: Truck
(3)Date: 2018-02-07  From: Malaysia Port  To: Shanghai Port  By: Sea
(4)Date: 2018-02-14  From: Shanghai Port  To: Shanghai Warehouse  By: Truck
-----------------------------------
Goods-5  Category: Cigarette
Start date: 2018-02-05
Arrival date: 2018-02-15
Route:
(1)Date: 2018-02-05  From: Wuxi Warehouse  To: Shanghai Warehouse  By: Truck
(2)Date: 2018-02-06  From: Shanghai Warehouse  To: Shanghai Port  By: Truck
(3)Date: 2018-02-07  From: Shanghai Port  To: Malaysia Port  By: Sea
(4)Date: 2018-02-14  From: Malaysia Port  To: Malaysia Warehouse  By: Truck
-----------------------------------
Goods-6  Category: Apple
Start date: 2018-02-06
Arrival date: 2018-02-16
Route:
(1)Date: 2018-02-06  From: Shanghai Warehouse  To: Shanghai Port  By: Truck
(2)Date: 2018-02-07  From: Shanghai Port  To: Malaysia Port  By: Sea
(3)Date: 2018-02-14  From: Malaysia Port  To: Malaysia Warehouse  By: Truck
(4)Date: 2018-02-15  From: Malaysia Warehouse  To: Singapore Warehouse  By: Truck
-----------------------------------
```



```
Goods-7  Category: Durian
Start date: 2018-02-07
Arrival date: 2018-02-08
Route:
(1)Date: 2018-02-07  From: Malaysia Warehouse  To: Singapore Warehouse  By: Truck
-----------------------------------
Goods-8  Category: Furniture
Start date: 2018-02-08
Arrival date: 2018-02-09
Route:
(1)Date: 2018-02-08  From: Wuxi Warehouse  To: Shanghai Warehouse  By: Truck
```

Finding and Learnings:

We have successfully implemented the optimization Algorithm on a multimodal dataset in python.