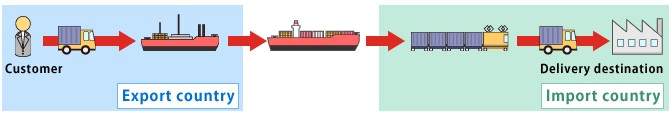
A project on using mathematical programming to solve multi-modal transportation cost minimization in goods delivery and supply chain management.

## **Project Overview:**

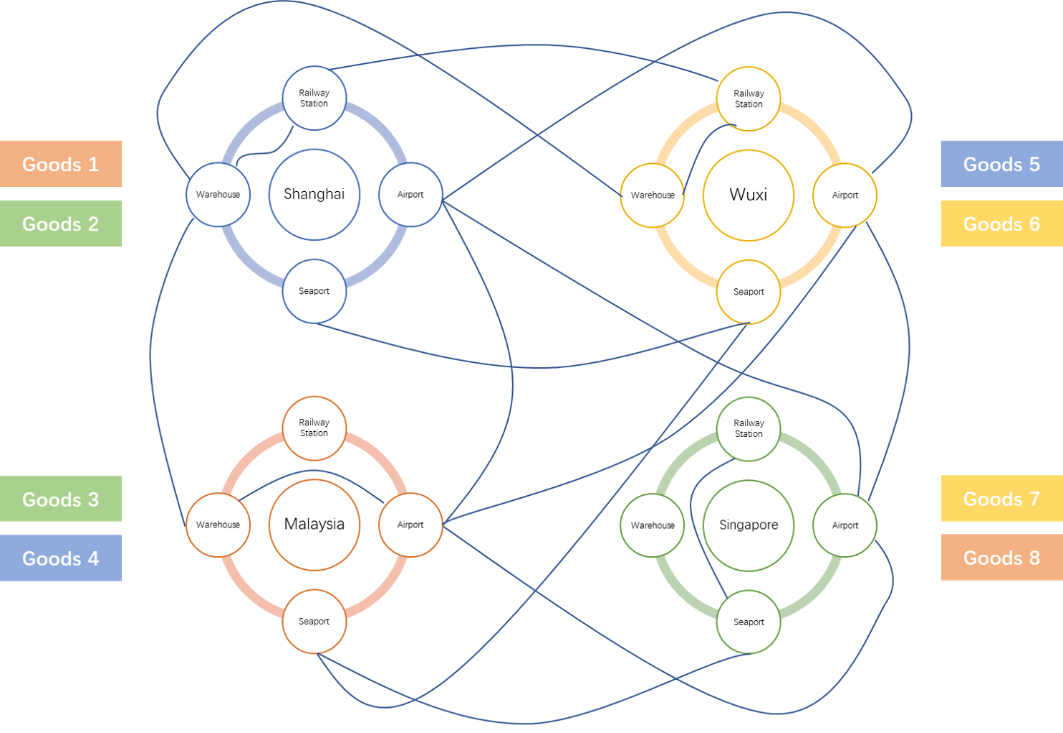
In the domain of delivery services, a myriad of transportation options, including trucks, airplanes, and ships, are at our disposal. Optimal decision-making in terms of route selection and transportation methods is crucial as it directly impacts costs. To achieve cost minimization, factors such as goods consolidation (where multiple goods share a single journey), diverse transportation costs, and delivery time constraints need to be taken into account. This project employs mathematical programming to formulate and solve the problem, aiming to minimize overall costs. The model's construction provides flexibility by offering the choice between two mathematical programming frameworks, namely DOcplex and CVXPY.

DOcplex stands as a proprietary programming framework crafted by IBM, directly interfacing with the CPLEX solver. Opting for this framework necessitates the prior installation of CPLEX Studio. On the other hand, CVXPY is an open-source programming framework initially crafted by a Stanford research team, and additional information about it is available [**here**](https://github.com/cvxgrp/cvxpy). This alternative provides the flexibility to choose from a range of solvers integrated within the CVXPY framework. For optimal performance in this model, the CBC solver is recommended, although it requires an initial installation; further details can be accessed [**here**](https://www.cvxpy.org/tutorial/advanced/index.html#choosing-a-solver).

[](https://user-images.githubusercontent.com/30411828/45585955-c6311e80-b920-11e8-95c9-bc90089446b4.jpg)

## **Problem Statement:**

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. Warehouse in each city allows 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?

[](https://user-images.githubusercontent.com/30411828/45705624-a44bcc00-bbac-11e8-8548-d8181d9fd848.png)

(The above diagram is only used to show the basic idea of the problem, the routes are not complete.)

## **Assumptions:**

Before model building, some assumptions should be made to simplify the case because real-world delivery problems consist of too many unmeasurable factors that can affect the delivery process and final outcomes. Here are the main assumptions:

1. The delivery process is **deterministic**, no random effect will appear on delivery time and cost etc.
2. Goods can be transported in **normal container**, no special containers (refrigerated, thermostatic etc.) will be needed.
3. Container only constraints on the good's **volume**, and all goods are **divisible in terms of volume**. (No bin packing problem needed to be considered.)
4. The model only evaluates the **major carriage routes**. The first and last mile between end user and origin/destination shipping point are not considered. (**From warehouse to warehouse**.)
5. There is **only one transportation tool available between each two ports**. For instance, we can only directly go from one airport to the other airport in different cities by flight, while direct journey by ship or railway or truck is infeasible.
6. Overall cost is restricted to the most important 3 parts, **transportation cost**, **warehouse cost** and **goods tariff**.
7. The minimum unit for time is **day** in the model, and there is **at most one transit in a route in one day**.

## **Dimension & Matrixing:**

In order to make the criteria logic clearer and the calculation more efficient, we use the concept of matrixing 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 a 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.

## **Decision Variables:**

As mentioned above, we will use the concept of **variable matrix**, a list of variables deployed in the form of a matrix or multi-dimensional array. In our model, 3 variable matrices will be introduced:

1. **Decision Variable Matrix:**    ***X***  
   The most important variable matrix in the model. It's a 4 dimensional matrix, each dimension representing start port, end port, time and goods respectively. Each element in the matrix is a binary variable, representing whether a route is taken by a specific goods. For example, element ***Xi,j,t,k*** represents whether **goods k** travels from **port i** to **port j** at **time t**.

varList1 = model.binary\_var\_list(portDim \* portDim \* timeDim \* goodsDim,name = 'x')

x = np.array(varList1).reshape(portDim, portDim, timeDim, goodsDim)

1. **Container Number Matrix:**    ***Y***  
   A variable matrix used to support the decision variable matrix. It's a 3 dimensional matrix, with each dimension representing start port, end port and time respectively. Each element in the matrix is an integer variable, representing the number of containers needed in a specific route. For example, ***Yi,j,t*** represents the number of containers needed to load all the goods travelling simultaneously from **port i** to **port j** at **time t**. Such matrix is introduced to make up for the limitation of "linear operator only" in mathematical programming, when we need a **roundup()** method in direct calculation of the container number.

varList2 = model.integer\_var\_list(portDim \* portDim \* timeDim,name = 'y')

y = np.array(varList2).reshape(portDim, portDim, timeDim)

1. **Route Usage Matrix:**    ***Z***  
   A variable matrix used to support the decision variable matrix. It's a 3 dimensional matrix, with each dimension representing start port, end port and time respectively. Each element in the matrix is a binary variable, representing whether a route is used or not. For instance, ***Zi,j,t*** represents whether the route from **port i** to **port j** at **time t** is used or not (no matter which goods). It's introduced with similar purpose to ***Yi,j,t*** .

varList3 = model.binary\_var\_list(portDim \* portDim \* timeDim,name = 'z')

z = np.array(varList3).reshape(portDim, portDim, timeDim)

## **Parameters:**

Similar to the decision variables, the following parameter arrays or matrices are introduced for the sake of later model building:

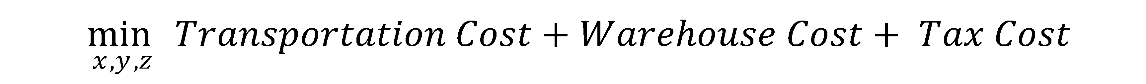
1. **Per Container Cost:**    ***C***  
   A 3 dimensional parameter matrix, each dimension representing start port, end port and time. ***Ci,j,t*** in the matrix represents the overall transportation cost per container from **port i** to **port j** at **time t**. This overall cost includes handling cost, bunker/fuel cost, documentation cost, equipment cost and extra cost from [**model data.xlsx**](https://github.com/hzjken/multimodal-transportation-optimization/blob/master/model%20data.xlsx). For infeasible route, the cost element will be set to be big M (an extremely large number), making the choice infeasible.
2. **Route Fixed Cost:**    ***FC***  
   A 3 dimensional parameter matrix, each dimension representing start port, end port and time. ***FCi,j,t*** in the matrix represents the fixed transportation cost to travel from **port i** to **port j** at **time t**, regardless of goods number or volume. For infeasible route, the cost element will be set to be big M as well.
3. **Warehouse Cost:**    ***wh***  
   A one dimension array with dimension start port. ***whi*** represents the warehouse cost per cubic meter per day at **port i**. Warehouse cost for ports with no warehouse function (like airport, railway station etc.) is set to be big M.
4. **Transportation Time:**    ***T***  
   A 3 dimensional parameter matrix, each dimension representing start port, end port and time. ***Ti,j,t*** in the matrix represents the overall transportation time from **port i** to **port j** at **time t**. This overall time includes custom clearance time, handling time, transit time and extra time. For infeasible route, the time element will be set to be big M.
5. **Tax Percentage:**    ***tax***  
   A one dimension array with dimension goods. ***taxk*** represents the tax percentage for **goods k** imposed by its destination country. If the goods only goes through a domestic transit, the tax percentage for such goods will be set as 0.
6. **Transit Duty:**    ***td***  
   A two dimensional matrix, each dimension representing start port and end port. ***tdi,j*** represents the transit duty (tax imposed on goods passing through a country) percentage for goods to go from **port i** to **port j**. If port i and port j belong to the same country, transit duty percentage is set to be 0. For simplicity purpose, transit duty is set to be equal among all goods. (can be extended easily)
7. **Container Volume:**    ***ctnV***  
   A two dimensional matrix, each dimension representing start port and end port. ***ctnVi,j*** represents the volume of container in the route from **port i** to **port j**.
8. **Goods Volume:**    ***V***  
   A one dimension array with dimension goods. ***Vk*** represents the volume of **goods k**.
9. **Goods Value:**    ***val***  
   A one dimension array with dimension goods. ***valk*** represents the value of **goods k**.
10. **Order Date:**    ***ord***  
    A one dimension array with dimension goods. ***ordk*** represents the order date of **goods k**.
11. **Deadline Date:**    ***ddl***  
    A one dimension array with dimension goods. ***ddlk*** represents the deadline delivery date of **goods k**.
12. **Origin Port:**    ***OP***  
    A one dimension array with dimension goods. ***OPk*** represents the port where **goods k** starts from.
13. **Destination Port:**    ***DP***  
    A one dimension array with dimension goods. ***DPk*** represents the port where **goods k** ends up to be in.

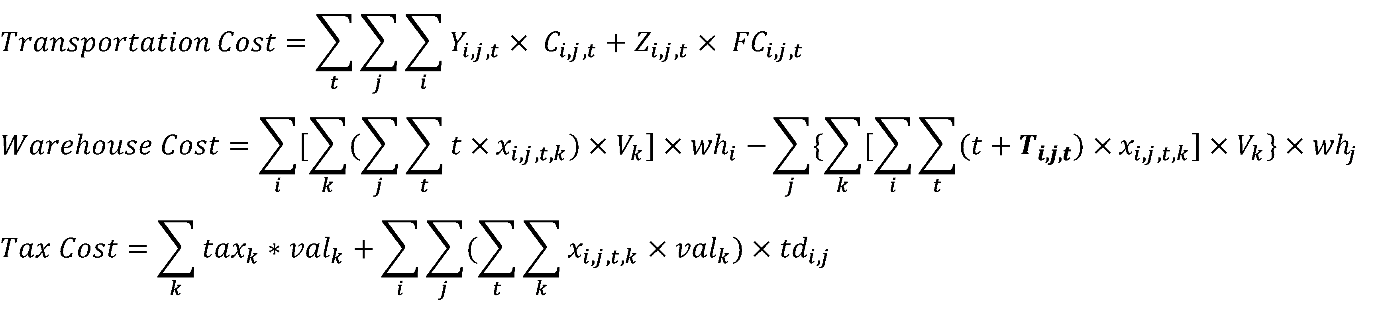
## **Mathematical Modelling:**

With all the variables and parameters defined above, we can build up the objectives and constraints to form an integer programming model.

## **Objective:**

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 fixed cost of all used routes. Secondly, the **warehouse cost** equals all goods' sum of volume times days of storage times warehouse fee per cubic meter per day in each warehouse. Finally, the **tax cost** equals the sum of import tariff and transit duty of all goods. Mathematic formulation and Python implementation are attached below.

[](https://user-images.githubusercontent.com/30411828/45684896-56b66b80-bb7a-11e8-8d6b-0da2d9ec709e.png)

[](https://user-images.githubusercontent.com/30411828/45684632-82852180-bb79-11e8-8fd9-547623e9ab66.png)

transportCost = np.sum(y\*perCtnCost) + np.sum(z\*tranFixedCost)

warehouseCost = warehouse\_fee(x)[0] #For details ,pleas refer to function warehouse\_fee() in code.

taxCost = np.sum(taxPct\*kValue) + np.sum(np.sum(np.dot(x,kValue),axis=2)\*transitDuty)

model.minimize(transportCost + warehouseCost + taxCost)

**Methodology:**

In addressing the complex challenges of multi-modal transportation cost minimization for goods delivery and supply chain management, a robust methodology is employed. Leveraging mathematical programming, this methodology seeks to optimize decision-making regarding route selection and transportation methods, thereby minimizing overall costs. The flexibility of this approach is exemplified by the choice between two powerful frameworks, DOcplex and CVXPY, each offering unique advantages.

**Key Components:**

**Problem Statement:**

The simulated case involves 8 goods, 4 cities, 16 ports, and 4 transportation tools. Various factors such as diverse transportation costs, delivery time constraints, and goods consolidation are considered. The objective is to determine solution routes for all goods that minimize the overall cost, encompassing transportation, warehouse, and tax costs.

**Assumptions:**

To streamline the model, certain assumptions are made, ensuring a deterministic delivery process, standard container usage, and a focus on major carriage routes. The methodology concentrates on the three crucial cost components: transportation cost, warehouse cost, and goods tariff.

**Dimension & Matrixing:**

The introduction of four dimensions—Start Port, End Port, Time, and Goods—facilitates a structured and efficient model. Variable matrices, including decision variables, container numbers, and route usage, are designed to capture the intricate nature of the transportation problem.

**Decision Variables:**

Three pivotal variable matrices—Decision Variable Matrix (X), Container Number Matrix (Y), and Route Usage Matrix (Z)—are defined. These matrices encapsulate binary and integer variables, providing a comprehensive representation of route choices, container numbers, and route usage.

**Parameters:**

Essential parameters, ranging from per-container cost to tax percentages and transit duties, are introduced to characterize the various aspects influencing the model. These parameters contribute to the realistic representation of costs and constraints.

**Mathematical Modeling:**

The core of the methodology lies in mathematical modeling, with a clear.

**objective:**

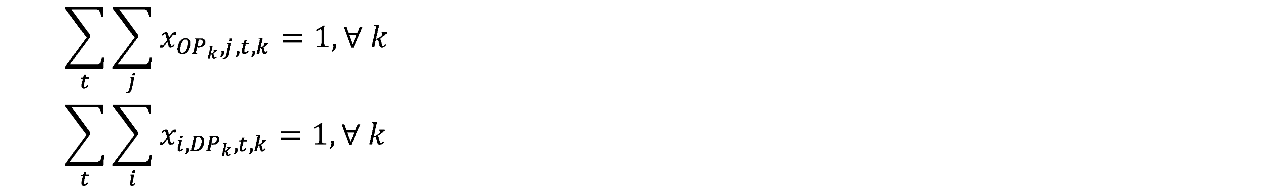
To minimize the overall cost. This involves formulating an objective function that considers transportation costs, warehouse fees, and tax costs. Constraints are then introduced to ensure adherence to shipping requirements, time constraints, and other logistical considerations.

**Innovative Aspects:**

1. The methodology offers flexibility through the choice of two programming frameworks, catering to different preferences and solver requirements.
2. Matrixing is employed for clarity and efficiency in representing the multidimensional nature of the transportation problem.
3. This methodology not only addresses the intricacies of multi-modal transportation but also provides a versatile and scalable framework for real-world applications in goods delivery and supply chain management.

## **Constraints**

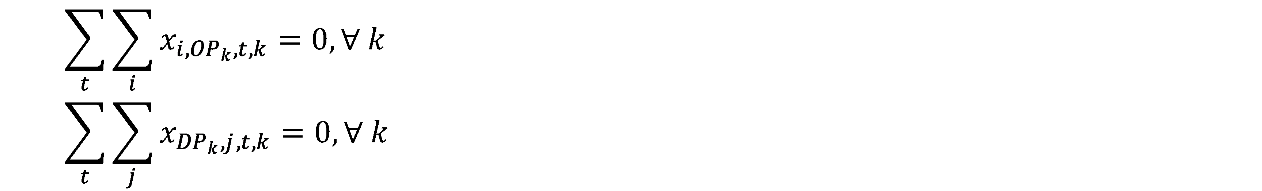
1. For each goods k, it must be shipped out from its origin to another node and shipped to its destination.

[](https://user-images.githubusercontent.com/30411828/45704415-6ac59180-bba9-11e8-989a-a02fa615cdf0.png)

model.add\_constraints(np.sum(x[OriginPort[k],:,:,k]) == 1 for k in range(goodsDim))

model.add\_constraints(np.sum(x[:,DestinationPort[k],:,k]) == 1 for k in range(goodsDim))

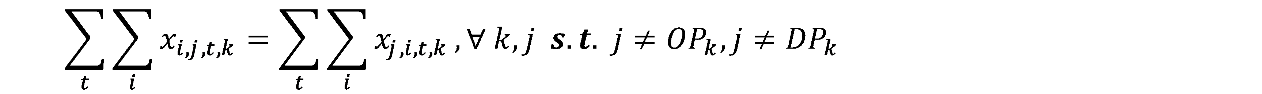
1. For each goods k, it couldn't be shipped out from its destination or shipped to its origin.

[](https://user-images.githubusercontent.com/30411828/45704443-7a44da80-bba9-11e8-82c5-1361ccdf1c47.png)

model.add\_constraints(np.sum(x[:,OriginPort[k],:,k]) == 0 for k in range(goodsDim))

model.add\_constraints(np.sum(x[DestinationPort[k],:,:,k]) == 0 for k in range(goodsDim))

1. For each goods k at transition point j (neither origin nor destination), ship-in times must equal ship-out times.

[](https://user-images.githubusercontent.com/30411828/45704617-f7704f80-bba9-11e8-8162-b9930a2ed4da.png)

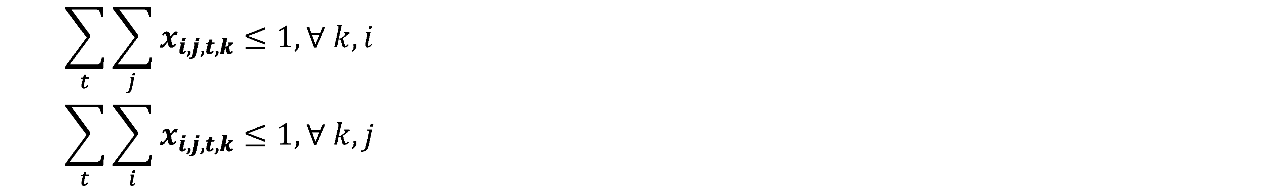
for k in range(goodsDim):

for j in range(portDim):

if (j != OriginPort[k]) & (j != DestinationPort[k]):

model.add\_constraint(np.sum(x[:,j,:,k])==np.sum(x[j,:,:,k]))

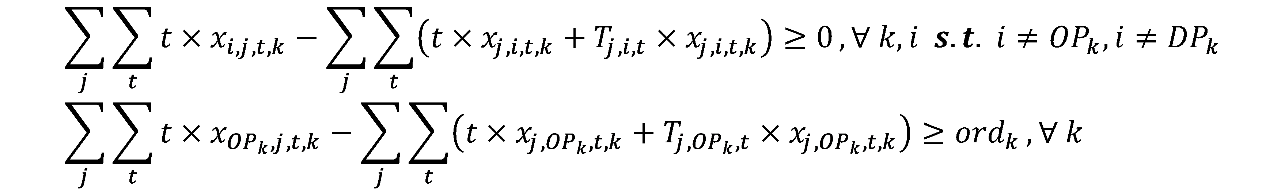
1. Each goods k can only be transitioned in or out of a port for at most once.

[](https://user-images.githubusercontent.com/30411828/45728788-29120680-bbfc-11e8-8b1a-780eba4b6a3f.png)

model.add\_constraints(np.sum(x[i,:,:,k]) <= 1 for k in range(goodsDim) for i in range(portDim))

model.add\_constraints(np.sum(x[:,j,:,k]) <= 1 for k in range(goodsDim) for j in range(portDim))

1. For each goods k at transition port j, ship-out time should be after ship-in time. For goods k at its origin port, ship-out time should be after order date. (Or stay time greater than order date, because ship-in time is none)

[](https://user-images.githubusercontent.com/30411828/45734505-65069500-bc17-11e8-92d9-dc38c652e8a6.png)

startTime = np.arange(timeDim).reshape(1,1,timeDim,1)\*x

arrTime = startTime + tranTime.reshape(portDim,portDim,timeDim,1)\*x

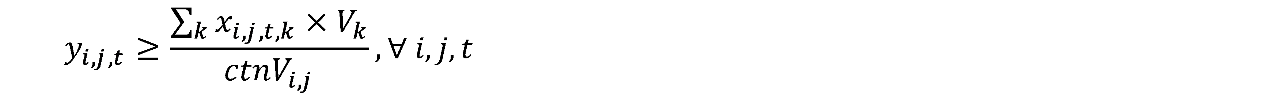
stayTime = np.sum(startTime,axis=(1,2)) - np.sum(arrTime,axis=(0,2))

stayTime[OriginPort,range(goodsDim)] -= OrderDate #ship-out time at origin port should be after order date

stayTime[DestinationPort,range(goodsDim)] = 0 #stay time at destination port is not considered

model.add\_constraints(stayTime[i,k] >= 0 for i in range(portDim) for k in range(goodsDim))

1. At each route at time t, the total volume of containers should be larger than the total volume of goods.

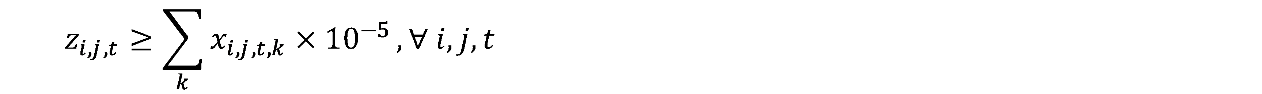
[](https://user-images.githubusercontent.com/30411828/45731416-19012380-bc0a-11e8-984d-4ecfd6272e66.png)

numCtn = np.dot(x,kVol) / ctnVol.reshape(portDim,portDim,1)

model.add\_constraints(y[i,j,t] - numCtn[i,j,t] >= 0 \

for i in range(portDim) for j in range(portDim) for t in range(dateDim))

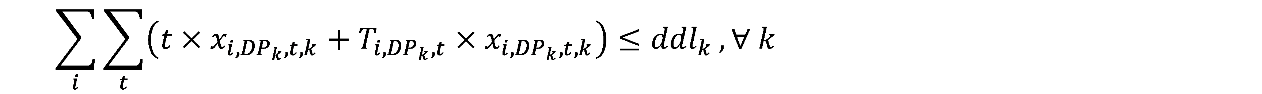
1. Check whether a route is used at time t. Because ***Zi,j,t*** is binary variable, if a route is used, sum of ***Xi,j,t,k*** for all goods k at **i,j,t** must be greater than 0. We can scale it back to [0,1] by multiplying a small number.

[](https://user-images.githubusercontent.com/30411828/45732047-223fbf80-bc0d-11e8-8216-8bc0e72d852d.png)

model.add\_constraints(z[i,j,t] >= np.sum(x[i,j,t,:])\*10e-5 \

for i in range(portDim) for j in range(portDim) for t in range(timeDim))

1. For each goods k, it should be shipped to its destination port before the deadline delivery date.

[](https://user-images.githubusercontent.com/30411828/45732937-27067280-bc11-11e8-9654-8f2b1fa5e2ce.png)

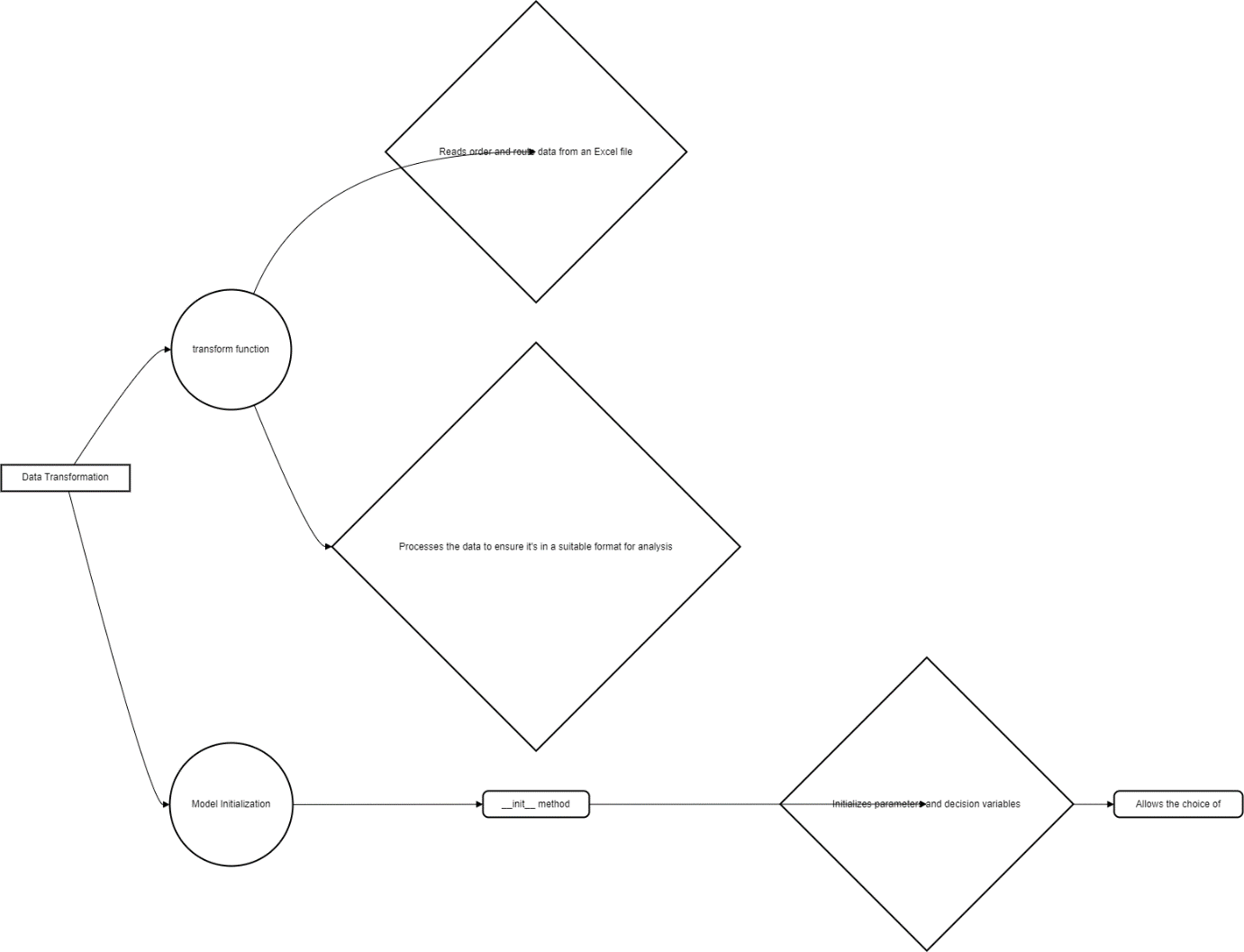
arrTime = np.arange(timeDim).reshape(1,1,timeDim,1)\*x + tranTime.reshape(portDim,portDim,timeDim,1)\*x

model.add\_constraints(np.sum(arrTime[:,DestinationPort[k],:,k]) <= kDDL[k] for k in range(goodsDim))

**Description of the algorithms you have tried:**

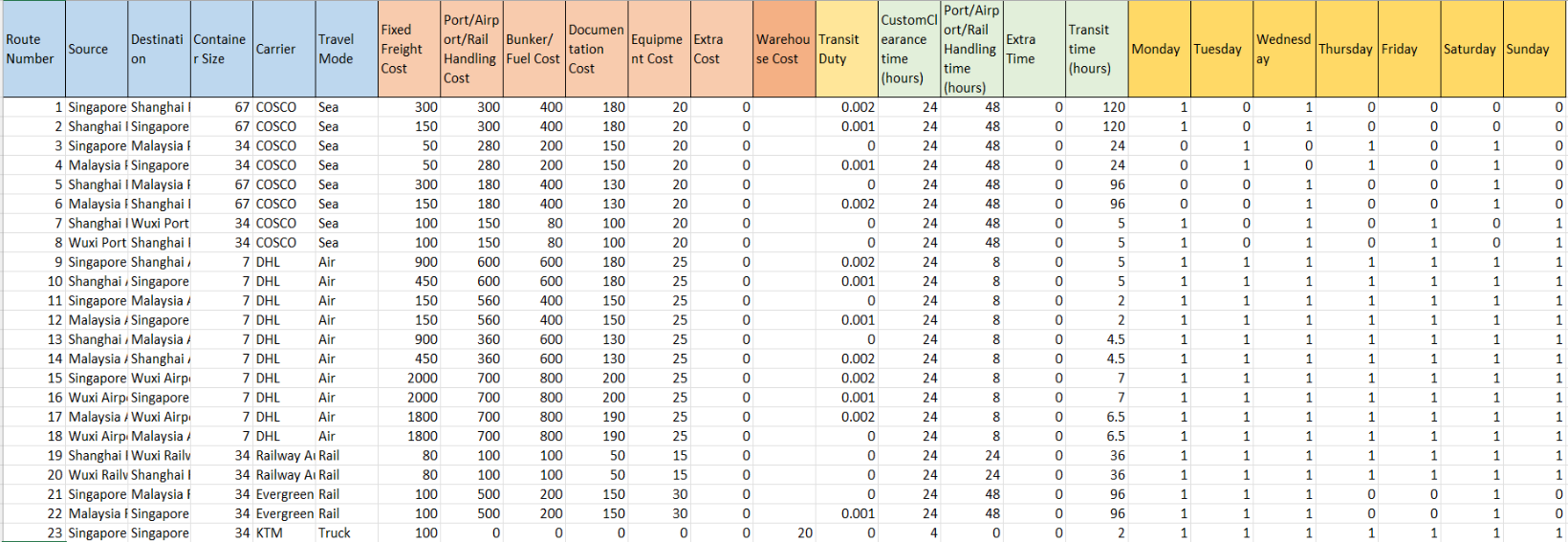
1. **Data Transformation (transform function):**
   * Reads order and route data from an Excel file.
   * Processes the data to ensure it's in a suitable format for analysis.
2. **Model Initialization (\_\_init\_\_ method):**
   * Initializes parameters and decision variables needed for the optimization problem.
   * Allows the choice of the optimization framework (CVXPY or DOCPLEX).
3. **Setting Parameters (set\_param method):**
   * Sets model parameters based on the transformed data.
   * Calculates various parameters such as port indices, date ranges, transit costs, warehouse costs, etc.
4. **Building Mathematical Model (build\_model method):**
   * Builds the optimization model's objective function and constraints based on the chosen framework (CVXPY or DOCPLEX).
5. **Solving the Model (solve\_model method):**
   * Solves the optimization model using the chosen framework (CVXPY or DOCPLEX).
   * Caches the optimized objective value, route, and arrival time for each goods.
6. **Output Generation (get\_output\_ method):**
   * Retrieves the optimized objective value, final solution, and arrival time for each goods.
   * These results can be accessed after the model is solved.
7. **Textual Solution (txt\_solution method):**
   * Transforms the cached results into a human-readable text format.
   * Includes details such as total cost, transportation cost, warehouse cost, tax cost, start date, arrival date, and routes for each good.
8. **Main Execution:**
   * Reads data, initializes the model, sets parameters, builds the model, solves it, generates the textual solution, and writes it to a file named "Solution.txt".

**Data processing pipeline diagram**

****

**Conclusion:**

* + Flexibility: The code allows for flexibility in choosing between two optimization frameworks, CVXPY and DOCPLEX, catering to different preferences or requirements.
  + Data Handling: The transform function efficiently reads and processes data from an Excel file, ensuring it's in the appropriate format for analysis.
  + Model Construction: The MMT class encapsulates the construction of the optimization model, including setting parameters, defining decision variables, building the objective function, and specifying constraints.
  + Solver Agnosticism: The pipeline is designed to work with different solvers seamlessly, abstracting away solver-specific details and providing a consistent interface for model solving.
  + Result Interpretation: The txt\_solution method generates a human-readable summary of the optimization results, including total cost breakdown, start and arrival dates, and detailed routes for each shipment.
  + Robustness: The code incorporates error handling to ensure graceful termination and informative error messages in case of issues during model solving.

Here are examples of routes and order details, along with the corresponding optimization results. This comprehensive pipeline efficiently processes the data, constructs mathematical models for optimization, and provides insightful solutions for multi-model transportation challenges.

**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