**Homework 3 - ORDER BATCHING PROBLEM**

Alberto Dian\*, Davide Nascivera, Yunju Li  
*\**Group representative

# Problem description

*In the context of warehouse management, the order batching problem arises when a set of orders needs to be picked. This problem involves grouping the orders into batches, considering different seed rules and accompanying rules. The objective is to optimize the order picking process, taking into account batch capacity constraints and a considered performance measure. In the problem offered by the assignment, a set of orders and the location of each item is given. We are going to implement 3 different seed rules, including random order, the largest order and order with furthest COG from depot. For each seed rule, we compare the resulting batches with 2 accompany rules, one is* *Euclidean distance between orders and the other one is**Sum of travel distances (Euclidean) between every location of the order to be added and the closest order line location of the seed. After these, we evaluate the performance of all batches as the distances between order lines in each batch.*

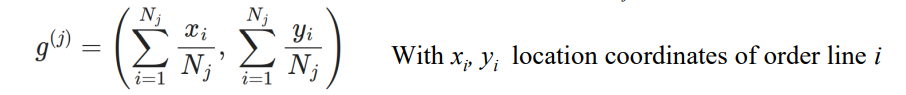
# Mathematical model

*2.1 Input*

1. *Set of Orders: A collection of orders, each consisting of a list of items to be picked from the warehouse.*
2. *Location Matrix: For each item in the warehouse, a matrix indicating its location.*

*2.2 COG*

*COG is used to compute the coordinates of the center of gravity (COG) g(j) of each order j of Nj order lines as:*

**

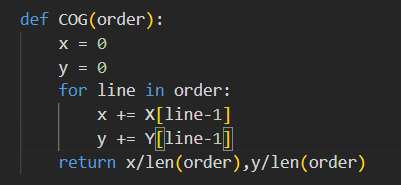
*In this problem, COG is used to implement 2 of the seed rules in which we use COG to proximately represent the location of an order which contains several order lines. Also in the first accompany ruhe, The resulting proximity between two orders is given by the Euclidean distance between their COGs.*

*2.3 constraint*

*In this problem, we make max capacity into consideration. for a single batch, the stopping criterion is the maximum capacity. The next batch is constructed using the same accompany rule.*

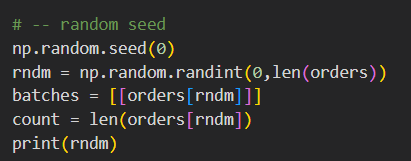
# Main code components

3.1 COG



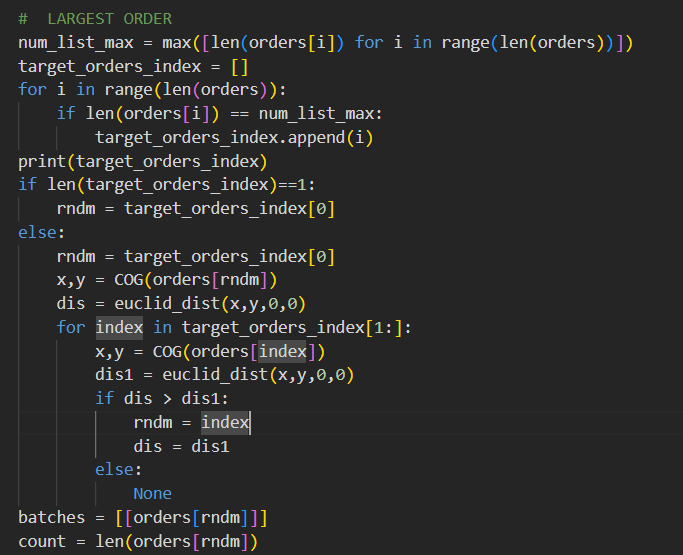
Using the formula mentioned above, this function is to compute the coordinates of the center of gravity of each order. In the problem, we will use it in seed rules 3.3 and 3.4 to compute the distance between the location of the order and the depot. Also in the accompany rule of euclidean distances between roders, COG is needed.

3.2 Seed Rule of Random Order



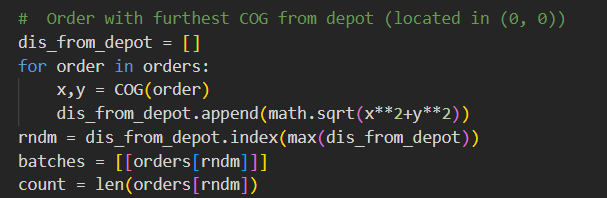
To generate a random number within the indices of orders. We initialize a list named batches, inside the list we have each batch as a list containing every order we add into it. Then add the order we chosoe into the first batch and consume the capacity.

3.3 Seed Rule of Largest Order



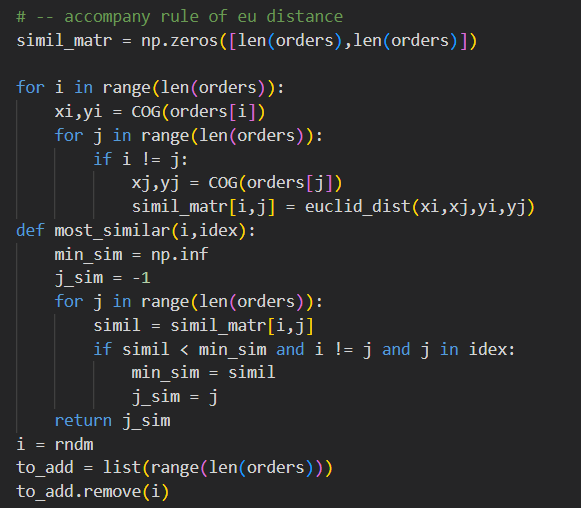
At first find out the orders with largest size. If there are more than one order with the same largest size, choose the order closest to the depot as the seed order We initialize a list named batches, inside the list we have each batch as a list containing every order we add into it. Then add the order we chosoe into the first batch and consume the capacity.

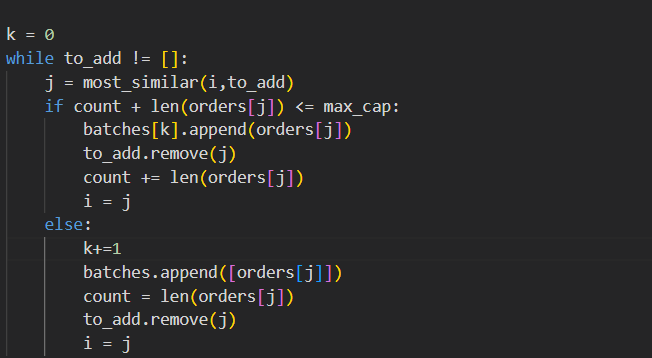
3.4 Seed Rule of Order with Furthest COG From Depot (located in (0, 0))



At first compute the distance between the COG of every order and the depot. Then choose the furthest order as the seed order. We initialize a list named batches, inside the list we have each batch as a list containing every order we add into it. Then add the order we chosoe into the first batch and consume the capacity.

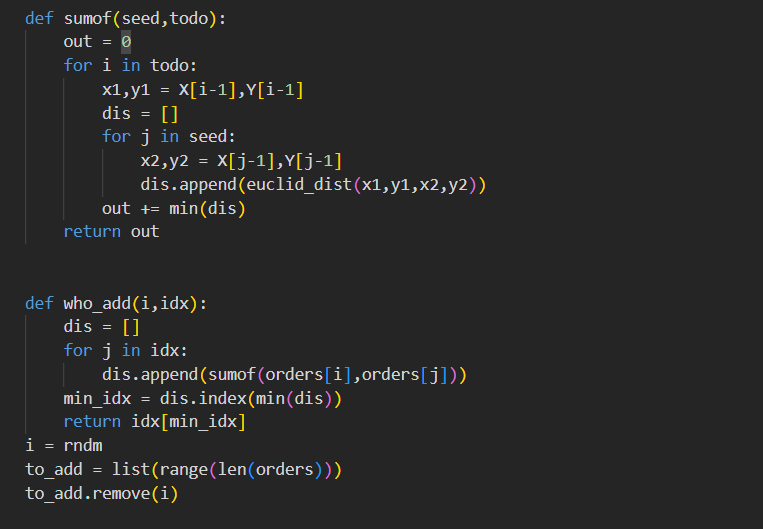
3.5 Accompany Rule of Euclidean Distance

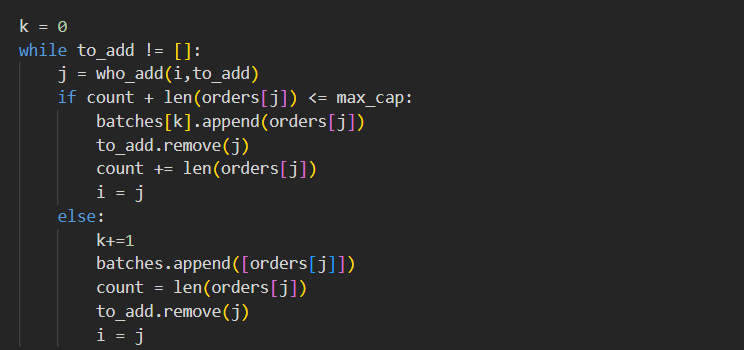




Simil\_matr is a matrix initialized with zeros to store the Euclidean distances between the COGs of orders. The input of the function most\_similar is i(the index of the last order we add into the batch, considered as seed order) and idex(a list of indices of orders to be added). The output is the order we should choose according to the accompany rule from idex. Each time we add an order into the batch, we remove it from the list of orders to add and consume the capacity it costs. If there is no enough capacity for the order we choose, we will open a new batch and add it into it.

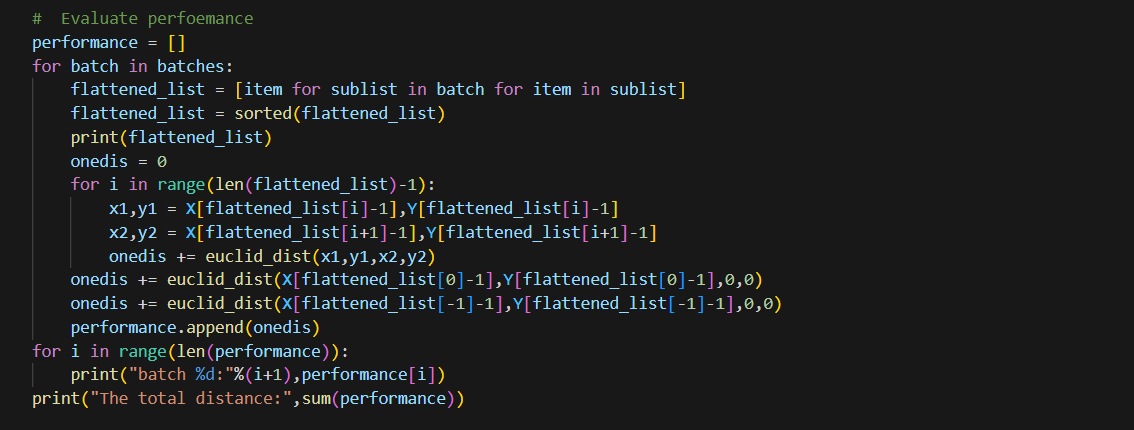
3.6 Accompany Rule of Sum of travel distances (Euclidean) between every location of the order to be added and the closest order line location of the seed





This part is similar to the accompany rule of 3.5. The input of sumof is function is used to compute sum of travel distances (Euclidean) between every location of the order to be added and the closest order line location of the seed. Then in the who\_add function, the input is the index of the seed order and a list of indices of orders to be added. Then we return the index of the most similar order. After that, we initialize the to\_add list as indices of all the orders except for the order chosen by seed rule.Then while the list of orders to be added is not empty, we choose the order from the function who\_add mentioned before. If the available capacity now is enough for this order, we add it into the batch, otherwise, we will put it in a new batch and bagin to pick orders into this new batch. After we add this new order, we remove it from the to\_add list.

3.7 Evaluate Performance



In this part, we create a list to record the performance of each batch at first. The way we evaluate the performance is computing the distances between order lines in each batch, assuming that the order picker starts and ends each batch at the depot and visits order lines following their location number in ascending order. So at first for each batch, it’s flattened because in the list each order is a list. Then we sort the list for computation. In this code, we at first compute the distance between each order and the order after it and add the distance to the variable onedis. Then we add the distance between the first order and depot and the distance between the last order and depot. In this way, we can obtain the performance of each batch.

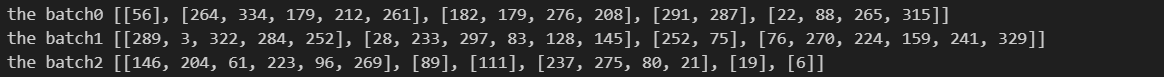
# Results and insights

*For each accompany rule, I implement 3 seed rules. The results are as follows.*

*4.1 Accompany rule of Euclidean distance between orders.*

1. *Seed rule of random order*

*The batches:*

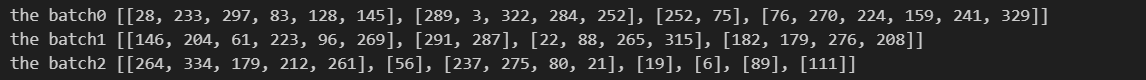
**

*The performance:*

**

1. *Seed rule of largest order*

*The batches*

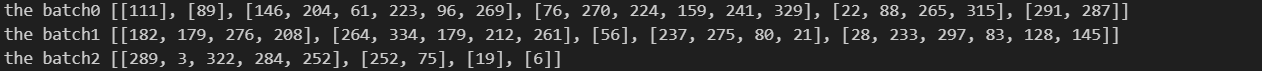
**

*The performance:*

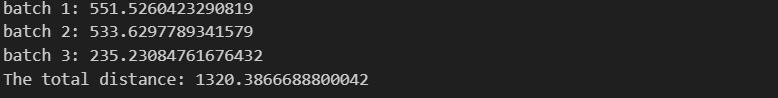
**

1. *Seed rule of furthest order from depot*

*The batches:*

**

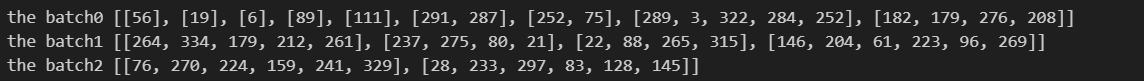
*The performance:*

**

*4.2 Accompany rule of Sum of travel distances (Euclidean) between every location of the order to be added and the closest order line location of the seed*

1. *Seed rule of random order:*

*Batches:*

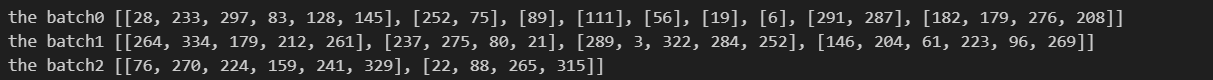
**

*Performance:*

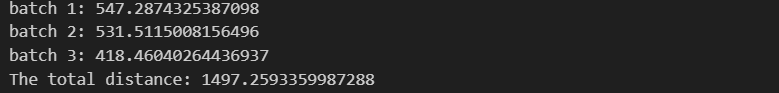
**

1. *Seed rule of largest order:*

*Batches:*

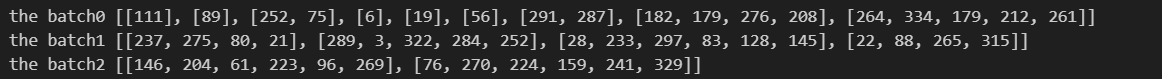
**

*Performance:*

**

1. *Seed rule of furthest order from the depot*

*Batches:*

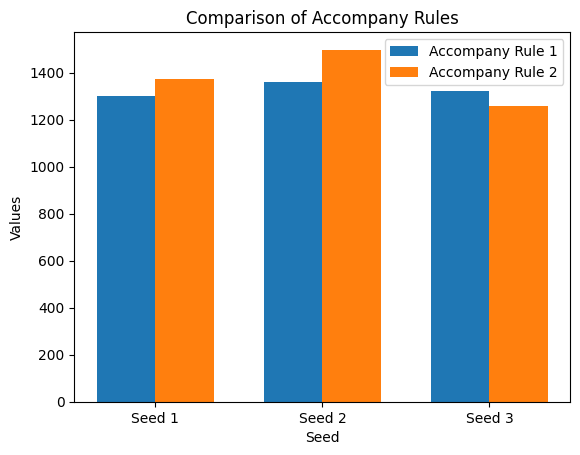
**

*Performance:*

**

*4.3 Visualize the results*

*We can evaluate different combination of accompany rule and seed rule with the sum of performance of each batch, which means the length of the route to pick all the orders. We can visualize the results with a graph as follows.*

**

*At first, for the same seed rule, if we compare the performance of different accompany rules, we can infer from the graph that accompany rule of Euclidean distance between orders has better performance than another one for the first 2 seed rules. And it’s slightly worse than the second accompany rule. So in this example, the accompany rule of Euclidean distance has better performance. A reason for this could be that in the first accompany rule, we choose the closest order by its COG which takes every order lines into consideration. But the second accompany rule has the tendency to choose order with less order lines, this may not lead to minimize the travel distance.*

*For the same accompany rule, compare different seed rules. We see that for different accompany rules, the seed rule that has the better performance is different. And it’s also related to the specific scenario of the problem. But for both accompany rules, the seed rule2 performs the worst. In practice, experiment with various rules using simulation or optimization techniques can help us compare the efficiency of different seed rules under different scenarios and make informed decisions based on the characteristics of our specific order picking operation.*