Risk and Logistics Assignment 2

Wei Li s2346729 Anqi He s2308848 Qiqi Xiao s2323497

March 20, 2023

1 Total Walking Distance Function

Considering different methods to calculate the walking distance, we first divide the order into three data set based on the number of items ordered. In each case, the function will loop over every order and find the shelves where they are placed, and all required distances in sequence. The total distance obtained by summing up the shortest distance for every order.

For orders only including one item, the walking distance can be simply calculated by doubling the distance between entrance and the shelf. We will compare distances if this product group is stored in two different shelves and the shortest one is preferred.

For orders with two items, the path consists of three parts: entrance to the first shelf, first shelf to another shelf and the second shelf to entrance. We can see that distance remains the same regardless of the order so it can be obtained by adding up three distances. Comparison is also needed in this case and the function will take the minimum distance if any group is stored in different shelves.

For orders with three items, the function try to return the shortest distance out of all possible routes. To find all possibilities, we only need to consider which shelf is the second one a picker goes to for collecting products since if the second one fixed, no matter the order of the other two, we will obtain the same distance. Also, if the three items are not all assigned to exactly one shelf, we will compare all the possibilities of picking them up from different shelves.

The last step is convert the distance units into meters, which needs to multiply the result in squares by 3.

Results

The total walking distance of the pickers for the current allocation is 361818 meters.
The running time of the function is 0.7725s.

2 New Shelf Allocation - Random Start Heuristic

We choose to use Random Start heuristic to construct a new shelf allocation from the scratch.

According to the given information, we know that one product group could be allocated in at most two shelves with sufficiently many units and pickers can go to either shelf to collect products from that group. In this case, it is better to make full use of all the shelves by allocating the most ordered products twice.

Firstly, selected the 6 products with the highest order frequency, which would be allocated on two cabinets while the other 84 products will only be allocated once. So, we got a list of item numbers of length 96.

Then, input the item list and find the best allocation in a finite iteration with Random Start heuristic. We created a loop with k iterations and in each loop, the item list would be randomly sorted to 96 shelves and the total walking distance will be calculated by the Total Walking Distance function constructed in question 1. Finally, the best allocation in the loop and its walking distance are obtained. The total walking distance for picker is 339642 meters and the running time of the heuristic is 15.8 seconds.

```
Total walking distance after random start heuristic:  339642
Run time:  15.8 s
```

Figure 1: Results from random start heuristic

3 Local Search Heuristic

In our solution, the basic idea for this part contains 2 steps. The first step is allocating 90 products into different groups based on order information. The second step is change the allocation between products and shelves based on groups which we obtained at the first step. We will describe them specifically in the follow sub parts.

3.1 Step 1: Allocating Products Groups

Our idea is from a classic machine learning model, K-means clustering, and we made some modification. The basic idea is clustering products, calculating its performance, and compare it with last performance. Compare it with last performance and determine if we need to refresh centers and do the progress again.
The specific algorithm steps and some simple pseudo code is below:

```
# Step 1: inputing
Input: Orders, cluster number: n (90 mod cluster number == 0)
# Step 2: calculating the relationship between products
```

```
initialize a 90*90 matrix R
for each Orders:
    if {item1,item2,item3} in same order:
        R[item1,item2],R[item1,item3],R[item2,item3] += 1
        R[item2,item1],R[item3,item1],R[item3,item1] += 1
    if {item1,item2} in same order:
        R[item1,item2],R[item2,item1] += 1
# Step 3: Initilize groups and centers to start
Initialize n random interger in [1,90] as initializing center products
for each centers: # define clusters for initial centers
     find the highest ((90-n)/n) products
     divide them into 1 cluster
this cluster division: this_cluster
for each clusters: # calculate the performance
    cluster_item: including products number in this cluster
    for each products i in this cluster:
        item_i_sum <- sum of R[item_i,item_(cluster_item|i)]
    this_cluster_sum <- sum of item_i_sum

# Step 4: iterating clustering
last_cluster_sum = this_cluster_sum
last_cluster <- this_cluster
this cluster division: this_cluster
for each clusters: # search new center
    cluster_item: including products number in this cluster
    for each products i in this cluster:
        item_i_sum <- sum of R[item_i,item_(cluster_item|i)]
    obtain max(item_i_sum) and corresponding i
    let it be 1 of the new centers
for each new centers: # cluster again within new center
     find the highest ((90-n)/n) products
     divide them into 1 cluster
this cluster division: this_cluster

for each clusters: # calculate the performance
    cluster_item: including products number in this cluster
    for each products i in this cluster:
        item_i_sum = sum of R[item_i,item_(cluster_item|i)]
    this_cluster_sum = sum of item_i_sum

# Step 5: measuring clustering with previous one
```

```
if this_cluster_sum <= last_cluster_sum:
    Output: Last_clsuter
else:
    return to Step 4
```

By upper steps, we can obtain some groups which the containing products have stronger link with each other. This means products in same cluster have higher probability to be in 1 order. This module can be find in the **cluster** function in our python code.

3.2 Step 2: Re-allocating Products and Shelves

Based on the groups we obtained from 1 step, we can do our local search. The basic idea is, first, input the original allocation(shelves-products), the local search sequence and defining rules of neighbor. Along the local search sequence, for each shelves in this sequence, finding its neighbor defined by the input neighbor rules. Finding the product in current shelf and products in neighbor shelves. If those shelves not in same group(obtain by previous step), find the closest shelf which store same group's product, change them. Then, calculating the total distance with after changing allocation, comparing it with total distance calculated by input allocation, if the result improved, save the changing allocation, otherwise, remain the previous allocation. After done these progress for each shelves in sequence, output allocation and total distance.
The specific algorithm steps and some simple pseudo code is below:

```
# Step 1: preparation
Input: alloction, sequence(direction), neighbor rule
best_allocation = alloction
best_distance = TotalDistance(last_allocation)
# Step 2: local search for each shelves in sequence
for each shelf:
    Use last_allocation
    item_i: current product in this shelf
    current_group: group(cluster) of item_i
    neighbor_shelf: this shelfs neighbors defined by neighbor rule
    neighbor_item: current item in neighbor shelf
    neighbor_group: item in neighbor shelves groups(cluster)
    compare: neighor_group and current_group
    if neighor_group != current_group
        N: neighbor shelf whose item not in current group
        C: clothes shelf not in neighbor whose item in current group
        change products in shelf N and C
        this_allocation <- the changing alloction
        if best_distance >= TotalDistance(this_allocation):
            best_allocation <- best_allocation
            best_distance <- TotalDistance(this_allocation)
```

```
22 Output: last allocation, last distance
```

By upper steps, we can obtain a 1 time local search's new allocation and total distance. The upper module can be seen from **clusterchange** function in our code. We have a huge flexibility to use this module to get a better solution, as we can define a searching range, sequence and rules' of neighborhood. Besides, we can use it more than 1 time to find a better solution with different sequence, neighborhood and even different groups(clusters) in step 1. We will show some results in the follow section.

3.3 Results From Our Algorithm

Remind that, in our algorithm, each times' clustering start from a random center, therefore, the result may not be the same if you run same code again. But you can see a obvious improving from result.

Notification that when running the algorithm, choose carefully for the number of groups. As if the number of groups is too big, elements in the same group is less. (e.g. 30 groups, only 3 elements in 1 group) It may lead to a error in step 2 that for 1 shelf, its neighbors' corresponding items may cover all its group members. So we cannot find a shelf to change for its neighbor with item not in same group.

3.3.1 Result: Single Iteration

With choosing 3 groups in step1 and the search sequence is from product 1 to product1 50, than products 90 to product 1. (The sequence and direction can be change, like $\{1, 3, 5..89\}, \{90, 88...2\}$). Here is the result for single running with giving data:

```
Total distance before is 361818
Run time:  63.795832334 s
Total distance is 349062
```

Figure 2: Single Running

3.3.2 Result: Multi Iteration

In multi iteration, you can set your own rule to fluctuate different trying and see if we can get a improved distance. Here, at each iteration, we use different clusters each times and run single iteration until the total distance not decrease.

The result for given allocation is:

The result for Q2 allocation is:

```
Total distance before interchange heuristic is 361818
Total distance after interchange heuristic is 319746
Run time:  446.62663299999997 s
```

Figure 3: Results after interchange heuristic

```
Total distance before interchange heuristic is 339642
Total distance after interchange heuristic is 339641
Run time:  74.72714791699991 s
```

Figure 4: Results after interchange heuristic

4 Products Allocation in the New Warehouse Layout

4.1 Floor Plan Rearrangement

Assumption:

• Our shelves will be primarily placed in the lower left corner. As the entrance is located in the lower left corner, positioning the shelves closer to the lower left corner will result in shorter walking distance for the pickers in total while the distance between each cabinet is fixed.

• Introducing a certain number of horizontal corridors can effectively mitigate the minimum walking distance for pickers. As such, we have devised a plan to incorporate two additional horizontal corridors in order to facilitate more pathways for pickers.

Taking into consideration the aforementioned assumptions, we have developed the following plan for the layout of the warehouse shelves.

4.2 New Distance Matrix

Utilizing the principles of the Breadth-first search algorithm, we have performed calculations to determine the distance between each shelf in the new warehouse layout, and subsequently generated a comprehensive distance matrix. Please refer to the source code provided for detail.

4.3 New Shelf Allocation Generated via Random Start Heuristic with New Floor Plan

We employed the Random Start heuristic to create a novel shelf allocation from the new layout. Figure 3 illustrates a shelf allocation generated through a single implementation of the Random Start heuristic. Total walking distance after random heuristic is 247770 meters.The total running time is 15.5 seconds approximately.

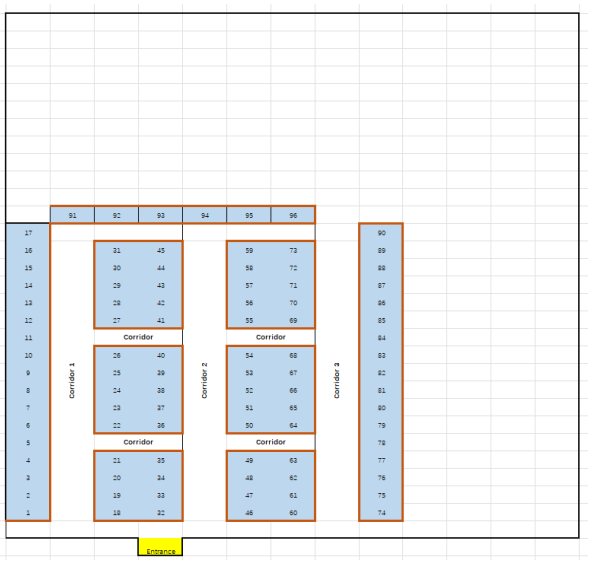

Figure 5: The New Floor Plan

```
Total walking distance after random start heuristic:  247770
Run time:  15.5 s
```

Figure 6: Results from Random Start heuristic on new floor plan

4.4 Improved Shelf Allocation Generated via Interchange Heuristic with New Floor Plan

We improved the current allocation by utilizing the Interchange heuristic and Figure 3 depicts a new allocation obtained from a single implementation of the Interchange heuristic. Total walking distance is 247770 meters before interchange heuristic and 247769 meters after. The total running time is 74.4 seconds approximately.

```
Total distance before interchange heuristic is 247770
Run time:  74.39835925 s
Total distance after interchange heuristic is 247769
```

Figure 7: Results from Interchange heuristic on the new floor plan