Implementing the Heuristics on the Fishbone Layout

In [ ]:
```
# Import the required packages
# Import packages
import numpy as np
import xpress as xp
import pandas as pd
import time
from functions import allocator
from functions import order_modifier
from functions import swaper
from shorthest_path import shortest_path
from shorthest_path import distance
from collections import Counter
```

In [ ]:
```
# Import data file of distances
distances_df = pd.read_excel('Distance Matrix 1.xlsx', sheet_name = "Shee

# multiply the dist
ind = distances_df.loc[:, "Index"]
distances_df = distances_df.loc[:, distances_df.columns != 'Index'].multi
distances_df = pd.concat([ind, distances_df], axis = 1)
# # Load in the data file of orders
orders = pd.read_excel('OrderList.xlsx', sheet_name = "Orders")

# Drop the order numbers
orders = orders.drop(columns = "Order No.")
```

In [ ]:
```
# Run the test

# Define start time
start = time.time()

# Run the function for the above order list (only first 20 for computatio
tot_distance = distance(orders, distances_df)

# Define end time
end = time.time()

# Print outcome
print()
print(f"The total distance is {tot_distance} metres")
print(f"The run time of the function is {round(end - start, 4)} seconds")
```

In [ ]:
```
# Run the construction heuristic
new_allocation = allocator(orders, distances_df)
new_allocation = new_allocation.loc[:, "Product":"Occurance"]
new_allocation = pd.concat([pd.DataFrame({"Shelve" : list(range(1,91))}),
new_ord = order_modifier(orders, new_allocation)

start = time.time()
tot_distance = distance(new_ord, distances_df)
end = time.time()

print(f"The total distance travelled after the construction heuristic is
print(f"The run time is {end - start}")
```

In [ ]:
```
# Run the local search heuristic
orders_with_distance = new_ord.copy()
orders_with_distance['distance'] = 0
for i in range(len(orders_with_distance)):
    dist = shortest_path(new_ord.iloc[i].values.tolist(),distances_df)
    orders_with_distance.loc[i,'distance'] = dist[0]

longest = orders_with_distance[orders_with_distance['distance']==120].sor

total_items = []
for order_list in longest.drop(columns=['distance']).values:
    for item in order_list:
        total_items.append(item)
```

In [ ]:
```
count = dict(Counter(total_items))
sorted_count = dict(sorted(count.items(),key=lambda item: item[1], revers
print(sorted_count)
```

In [1]:
```
initial_distance = 203595.0
a = 54
imp = {}
for i in range(96):
    if i+1 != a:
        alloc, orders2 = swaper(new_allocation,a,i+1)
        new_distance = distance(orders2,distances_df)
        if new_distance < initial_distance:
            print(new_distance)
            print(i+1)
            break
```

In [ ]:
```

```