Implementing Heuristics

Note that, to be able to create a pdf and upload the ipynb file, the output has been removed for the cells. This code can be run to produce the outputs given in the overleaf.

In [ ]:
```
import pandas as pd
import openpyxl
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
import numpy as np

import shorthest_path as sp
import importlib
importlib.reload(sp);
```

In [ ]:
```
orders = pd.read_excel("OrderList.xlsx")

orders.reset_index(inplace=True)

orders.set_index(orders['Order No.'],inplace=True)
orders = orders.drop('Order No.',axis=1)
orders.reset_index(inplace=True)
orders = orders.drop(['Order No.','index'],axis=1)
orders
```

In [ ]:
```
total_items = []
for order_list in orders.values:
    for item in order_list:
        total_items.append(item)
total_items
```

In [ ]:
```
from collections import Counter
count = dict(Counter(total_items))
del count[0]
sorted_count = dict(sorted(count.items(),key=lambda item: item[1], revers
sorted_count
```

In [ ]:
```
bin_df = pd.DataFrame.from_dict(sorted_count, orient='index').reset_index
bin_df.columns = ['Product','Occurance']
bin_df['cum_sum'] = bin_df['Occurance'].cumsum()
bin_df['cum_sum_perc'] = bin_df['Occurance'].cumsum() / bin_df['Occurance

conditions = [
    bin_df['cum_sum_perc'] < 0.60,
    (bin_df['cum_sum_perc'] >= 0.60) & (bin_df['cum_sum_perc'] <= 0.85),
    bin_df['cum_sum_perc'] > 0.85
]
# Define the values to assign for each condition
values = ['royalblue','cornflowerblue','lightsteelblue']
bin_df['Bin'] = np.select(conditions, values)

bin_df.head()
```

In [ ]:
```
fig, ax1 = plt.subplots()
ax1.bar(bin_df.index,height=bin_df['Occurance'],color=bin_df['Bin'])
ax2 = ax1.twinx()
ax2.plot(bin_df.index,bin_df['cum_sum_perc'],color='darkblue')
ax1.set_title('ABC Analysis')
ax1.set_xlabel('Product')
ax1.set_ylabel('Product Demand')
ax2.set_ylabel('Cumulative Frequency %')
plt.show()
```

In [ ]:
```
bin_df.groupby(by='Bin').count()
```

In [ ]:
```
distances = pd.read_excel("DistanceMatrix.xlsx")
distances.columns = distances.iloc[1]
distances = distances[2:]
# distances.set_index('NaN',inplace=True)
# distances[['Packaging']]
# distances.iloc[-1,:]
dist_from_pack = distances.iloc[-1]
distance_df = pd.DataFrame(dist_from_pack).reset_index()
distance_df.columns = ['Shelve','Distance']
distance_df = distance_df[1:]
distance_df = distance_df[:-1]
distance_df = distance_df.sort_values(by="Distance",ascending=True)
distance_df.reset_index(inplace=True)
distance_df.head(26)
```

In [ ]:
```
new_allocation = distance_df.merge(bin_df,left_index=True, right_index=Tr
# new_allocation.sort_values(by='Shelve',inplace=True)
new_allocation.head(25)
```

In [ ]:
```
# 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.")

# Display a snippet of the data frame
orders
```

In [ ]:
```
equivalent_values_dict = new_allocation.set_index('Product')['Shelve'].to
equivalent_values_dict
```

In [ ]:
```
positions = ['Position 1','Position 2','Position 3','Position 4','Positio
for col in positions:
    orders[col] = orders[col].map(equivalent_values_dict)
    orders[col] = orders[col].astype('Int64')
orders = orders.fillna(0)
orders.head(25)
# orders.to_excel("orders_2.xlsx")
```

In [ ]:
```
distances_data = pd.read_excel('DistanceMatrix.xlsx', sheet_name = "Dista
sp.distance(orders,distances_data)
```

In [ ]:
```
def allocator(orders,distance):
    total_items = []
    for order_list in orders.values:
        for item in order_list:
            total_items.append(item)

    count = dict(Counter(total_items))
    del count[0]
    sorted_count = dict(sorted(count.items(),key=lambda item: item[1], re

    bin_df = pd.DataFrame.from_dict(sorted_count, orient='index').reset_i
    bin_df.columns = ['Product','Occurance']

    new_allocation = distance_df.merge(bin_df,left_index=True, right_inde

    return new_allocation

def order_modifier(orders,new_allocation):
    equivalent_values_dict = new_allocation.set_index('Product')['Shelve'
    new_ord = orders.copy()
    positions = ['Position 1','Position 2','Position 3','Position 4','Pos
    for col in positions:
        new_ord[col] = new_ord[col].map(equivalent_values_dict)
        new_ord[col] = new_ord[col].astype('Int64')
    new_ord = new_ord.fillna(0)

    return new_ord

def swaper(allocation,a,b):
    # 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.")

    # a = np.random.randint(allocation['Shelve'].min(), high=allocation['
    # b = np.random.randint(allocation['Shelve'].min(), high=allocation['
    # while a==b:
    #     b = np.random.randint(allocation['Shelve'].min(), high=allocati

    product_a = allocation['Product'][allocation['Shelve']==a].values[0]
    product_b = allocation['Product'][allocation['Shelve']==b].values[0]

    allocation['Product'][allocation['Shelve']==a] = product_b
    allocation['Product'][allocation['Shelve']==b] = product_a

    orders2 = order_modifier(orders,allocation)

    return allocation, orders2
```

In [ ]:
```
orders
```

In [ ]:
```
# 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.")


distances = pd.read_excel("DistanceMatrix.xlsx")
distances.columns = distances.iloc[1]
distances = distances[2:]
# distances.set_index('NaN',inplace=True)
# distances[['Packaging']]
# distances.iloc[-1,:]
dist_from_pack = distances.iloc[-1]
distance_df = pd.DataFrame(dist_from_pack).reset_index()
distance_df.columns = ['Shelve','Distance']
distance_df = distance_df[1:]
distance_df = distance_df[:-1]
distance_df = distance_df.sort_values(by="Distance",ascending=True)
distance_df.reset_index(inplace=True)
distance_df.head(26)

new_alloc = allocator(orders,distance_df)
```

In [ ]:
```
# 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.")
orders
```

In [ ]:
```
new_order = order_modifier(orders,new_alloc)
new_order
```

In [ ]:
```
allocation = new_alloc.sort_values(by="Shelve")
# allocation[allocation['Product']==30]
allocation.head(30)
```

In [ ]:
```
distances_data = pd.read_excel('DistanceMatrix.xlsx', sheet_name = "Dista
```

In [ ]:
```
import shorthest_path as sp
sp.distance(new_order,distances_data)
```

In [ ]:
```
alloc, orders2 = swaper(allocation)
```

In [ ]:
```
new_order
distances_data = pd.read_excel('DistanceMatrix.xlsx', sheet_name = "Dista
distances_data
orders_with_distance = new_order.copy()
orders_with_distance['distance'] = 0
for i in range(len(orders_with_distance)):
    dist = sp.shortest_path(new_order.iloc[i].values.tolist(),distances_d
    orders_with_distance.loc[i,'distance'] = dist[0]
orders_with_distance
```

In [ ]:
```
longest = orders_with_distance[orders_with_distance['distance']==168].sor

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


from collections import Counter
count = dict(Counter(total_items))
del count[0]
sorted_count = dict(sorted(count.items(),key=lambda item: item[1], revers
sorted_count
```

In [ ]:
```
new_order
```

In [ ]:
```
import shorthest_path as sp
initial_distance = 201450
a = 49
imp = {}
for i in range(96):
    if i+1 != a:
        alloc, orders2 = swaper(allocation,a,i+1)
        new_distance = sp.distance(orders2,distances_data)
        if new_distance < initial_distance:
            print(new_distance)
            print(i+1)
            break
```

In [ ]:
```

```

In [ ]:
```
orders = pd.read_excel("OrderList.xlsx")

orders.reset_index(inplace=True)

orders.set_index(orders['Order No.'],inplace=True)
orders = orders.drop('Order No.',axis=1)
orders.reset_index(inplace=True)
orders = orders.drop(['Order No.','index'],axis=1)
orders
```

In [ ]:
```
orders
distances_data = pd.read_excel('DistanceMatrix.xlsx', sheet_name = "Dista
orders_with_distance = orders.copy()
orders_with_distance['distance'] = 0
for i in range(len(orders_with_distance)):
    dist = sp.shortest_path(orders.iloc[i].values.tolist(),distances_data
    orders_with_distance.loc[i,'distance'] = dist[0]
orders_with_distance
```

In [ ]:
```
orders_with_distance.sort_values(by='distance',ascending=False)
```

In [ ]:
```
longest = orders_with_distance[orders_with_distance['distance']==168].sor

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


from collections import Counter
count = dict(Counter(total_items))
del count[0]
sorted_count = dict(sorted(count.items(),key=lambda item: item[1], revers
sorted_count
```

In [ ]:
```
import shorthest_path as sp


distances = pd.read_excel("DistanceMatrix.xlsx")
distances.columns = distances.iloc[1]
distances = distances[2:]
dist_from_pack = distances.iloc[-1]
distance_df = pd.DataFrame(dist_from_pack).reset_index()
distance_df.columns = ['Shelve','Distance']
distance_df = distance_df[1:]
distance_df = distance_df[:-1]
distance_df = distance_df.sort_values(by="Distance",ascending=True)
distance_df.reset_index(inplace=True)



allocation = allocator(orders,distance_df)
allocation = distance_df.copy()
allocation['Product'] = allocation['Shelve']
allocation = allocation[['Shelve','Product']]
allocation


initial_distance = 269502
a = 71
imp = {}
for i in range(96):
    if i+1 != a:
        alloc, orders2 = swaper(allocation,a,i+1)
        new_distance = sp.distance(orders2,distances_data)
        if new_distance < initial_distance:
            print(new_distance)
            print(i+1)
            break
```