과 제

보고서 및 논문 윤리 서약

1. 나는 보고서 및 논문의 내용을 조작하지 않겠습니다.
2. 나는 다른 사람의 보고서 및 논문의 내용을 내 것처럼 무단으로 복사하지 않겠습니다.
3. 나는 다른 사람의 보고서 및 논문의 내용을 참고하거나 인용할 시 참고 및 인용 형식을 갖추고 출처를 반드시 밝히겠습니다.
4. 나는 보고서 및 논문을 대신하여 작성하도록 청탁하지도 청탁받지도 않겠습니다.

나는 보고서 및 논문 작성 시 위법 행위를 하지 않고, 명지인으로서 또한 공학인으로서 나의 양심과 명예를 지킬 것을 약속합니다.

과 목 명 : 산업경영알고리즘

담당교수 : 김 경 민

제 출 일 : 2023. 06. 18

학 번 : 60182466

이 름 : 김한솔

목차

1. 데이터 전처리

각 주문번호별로 한 묶음을 가정하기에 데이터 전처리 과정이 가장 우선적으로 필요하다고 생각하여, 전처리를 진행 후 모형화를 진행하여 가장 처음으로 가져왔다. 각 주문번호에 대해서 아이템은 1 개부터 5 개 이하로 존재한다. 각각의 주문에는 주문자가 주문한 상품이 모두 들어가게끔 배송을 해야 하기에 각 주문번호별로 장, 폭, 고가 최소가 되는 방법을 생각했다.

주문번호별 여러 상품이 있는 경우 각각의 아이템에 대해 장, 폭, 고가 존재하지만 3 차원으로 되어 있어 각각의 아이템은 길이가 너비, 높이가 될 수 있고 너비와 높이 또한 길이와 너비 그리고 높이가 될 수 있다. 이 생각을 통하여 각각의 주문 번호에 대해 필요한 표준 박스를 개별적으로 구해봤다.

방법은 다음과 같다.[1]

- 장 : 첫번째 아이템의 가장 큰 길이를 선택하고 두번째 아이템 이후부터 가장 짧은 길이를 선택 후 총합으로 장을 구하였다.
- 폭 : 첫번째 아이템의 두번째 큰 길이를 선택하고 두번째 아이템 이후부터 장에서 선택하지 않은 가장 짧은 길이를 선택 후 총합으로 폭을 구했다.
- 고 : 첫번째 아이템의 가장 짧은 길이를 선택하고 두번째 아이템 이후부터 장, 폭에서 선택하지 않은 남은 길이를 선택 후 총합으로 폭을 구했다.

위와 같은 방법으로 선택된 길이는 후에 선택에서 제거하는 방법으로 필요한 최소 박스 크기를 구하였다.

위의 방법을 통해 각 주문번호 별 필요한 표준박스의 장, 폭, 고를 알 수 있었으며, 이를 통해 5 단위의 제약을 가진 표준박스의 크기를 알 수 있었다. 5 단위 표준박스의 부피 계산을 통해 어떤 부피가 가장 많이 나오는지 확인할 수 있었고, 후보 표준박스를을 알 수 있었다. 구체적인 전처리 방법은 4. 나만의 알고리즘 전처리 단계[1]에서 통해 설명하겠다.

[1] 4.2 데이터 전처리

2. 혼합정수계획 모형 및 Gurobi 코드

```
if __name__=="__main__":
    # Parameters
    k = 3               # CBM의 개수
    c = candi_volume    # 5단위로 만든 표준 박스의 부피(계수)
    p = wdh             # 주문번호에 해당하는 아이템셋의 부피

    # Model
    model = Model()

    # Decision Variables
    y = model.addVars(len(c), vtype=GRB.BINARY, name="pick")        # 표준 박스 i가 사용 되는지
    x = model.addVars(len(c),len(p), vtype=GRB.BINARY, name="box")  # 표준 박스 i에 주문 번호 j가 들어갈 수 있는지

    # Objective function  → 표준 박스 i의 부피 최소화
    model.setObjective(quicksum(c[i]*y[i] for i in range(len(c))), GRB.MINIMIZE)

    # Constraints
    # 표준 박스 i의 개수는 k개
    model.addConstr(quicksum(y[i] for i in range(len(c))) == k)
    # 총 가능한 경우의 수는 주문 번호를 만족해야함
    model.addConstr(quicksum(quicksum(x[i, j] for j in range(len(p))) for i in range(len(c))) == 50)
    # 표준박스 i에 가능한 주문번호 j의 합은 50을 넘길 수 없음, 안 들어가는 것은 가능
    for i in range(len(c)):
        model.addConstr(quicksum(x[i,j] for j in range(len(p))) <= 50)
    # 사용되는 표준박스 i는 가능한 표준박스 i의 주문번호에 해당하는 부피보다 커야함
    for i in range(len(c)):
        model.addConstr(c[i]*y[i] - quicksum(p[j][0]*p[j][1]*p[j][2]*x[i,j] for j in range(len(p))) >= 0)

    model.write('IMA project.lp')
    model.optimize()
    if model.Solcount > 0:
        model.printAttr('X')
```

파라미터

k = 표준박스의 개수

c = 주어진 데이터로 구한 5 단위로 만든 표준박스 후보군의 부피

p = 주문번호에 해당하는 아이템 묶음의 장(=$p[j][0]$), 폭(=$p[j][1]$), 고(=$p[j][2]$)

변수

$y[i]$: 표준 박스 i 가 사용되는지에 대한 여부 (이진변수) $(i = 1,2,\cdots,c)$

$x[i,j]$: 표준 박스 i 에 주문번호 j 가 들어가는지에 대한 여부 (이진변수) $(i = 1,2,\cdots,c), (j = 1,2,\cdots,p)$

목적함수

$$Min \sum_{i=1}^{32} c[i] * y[i]$$

제약식

1. 선택한 표준박스의 개수의 총합은 k 개가 되어야한다.

$$\sum_{i=1}^{c} y[i] = k$$

2. 총 가능한 경우의 수는 모든 주문번호를 만족해야한다.

$$\sum_{i=1}^{c}\sum_{j=1}^{p} x[i,j] = 50$$

3. 표준박스 i 에 안들어가는 것은 가능하지만, 표준박스 i 에 가능한 주문번호 j 의 합은 50 을 넘길 수 없다.

$$\sum_{i=1}^{c}(\sum_{j=1}^{p} x[i,j]) \leq 50$$

4. 사용되는 표준박스 i 는 가능한 표준박스 i 의 주문번호에 해당하는 부피보다 커야한다.

$$\sum_{i=1}^{c}\{c[i] * y[i] - \sum_{j=1}^{p}(p[j][0] * p[j][1] * p[j][2] * x[i,j])\} \geq 0$$

$$y[i], x[i,j] \in \{0,1\}$$

$$(i = 1,2,\cdots,c), (j = 1,2,\cdots,p)$$

계산 시간 및 로그 분석

K=3 일 때 계산 시간 및 로그 분석이다.

```
Optimize a model with 66 rows, 1632 columns and 4864 nonzeros
Model fingerprint: 0x6ca7a268
Variable types: 0 continuous, 1632 integer (1632 binary)
Coefficient statistics:
  Matrix range     [1e+00, 4e+05]
  Objective range  [6e+02, 4e+05]
  Bounds range     [1e+00, 1e+00]
  RHS range        [3e+00, 5e+01]
Presolve removed 32 rows and 869 columns
Presolve time: 0.00s
Presolved: 34 rows, 763 columns, 1526 nonzeros
Variable types: 0 continuous, 763 integer (709 binary)
Found heuristic solution: objective 603625.00000
```

변수는 총 0 개의 연속형 변수와 1632 개의 정수형(이진변수)가 존재한다.

Presolving 을 통해 32 개의 행과 869 개의 열이 지워져 총 34 개 행과 763 개 열로 계산되었다.

휴리스틱을 통해 찾은 목적함수 값은 603525 이다.

```
    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 12743.0769    0   34 603625.000 12743.0769  97.9%     -    0s
H    0     0                      183875.00000 13368.2210  92.7%     -    0s
     0     0 24564.7094    0   76 183875.000 24564.7094  86.6%     -    0s
     0     0 35083.5130    0   73 183875.000 35083.5130  80.9%     -    0s
     0     0 38589.9519    0  105 183875.000 38589.9519  79.0%     -    0s
     0     0 38810.2705    0  105 183875.000 38810.2705  78.9%     -    0s
     0     2 38810.2705    0  105 183875.000 38810.2705  78.9%     -    0s
H   30    35                      175875.00000 46211.8165  73.7%   3.6    0s
H   34    35                      151875.00000 49062.1877  67.7%   3.5    0s
H   70    62                      124750.00000 53090.2322  57.4%   3.2    0s
H   77    62                      124125.00000 53090.2322  57.2%   3.3    0s
H   85    62                      104500.00000 53090.2322  49.2%   4.8    0s
H  107    79                      68125.000000 53090.2322  22.1%   4.7    0s
H  116    79                      67500.000000 53090.2322  21.3%   5.5    0s
H  182    74                      66500.000000 55633.7462  16.3%   5.8    0s
H  464    93                      65500.000000 63520.8044  3.02%   5.4    0s
```

휴리스틱으로 찾은 가능해는 총 10 개며 목적함수가 점점 커지는 것을 통해 목적함수를 최소화 하는 문제임을 알 수 있다. 가장 마지막에 존재하는 열이 guobi 를 통해 얻은 가능해이며, 65500 의 값을 갖는다. 탐험한 노드의 수는 464 개 탐험하지 않은 노드의 수는 93 개다. 심플렉스 횟수는 5.4 회며 이전 가능해와 계산 결과차이는 3.02% 차이가 난다.

```
Optimal solution found (tolerance 1.00e-04)
Best objective 6.550000000000e+04, best bound 6.550000000000e+04, gap 0.0000%

    Variable           X
-------------------------
     pick[4]           1
     pick[5]           1
    pick[31]           1
```

K=3 일 때, 변수의 값은 pick[4], pick[5], pick[31] 로 얻어졌다. 구로비에서는 결정변수의 아래 첨자가 0 부터 시작하기에 5 번째, 6 번째, 32 번째 부피를 가지는 해가 선택되었다고 할 수 있다. 즉, 장, 폭, 고가 **(35, 35, 25), (30, 30, 30), (40, 35, 25)** 가 되는 해를 얻었다고 할 수 있다.

1: (30, 10, 10),
2: (55, 50, 50),
3: (35, 40, 30),
4: (35, 35, 25),
5: (30, 30, 30),
6: (30, 25, 35),
7: (20, 20, 5),
8: (30, 25, 15),
9: (25, 5, 5),
10: (25, 10, 5),
11: (25, 15, 10),
12: (25, 15, 10),
13: (20, 20, 15),
14: (30, 35, 55),
15: (25, 10, 10),
16: (35, 20, 20),
17: (30, 30, 35),
18: (40, 25, 30),
19: (15, 10, 10),
20: (50, 40, 35),
21: (20, 15, 15),
22: (45, 45, 45),
23: (30, 25, 30),
24: (50, 75, 100),
25: (25, 20, 10),
26: (30, 30, 40),
27: (45, 25, 5),
28: (20, 10, 5),
29: (40, 45, 45),
30: (35, 20, 15),
31: (40, 35, 25),
32: (35, 10, 35

3. 표준박스 개수에 따른 CBM 및 적재율 변화 분석

```
Optimal solution found (tolerance 1.00e-04)
Best objective 6.550000000000e+04, best bound 6.550000000000e+04, gap 0.0000%

    Variable           X
-------------------------
     pick[4]           1
     pick[5]           1
    pick[31]           1
```

K=3

```
Optimal solution found (tolerance 1.00e-04)
Best objective 5.650000000000e+04, best bound 5.650000000000e+04, gap 0.0000%

    Variable           X
-------------------------
     pick[7]           1
    pick[22]           1
    pick[29]           1
    pick[31]           1
```

K=4

```
Optimal solution found (tolerance 1.00e-04)
Best objective 4.912500000000e+04, best bound 4.912500000000e+04, gap 0.0000%

    Variable           X
-------------------------
     pick[7]           1
    pick[11]           1
    pick[15]           1
    pick[26]           1
    pick[31]           1
```

K=5

```
Optimal solution found (tolerance 1.00e-04)
Best objective 4.537500000000e+04, best bound 4.537500000000e+04, gap 0.0000%

    Variable           X
-------------------------
     pick[0]           1
    pick[11]           1
    pick[15]           1
    pick[20]           1
    pick[26]           1
    pick[31]           1
```

K=6

```
Best objective 4.200000000000e+04, best bound 4.200000000000e+04, gap 0.0000%

    Variable           X
-------------------------
     pick[0]           1
     pick[6]           1
     pick[7]           1
    pick[12]           1
    pick[20]           1
    pick[26]           1
    pick[31]           1
```

K=7

표준박스의 개수(K)가 증가함에 따라 목적함수가 점점 작아지는 것을 확인할 수 있다.

4. 표준박스의 개수 및 크기를 결정하는 나만의 알고리즘

이 문제는 특정한 정답이 존재하지 않는 데이터를 가지고 최선의 표준박스의 개수와 크기를 찾는 문제다. 그렇기에 데이터를 전처리하는 방법에 따라서 결과의 차이가 분명히 존재한다. 표준박스의 크기에 대한 비용이 존재하지 않는 상황을 가정하기에 문제에서 주문번호 별로 하나의 표준박스를 사용하는 것을 무조건 만족해야 한다는 가정을 세웠다. 데이터 전처리과정 역시 가정에 만족하도록 하여 1. 데이터 전처리와 같은 방법을 사용하였다.

데이터의 정답이 없는 경우로 가장 먼저 비지도학습이 떠올랐다. 어떤 탐욕적 알고리즘을나의 생각대로 짜는 것도 의미가 있지만, 기존에 흥미있던 머신러닝과 데이터 분석 과정을 활용해 볼 수 있을 것 같아 머신러닝 비지도학습 중 K-Means Clustering 을 선택하여 진행하기로 해보았다.

각각의 아이템에 대해서는 규격이 존재하기에 머신러닝을 통해 최적의 표준박스 개수와 크기를 구하는 과정이다.

4.1 필요 라이브러리

데이터에 핸들링을 수행하기 위해 pandas libaray를 import 한다.

K-means 클러스터링을 위해 sklearn libaray를 import한다.

시각화를 위한 matplotilib를 libaray를 import한다

4.2 데이터 전처리

1. data_proprecessing(df)

```
def data_proprecssing(df):
    groups = df.groupby("주문번호")
    dataSet = dict(list(groups))
    matrix = {}
    for i in range(1,len(dataSet)+1):
        row = dataSet[i].values.tolist()
        matrix[i] = row

    result = {}
    for orderNumber, items in matrix.items():
        group = []
        for item in range(len(items)):
            group.append(items[item][1:])
        result[orderNumber] = group
    return result
```

주문 번호 별 각 아이템의 장, 폭, 고를 가져와 나만의 greedy 한 방법으로 각 주문번호별 필요한 표준 박스를 계산하기 위해 필요한 전처리다.

Input : df (=박스규격_최적하_data.xlsx)

Output : 주문번호 별 각 아이템의 장, 폭, 고

2. mymethod(data)

```
def mymethod(data):
    result = {}
    for orderNumber, items in data_proprecssing(df).items():
```

```
        if len(items) > 1:
            while len(items) != 1:
                a = items.pop((items.index(items[0])))
                b = items.pop((items.index(items[0])))
                box = []
                while a != []:
                    val = a.pop(a.index(max(a)))
                    val += b.pop(b.index(min(b)))
                    box.append(val)
                items.append(box)

        result[orderNumber] = items
    return result
```

각 주문번호 마다 하나의 필요한 표준박스의 크기를 하나로 만들기 위해서 사용하였다.

Input : data (=주문번호 별 각 아이템의 장, 폭, 고)

Output : 주문변호 별 필요 박스 크기

3. constraints(orders)

```
def constraints(orders):
    # step 1 : mymethod 으로 얻은 값에 필요한 최소 CBM 찾기 -> 현재 주문에 대해서
5 단위로 구성된 가장 적합한 CBM 찾기
    result_CBM = []
    for item in orders:
        SearchCBM = []
        WDH = orders[item][0]
        # print(WDH)
        for k in range(len(WDH)):
            i = 1
            while 5 * i <= WDH[k]:
                i += 1
            SearchCBM += [5 * i]
        result_CBM.append(tuple(SearchCBM))

    # step 2 : 적정 CBM(titrationCBM) 딕셔너리에 각 주문별 필요한 WDH 매칭
    result = dict()
    for i in range(len(result_CBM)):
        result[i + 1] = result_CBM[i]
    return result
```

mymethod 함수로 얻은 값을 사용하여 제약조건인 5 단위로 구성된 장, 폭, 고를 만족하는 CBM 을 찾고자 올림하는 과정이다.

두 단계로 나눠 값을 찾는 과정과 찾은 값을 다시 주문번호 별로 할당하는 과정이 있다.

Input : orders (=주문변호 별 필요 박스 크기)

Output : 주문변호 별 제약조건(5 단위)을 만족하는 최소 박스 크기

4.3 데이터 탐색

데이터 전처리를 통해서 얻은 값에 대해 50 개의 결과가 존재하며 장, 폭, 고의 값이 달라도 각 주문번호 별로 필요한 표준박스의 크기(부피)가 같을 수도 있다고 생각하였다. 동일 한 부피를 가지는 주문이 몇 개인지 확인해 보았다.

```
def volumePerOrder(const_order):
    order_volume = dict()
    for i in range(len(const_order)):
        ith_volume = const_order[i+1][0] * const_order[i+1][1] *
const_order[i+1][2]
        order_volume[i+1] = ith_volume
    return order_volume

# 부피별로 개수 카운트
def candidateVolume(order_volume):
    count = {}
    candi_volume = []
    for i in order_volume.values():
        try:
            count[i] += 1
        except:
            count[i] = 1
        finally:
            if i not in candi_volume:
                candi_volume.append(i)
    count = (sorted(count.items(), key=lambda x: x[1], reverse=True))
    candi_volume = sorted(candi_volume)
    return count, candi_volume
```

위 코드는 데이터 탐색에 사용하기 위한 함수를 작성한 것이며, 각 주문 번호 별 필요 표준박스의 부피를 계산하고 동일한 부피의 개수와 후보가 되는 부피를 알아보았다.

데이터 탐색 결과는 다음과 같았다.

count :

[(3000, 4), (3750, 3), (6000, 3), (2500, 3), (31500, 3), (42000, 2), (30625, 2), (26250, 2), (11250, 2), (3375, 2), (14000, 2), (4500, 2), (137500, 1), (27000, 1), (2000, 1), (625, 1), (1250, 1), (57750, 1), (30000, 1), (1500, 1), (70000, 1), (91125, 1), (22500, 1), (375000, 1), (5000, 1), (36000, 1), (5625, 1), (1000, 1), (81000, 1), (10500, 1), (35000, 1), (12250, 1)]

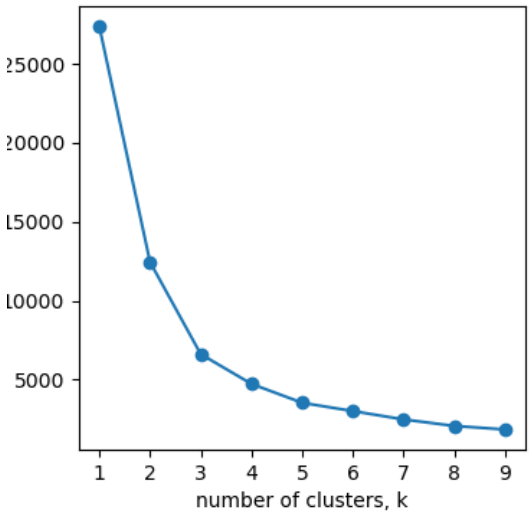
candi_volume :

[625, 1000, 1250, 1500, 2000, 2500, 3000, 3375, 3750, 4500, 5000, 5625, 6000, 10500, 11250, 12250, 14000, 22500, 26250, 27000, 30000, 30625, 31500, 35000, 36000, 42000, 57750, 70000, 81000, 91125, 137500, 375000]

4.4 K-Means Clustering

먼저 적정 K 를 찾기 위해서 클러스터 내의 오차제곱합(SSE)이 최소가 되도록 하는 엘보우기법을 상용했다. K 가 늘어날수록 대략적으로 오류율이 급속히 줄어들다가 어느시점부터 오류율이 거의 감소하지 않는 지점이 발생하는데 이 점을 최적의 K 로 선택하였다. 전처리 과정을 통해 주문번호가 총 50 개지만 1 부터 10 개의 클러스터 후보군을 두었고 최적의 K 는 완만해지기 시작한 3 로 결정하게 되었다. 아래는 K 를 찾는 코드와 결과다.

```
from sklearn.cluster import KMeans

transpose_df = pd.DataFrame(titration_CBM).transpose()
print(transpose_df)

# 적절한 군집수 찾기
ks = range(1, 10)
inertias = []

for k in ks:
    model = KMeans(n_clusters=k)
    model.fit(transpose_df)
    inertias.append(model.inertia_)

# Plot ks vs inertias
plt.figure(figsize=(4, 4))

plt.plot(ks, inertias, '-o')
plt.xlabel('number of clusters, k')
plt.ylabel('inertia')
plt.xticks(ks)
plt.show()
```



다음은 3 개의 군집(표준박스)가 존재할 때 데이터를 통해 학습하여 각 군집의 중심을 통해 각각의 표준박스의 규격을 계산했다. 아래는 코드와 결과다.

```
# 모델 호출
clust_model = KMeans(n_clusters=3, random_state = 23)

# 생성한 모델로 데이터를 학습
clust_model.fit(transpose_df)  # unsupervised learning

# 결과 값을 변수에 저장
centers = clust_model.cluster_centers_  # 각 군집의 중심점
print(pd.DataFrame(centers))
```

clust	장	폭	고
0	25.344828	16.034483	11.206897
1	50.000000	75.000000	100.000000
2	35.500000	33.000000	35.500000

위 K-Means 클러스터링 결과를 토대로 제약조건을 만족하는 표준박스를 구하면 다음과 같은 결과 값을 얻을 수 있다.

표준박스	장	폭	고	부피
0	30	20	15	9000
1	50	75	100	375000
2	40	35	40	56000

5. 결론 및 한계점

Gurobi 와 머신러닝을 통해 각각의 해를 얻을 수 있었다. 결과적으로 어떠한 값을 얻었긴 했지만 가정을 통해 문제를 완화하였다는 점에서 제대로 된 최적해를 구하지 못하였다.

특히 Gurobi 를 사용할 때 추가적인 제약식을 더 존재해야 원하는 결과값을 얻을 수 있다는 것을 알지만 복잡한 수리 모형으로 제대로 완성하지 못했다. 전처리 과정을 가정하여 최적값을 구할 때, 어떠한 k 값을 되던 무조건 가장 큰 장, 폭, 고를 가지는 표준박스가 선택 되야 한다. 하지만 그렇지 못한 부분에서 수리모형의 문제를 알 수 있었다

또한 전처리 과정에서 가장 크게 드러나는 한계점이 있다. 현재 가장 큰 가정으로 주문번호에 대한 하나의 표준박스를 사용해야 한다는 가정이 있다는 문제가 있다. 주문번호에 대해 하나의 표준박스만을 사용하지 않아도 되는 상황이라면 더욱 효과적인 표준박스를 만들 수 있다는 점이 있다. 예로 표준박스 1 번 같은 경우 1 회 사용이 된다. 33 번 주문 같은 경우 때문에 존재한다. 33 번 주문에는 5 가지 아이템이 존재하는데 이 경우, 한 번에 모든 상품을 묶어서 보내는 것보다 2 개의 묶음으로 나눠서 배송하는 것이 표준박스 사이즈를 구하는 측면에서 더 유리하고 경제적이다. 아이템이 4 개 이상 되는 경우 2 개의 묶음으로 더 좋은 표준박스의 규격을 구할 수 있을 것이라고 보인다.

전처리 과정을 수행하면서 주어진 원 데이터가 아닌 제약식을 만족하는 필요한 표준박스를 계산하고 학습시켜 이와 같은 결과를 얻었다. 원 데이터를 학습데이터로 사용하고 위의 방법과 비교해보며 어떤 데이터를 사용하는 것이 성능이 더 나오는지도 확인해 보면 좋을 것 같다.

K-Means clustering 같은 경우 데이터 양이 많으면 많을수록 더 정확한 결과값을 얻는다. 하지만 현재 데이터의 양은 50 개로 매우 협소한 상태에서 학습을 진행하였다. 이에 따라 부정확한 결과값을 얻었을 수 있다. 학습을 통해 얻은 결과값을 실제값과 비교해보면 주문번호에 예측된 표준박스 크기에 넣을 수 없을 수 없는 경우도 존재한다.

이번 프로젝트를 통해 다양한 스킬셋에 대해서 얻어갈 수 있었고, 결과적으로 어떻게 문제 정의를 할까에 대한 고민을 많이 하고 그 만큼 시간을 많이 썼던 것 같다. 프로젝트에 하는 도중, 그 과정 하나하나 돌아보면 Gurobi 사용, 알고리즘 만들어보기 그리고 보고서 작성까지 전반적으로 나의 부족함을 느끼게 되었다. 방학기간 여유가 있을 때 위 문제를 다시 해결해 보면 좋을 것 같다는 생각이 들었다.