MGTA613 Group Assigment Report FINAL

March 12, 2024

By: Owen Petter & Michael Jarvie

Decesion Variables:
Let D_d represent whether or not depot d is opened ($D_d = 1$ means open, $D_d = 0$ means closed).
$d = 1, \ldots, 19$

Let X_{psd} represent the number of units of part p moved from supplier s to depot d.
$p = 1, \ldots, 144$
$s = 1, \ldots, 14$
$d = 1, \ldots, 19$

Let X_{pdcc} represent the number of units of part p moved from depot d to country code cc.
$p = 1, \ldots, 144$
$d = 1, \ldots, 19$
$cc = ij$ where $i = a, b, \ldots, z$ and $j = a, b, \ldots, z$

Suppliers send parts to depots, and then depots send parts to country codes.

Note: some part indices have no demand, such as P_{136}, but our choice to index p from 1 to 144 will not be affected by gaps, and allows us to easily recognize which part class is being discussed. Similarly, the choice of indexing country codes will not be affected by gaps.

Objective Function:
Minimize $F_d \cdot D_d + W_p \cdot M_{sd} \cdot X_{psd} + W_p \cdot M_{dcc} \cdot X_{pdcc}$

where:

F_d = the annualized fixed cost of opening depot d (found in "FixedCost")

M_{sd} = the \$ per kg cost of transporting parts from supplier s to depot d (found in "TransCost IN")

M_{dcc} = the \$ per kg cost of transporting parts from depot d to country code cc (found in"TransCost")

W_p = the kg weight of part p (found in "Weights")

Constraints:
Non-Negativity:

$X_{psd} \geq 0$
$X_{pdcc} \geq 0$

Supplier Capacity:

For each $s = 2, \ldots, 14$:
$\sum_{p,d} W_p \cdot X_{psd} \leq 200,000$
(the weight of all parts shipped out of supplier s cannot exceed 200,000 kg except in the US)

Depots:

For each $d = 2, \ldots, 19$:
$\sum_{p,s,d} X_{psd} \leq D_d \cdot 150,000$
For $d = 1$:
$\sum_{p,s,d} X_{psd} \leq D_d \cdot 300,000$
(cannot ship any units into a depot if the depot is not open, and each depot can only process 150,000 units per year except for the Chicago depot which can process 300,000)
For each $d = 1, \ldots, 19$:
$\sum_{p,s,d} X_{psd} = \sum_{p,d,cc} X_{pdcc}$
(trans-shipment: the number of units of part p shipped into depot d must equal the number of units of part p shipped out of depot d)

Demand:

For each $cc = ij$ where $i = a, b, \ldots, z$ and $j = a, b, \ldots, z$
$\sum_{d} X_{pdcc} = N_{pcc}$
N_{pcc} = the number of units of part p demanded by country code cc (the number of units of part p received by the country code must equal demand)

Service Levels:

$\sum_{p,d,cc} X_{pdcc} \cdot W_p \cdot M_{dcc} \leq X_{pdcc} \cdot W_p \cdot 5$
(Actual cost of receiving the parts $\leq$ Cost of receiving the parts if the $$ $ per kg cost was 5)
This set of constraints ensures that any part received by a country code can only come from a depot within 9 hours flight time of that country code. The issue of whether the depot is open or not is handled by the depot constraints.

[ ]:
```
import gurobipy as gp
from gurobipy import *
import numpy as np
import pandas as pd
import os
```

[ ]:
```
demand = pd.read_csv(r"C:\Users\opett\OneDrive - Wilfrid Laurier␣
↪University\Decision Making with Analytics files\Group Assignment\Demand.
↪csv", header = 0)
demand
```

[ ]:

	Unnamed: 0	AE	AF	AG	AN	AO	AR	\
0	P1	187.676333	1.295667	17.823000	0.072667	3.713333	0.0	
1	P2	16.116667	0.000000	5.785833	2.725333	3.217500	0.0	
2	P3	0.640667	0.000000	0.110833	0.000000	3.114000	0.0	
3	P4	0.176237	0.000000	1.095500	0.000000	1.153333	0.0	
4	P5	0.514205	0.000000	0.000000	0.000000	0.000000	0.0	

```
..          …          …       …          …       …       …  …
122        P140   0.000000  0.000000   0.000000  0.000000  0.000000  0.0
123        P141   0.000000  0.000000   0.000000  0.000000  0.000000  0.0
124        P142   0.000000  0.000000   0.000000  0.000000  0.000000  0.0
125        P143   0.012737  0.000000   0.000000  0.000000  0.000000  0.0
126        P144   0.012737  0.000000   0.000000  0.000000  0.000000  0.0

             AT          AU   AW  …   UA        UG           US        UY  \
0     26.225333  192.034333  0.0  …  0.0  0.115333  7589.301394  0.366167
1      7.691667   15.634000  0.0  …  0.0  0.000000   536.821833  1.563833
2      1.424500    1.015333  0.0  …  0.0  0.000000    53.654167  0.011500
3      0.318500    2.789379  0.0  …  0.0  0.000000    50.350884  0.011500
4      0.364500    1.621491  0.0  …  0.0  0.000000    14.169240  0.000000
..          …           …   …  …  …  …       …           …        …
122    0.000000    0.000000  0.0  …  0.0  0.000000     0.496833  0.000000
123    0.000000    0.000000  0.0  …  0.0  0.000000     0.580000  0.000000
124    0.000000    0.000000  0.0  …  0.0  0.000000     0.011500  0.000000
125    0.000000    0.193692  0.0  …  0.0  0.000000     0.175061  0.000000
126    0.000000    0.169725  0.0  …  0.0  0.000000     0.084862  0.000000

            VE   VG   VN        YE         ZA   ZM
0     1.786667  0.0  0.0  0.110000  40.722000  0.0
1     0.363333  0.0  0.0  0.029333   8.784667  0.0
2     0.000000  0.0  0.0  0.000000   1.295000  0.0
3     0.218000  0.0  0.0  0.011000   0.717500  0.0
4     0.000000  0.0  0.0  0.000000   0.322167  0.0
..         …   …   …       …        …   …
122   0.000000  0.0  0.0  0.000000   0.000000  0.0
123   0.000000  0.0  0.0  0.000000   0.000000  0.0
124   0.000000  0.0  0.0  0.000000   0.000000  0.0
125   0.000000  0.0  0.0  0.000000   0.169725  0.0
126   0.000000  0.0  0.0  0.000000   0.000000  0.0

[127 rows x 120 columns]
```

[ ]:
```
fixed_costs = pd.read_csv(r"C:\Users\opett\OneDrive - Wilfrid Laurier␣
↪University\Decision Making with Analytics files\Group Assignment\FixedCost.
↪csv", header = None)
fixed_costs
```

[ ]:
```
    0       1
0   D1  431365
1   D2  387232
2   D3  272284
3   D4  172474
4   D5  345238
5   D6  353491
```

```
6    D7  349015
7    D8  304631
8    D9  353582
9   D10  399880
10  D11  250825
11  D12  222057
12  D13  180916
13  D14  241095
14  D15  250825
15  D16  355712
16  D17  318570
17  D18  201235
18  D19  399880
```

[ ]:
```
Trans_costs = pd.read_csv(r"C:\Users\opett\OneDrive - Wilfrid Laurier␣
↪University\Decision Making with Analytics files\Group Assignment\TransCost.
↪csv", header = 0)
Trans_costs
```

[ ]:
```
    Unnamed: 0     D1     D2     D3     D4     D5     D6     D7     D8     D9  \
0           AE  5.915  4.343  5.996  4.619  0.872  5.135  6.031  4.354  4.635
1           AF  5.827  4.427  5.874  4.343  3.395  4.738  6.324  4.491  4.658
2           AG  4.030  4.946  6.187  6.976  5.936  6.310  4.348  4.889  4.732
3           AN  4.056  4.369  6.603  7.125  6.121  6.339  4.302  5.104  4.950
4           AO  5.835  4.799  6.279  5.599  4.555  6.349  4.795  4.868  4.971
..         ...    ...    ...    ...    ...    ...    ...    ...    ...    ...
114         VG  3.945  4.967  6.156  6.984  5.968  6.264  4.412  4.908  4.738
115         VN  6.159  4.534  5.086  3.571  4.354  4.039  6.877  5.337  5.456
116         YE  5.980  4.399  6.085  4.758  3.361  5.477  5.673  4.495  4.678
117         ZA  6.303  4.516  5.800  4.514  4.750  6.286  5.013  5.358  5.475
118         ZM  6.234  4.956  5.960  4.478  4.499  6.176  5.098  5.108  4.451

       D10    D11    D12    D13    D14    D15    D16    D17    D18    D19
0    5.706  4.774  4.762  6.442  4.617  4.555  6.165  4.613  4.617  6.098
1    5.655  4.345  4.354  6.145  4.170  4.278  6.001  4.633  4.244  6.093
2    3.926  6.343  6.486  4.582  6.307  6.661  4.093  4.768  6.642  3.550
3    4.046  6.414  6.567  4.408  6.407  6.176  4.021  4.982  6.161  3.378
4    5.676  6.122  6.034  5.342  5.939  5.887  6.067  4.994  5.869  5.676
..     ...    ...    ...    ...    ...    ...    ...    ...    ...    ...
114  3.860  6.307  6.455  4.609  6.280  6.638  3.902  4.780  6.615  3.439
115  6.107  3.642  3.472  7.163  3.610  3.099  6.288  5.421  3.071  6.444
116  5.749  4.343  5.122  6.219  5.003  4.986  6.228  4.678  4.962  6.073
117  6.162  6.074  5.933  5.413  5.920  5.721  6.493  5.489  5.713  6.173
118  6.015  5.947  5.822  5.545  5.775  5.630  6.402  4.474  5.616  6.098

[119 rows x 20 columns]
```

```
[ ]: Trans_cost_IN = pd.read_csv(r"C:\Users\opett\OneDrive - Wilfrid Laurier␣
      ↪University\Decision Making with Analytics files\Group Assignment\TransCostIn.
      ↪csv", header = 0)
     Trans_cost_IN
```

[ ]:

	Unnamed: 0	Chicago	Frankfurt	Sydney	Singapore	Dubai	Narita
0	S1	2.000	10.972	14.628	14.712	13.256	12.547
1	S2	9.978	12.418	14.003	15.071	13.514	16.028
2	S3	7.085	10.245	15.024	14.609	12.694	12.637
3	S4	10.063	13.451	13.123	13.795	14.436	15.413
4	S5	12.786	11.404	11.995	9.536	10.346	7.955
5	S6	10.972	2.000	13.972	12.626	9.733	12.191
6	S7	12.566	12.183	11.429	10.044	11.483	2.000
7	S8	12.727	10.063	9.737	9.659	10.697	7.218
8	S9	10.781	3.900	14.040	12.735	9.757	12.168
9	S10	14.712	12.626	10.608	2.000	10.351	10.072
10	S11	13.256	9.733	13.437	10.351	2.000	11.508
11	S12	10.639	3.900	15.450	12.901	10.114	12.296
12	S13	14.628	13.972	2.000	10.608	13.437	11.429
13	S14	12.000	12.000	12.000	12.000	12.000	12.000

	SaoPaulo	Amsterdam	Belfast	Montreal	Seoul	Shanghai	Santiago
0	9.978	10.781	10.360	7.218	12.727	13.137	10.063
1	2.000	12.407	12.269	9.820	15.972	16.051	8.276
2	9.863	10.208	9.781	2.000	12.727	13.119	10.198
3	8.276	13.423	13.079	10.198	15.963	16.160	2.000
4	15.693	11.425	9.834	12.720	7.012	2.000	16.236
5	12.418	3.900	7.139	10.351	10.063	11.932	13.451
6	16.040	12.160	12.241	12.682	6.916	7.676	15.542
7	15.972	10.068	11.920	12.761	2.000	6.957	15.963
8	12.407	2.000	3.900	10.147	10.068	11.960	13.423
9	15.071	12.735	13.032	14.602	9.659	9.078	13.795
10	13.514	9.757	10.387	12.786	10.697	10.670	14.436
11	12.261	3.900	2.000	9.987	11.948	12.117	13.270
12	14.003	14.040	15.508	13.764	9.737	11.433	13.123
13	12.000	12.000	12.000	12.000	12.000	12.000	12.000

	Beijing	HongKong	Dallas	Linwood	Guangzhou	FortLauderdale
0	12.786	13.650	2.000	10.382	13.622	2.000
1	15.693	15.859	9.863	12.353	15.708	10.772
2	12.716	13.601	8.061	9.815	13.426	7.984
3	16.236	16.099	11.442	13.302	16.143	10.829
4	2.000	7.576	13.076	11.512	2.000	13.622
5	11.404	11.940	9.887	7.085	12.035	11.379
6	7.919	8.490	12.686	12.156	8.485	13.398
7	7.012	7.919	12.952	10.119	7.901	13.444
8	11.425	12.156	11.466	3.900	11.920	11.212

```
9     9.536     8.276  14.939   12.967      8.310          15.441
10   10.346    10.208  13.816   10.337     10.346          13.664
11    9.820    12.322  11.329    2.000     12.261          11.041
12   11.995    11.190  14.194   15.441     11.258          14.696
13   12.000    12.000  12.000   12.000     12.000          12.000
```

[ ]:
```
city_to_depot = {
    'Chicago': 'D1',
    'Frankfurt': 'D2',
    'Sydney': 'D3',
    'Singapore': 'D4',
    'Dubai': 'D5',
    'Narita': 'D6',
    'SaoPaulo': 'D7',
    'Amsterdam': 'D8',
    'Belfast': 'D9',
    'Montreal': 'D10',
    'Seoul': 'D11',
    'Shanghai': 'D12',
    'Santiago': 'D13',
    'Beijing': 'D14',
    'HongKong': 'D15',
    'Dallas': 'D16',
    'Linwood': 'D17',
    'Guangzhou': 'D18',
    'FortLauderdale': 'D19'
}

#replaceing city names with depot IDs
Trans_cost_IN = Trans_cost_IN.rename(columns=city_to_depot)
Trans_cost_IN
```

[ ]:
```
   Unnamed: 0      D1      D2      D3      D4      D5      D6      D7      D8  \
0          S1   2.000  10.972  14.628  14.712  13.256  12.547   9.978  10.781
1          S2   9.978  12.418  14.003  15.071  13.514  16.028   2.000  12.407
2          S3   7.085  10.245  15.024  14.609  12.694  12.637   9.863  10.208
3          S4  10.063  13.451  13.123  13.795  14.436  15.413   8.276  13.423
4          S5  12.786  11.404  11.995   9.536  10.346   7.955  15.693  11.425
5          S6  10.972   2.000  13.972  12.626   9.733  12.191  12.418   3.900
6          S7  12.566  12.183  11.429  10.044  11.483   2.000  16.040  12.160
7          S8  12.727  10.063   9.737   9.659  10.697   7.218  15.972  10.068
8          S9  10.781   3.900  14.040  12.735   9.757  12.168  12.407   2.000
9         S10  14.712  12.626  10.608   2.000  10.351  10.072  15.071  12.735
10        S11  13.256   9.733  13.437  10.351   2.000  11.508  13.514   9.757
11        S12  10.639   3.900  15.450  12.901  10.114  12.296  12.261   3.900
12        S13  14.628  13.972   2.000  10.608  13.437  11.429  14.003  14.040
13        S14  12.000  12.000  12.000  12.000  12.000  12.000  12.000  12.000
```

	D9	D10	D11	D12	D13	D14	D15	D16	D17	\
0	10.360	7.218	12.727	13.137	10.063	12.786	13.650	2.000	10.382	
1	12.269	9.820	15.972	16.051	8.276	15.693	15.859	9.863	12.353	
2	9.781	2.000	12.727	13.119	10.198	12.716	13.601	8.061	9.815	
3	13.079	10.198	15.963	16.160	2.000	16.236	16.099	11.442	13.302	
4	9.834	12.720	7.012	2.000	16.236	2.000	7.576	13.076	11.512	
5	7.139	10.351	10.063	11.932	13.451	11.404	11.940	9.887	7.085	
6	12.241	12.682	6.916	7.676	15.542	7.919	8.490	12.686	12.156	
7	11.920	12.761	2.000	6.957	15.963	7.012	7.919	12.952	10.119	
8	3.900	10.147	10.068	11.960	13.423	11.425	12.156	11.466	3.900	
9	13.032	14.602	9.659	9.078	13.795	9.536	8.276	14.939	12.967	
10	10.387	12.786	10.697	10.670	14.436	10.346	10.208	13.816	10.337	
11	2.000	9.987	11.948	12.117	13.270	9.820	12.322	11.329	2.000	
12	15.508	13.764	9.737	11.433	13.123	11.995	11.190	14.194	15.441	
13	12.000	12.000	12.000	12.000	12.000	12.000	12.000	12.000	12.000	

	D18	D19
0	13.622	2.000
1	15.708	10.772
2	13.426	7.984
3	16.143	10.829
4	2.000	13.622
5	12.035	11.379
6	8.485	13.398
7	7.901	13.444
8	11.920	11.212
9	8.310	15.441
10	10.346	13.664
11	12.261	11.041
12	11.258	14.696
13	12.000	12.000

[ ]:
```
weights = pd.read_csv(r"C:\Users\opett\OneDrive - Wilfrid Laurier␣
↪University\Decision Making with Analytics files\Group Assignment\Weights.
↪csv", header = None)
weights
```

[ ]:

	0	1
0	P1	1.454058
1	P2	1.559032
2	P3	6.043940
3	P4	2.390903
4	P5	4.831571
..	…	…
122	P140	1.360770
123	P141	1.360770

```
124  P142  1.360770
125  P143  2.162200
126  P144  1.423280

[127 rows x 2 columns]
```

Structering each dataframe to dict for easier workability and indexing

[ ]:
```
demand_dict = {}
for index, row in demand.iterrows():
    part_class = row['Unnamed: 0']
    for country_code in demand.columns[1:]:
        if part_class not in demand_dict:
            demand_dict[part_class] = {}
        demand_dict[part_class][country_code] = row[country_code]
```

[ ]:
```
fixed_costs_dict = fixed_costs.set_index(0)[1].to_dict()
```

[ ]:
```
Trans_costs.set_index('Unnamed: 0', inplace=True)

trans_costs_dict = {depot: Trans_costs[depot].to_dict() for depot in␣
 ↪Trans_costs.columns}
```

[ ]:
```
Trans_cost_IN.set_index('Unnamed: 0', inplace=True)

trans_cost_in_dict = {supplier: Trans_cost_IN.loc[supplier].to_dict() for␣
 ↪supplier in Trans_cost_IN.index}
```

[ ]:
```
weights_dict = pd.Series(weights[1].values, index=weights[0]).to_dict()
```

[ ]:
```
m = Model("Group Assignment")
```

```
Set parameter Username
Academic license - for non-commercial use only - expires 2025-02-06
Academic license - for non-commercial use only - expires 2025-02-06
```

Setting up Decsion Variables

[ ]:
```
depots = fixed_costs_dict.keys()
parts = weights_dict.keys()
suppliers = trans_cost_in_dict.keys()
country_codes = list(next(iter(trans_costs_dict.values())).keys())
```

[ ]:
```
#making depot vars (will be binary to reflect open/close decision)
D = m.addVars(depots, vtype=GRB.BINARY, name="D")
```

[ ]:
```
#units of part p from supplier s to depot d
Xpsd = m.addVars(parts, suppliers, depots, vtype=GRB.CONTINUOUS, name="Xpsd")
```

[ ]:
```
#units of part p from depot d to country code cc
Xpdcc = m.addVars(parts, depots, country_codes, vtype=GRB.CONTINUOUS,␣
↪name="Xpdcc")
```

[ ]:
```
#non neg constaraints
m.addConstrs((Xpsd[p, s, d] >= 0 for p in parts for s in suppliers for d in␣
↪depots), "NonNeg_Xpsd")
m.addConstrs((Xpdcc[p, d, cc] >= 0 for p in parts for d in depots for cc in␣
↪country_codes), "NonNeg_Xpdcc")
m.update()
```

[ ]:
```
#supplier capacity for S2 through S14 (not incldуing s1 as it does not have␣
↪same cap, it is unlimited)
m.addConstrs((
    quicksum(weights_dict[p] * Xpsd[p, s, d] for p in parts for d in depots) <=␣
↪200000
    for s in suppliers if s != 'S1'), "SupplierCapacity")
m.update()
```

[ ]:
```
#depot capacities, 300000 for Chicago, 150000 for all else
m.addConstrs((
    quicksum(Xpsd[p, s, d] for p in parts for s in suppliers) <= D[d] * (300000␣
↪if d == 'D1' else 150000)
    for d in depots), "DepotCapacity")
m.update()
```

[ ]:
```
#equality constraints, amount of parts into depot must be equal to amount of␣
↪parts out
m.addConstrs((
    quicksum(Xpsd[p, s, d] for s in suppliers) == quicksum(Xpdcc[p, d, cc] for␣
↪cc in country_codes)
    for p in parts for d in depots), "TransshipmentEquality")
m.update()
```

[ ]:
```
#demand satisfaction constraints
m.addConstrs((
    quicksum(Xpdcc[p, d, cc] for d in depots) == demand_dict[p][cc]
    for p in parts for cc in country_codes if cc in demand_dict[p]),␣
↪"DemandSatisfaction")
m.update()
```

[ ]:
```
#service level constraints
m.addConstrs((
    Xpdcc[p, d, cc] * weights_dict[p] * trans_costs_dict[d][cc] <= Xpdcc[p, d,␣
↪cc] * weights_dict[p] * 5
    for p in parts for d in depots for cc in country_codes), "ServiceLevel")
m.update()
```

[ ]:
```
#setting up objective function
objective_func = quicksum(fixed_costs_dict[d] * D[d] for d in depots) + \
            quicksum(weights_dict[p] * trans_cost_in_dict[s][d] * Xpsd[p, s, d]␣
 ↪for p in parts for s in suppliers for d in depots) + \
            quicksum(weights_dict[p] * trans_costs_dict[d][cc] * Xpdcc[p, d,␣
 ↪cc] for p in parts for d in depots for cc in country_codes)

m.setObjective(objective_func, GRB.MINIMIZE)
```

[ ]:
```
m.optimize()
```

```
Gurobi Optimizer version 11.0.0 build v11.0.0rc2 (win64 - Windows 11+.0
(22631.2))

CPU model: 11th Gen Intel(R) Core(TM) i5-1155G7 @ 2.50GHz, instruction set
[SSE2|AVX|AVX2|AVX512]
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads


CPU model: 11th Gen Intel(R) Core(TM) i5-1155G7 @ 2.50GHz, instruction set
[SSE2|AVX|AVX2|AVX512]
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 625634 rows, 320948 columns and 1281322 nonzeros
Model fingerprint: 0xa3a49656
Variable types: 320929 continuous, 19 integer (19 binary)
Coefficient statistics:
  Matrix range     [1e-03, 3e+05]
  Objective range  [1e+00, 4e+05]
  Bounds range     [1e+00, 1e+00]
  RHS range        [4e-03, 2e+05]
Presolve removed 620245 rows and 260136 columns
Presolve time: 0.23s
Presolved: 5389 rows, 60812 columns, 152974 nonzeros
Variable types: 60793 continuous, 19 integer (19 binary)

Root relaxation: objective 3.579949e+06, 4559 iterations, 0.08 seconds (0.04
work units)

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

     0     0 3579949.18    0   17          - 3579949.18      -     -    1s
H    0     0                    7680785.0730 3579949.18  53.4%     -    1s
H    0     0                    7436606.3456 3579949.18  51.9%     -    1s
H    0     0                    4741015.1955 3579949.18  24.5%     -    1s
     0     0 4147333.35    0   18 4741015.20 4147333.35  12.5%     -    3s
H    0     0                    4618757.4275 4158479.82  10.0%     -    5s
```

```
     0     0 4251774.69    0   18 4618757.43 4251774.69  7.95%     -    5s
     0     0 4334620.91    0   18 4618757.43 4334620.91  6.15%     -    6s
     0     0 4345787.85    0   18 4618757.43 4345787.85  5.91%     -    7s
H    0     0                     4367933.5784 4358345.08  0.22%     -    9s
     0     0 4367859.11    0   15 4367933.58 4367859.11  0.00%     -    9s

Cutting planes:
  Cover: 2
  Implied bound: 2958
  MIR: 5
  Flow cover: 281
  Flow path: 333
  Network: 34
  Relax-and-lift: 14

Explored 1 nodes (11233 simplex iterations) in 9.17 seconds (5.64 work units)
Thread count was 8 (of 8 available processors)

Solution count 6: 4.36793e+06 4.36793e+06 4.61876e+06 … 7.68079e+06

Optimal solution found (tolerance 1.00e-04)
Best objective 4.367932967102e+06, best bound 4.367859107168e+06, gap 0.0017%
```

[ ]:
```
print(f"Number of Variables in model 'm': {m.numVars}")
print(f"Number of Constraints in model 'm': {m.numConstrs}")
```

```
Number of Variables in model 'm': 320948
Number of Constraints in model 'm': 625634
```

[ ]:
```
opened_depots = [d for d in depots if D[d].X == 1]
print("Depots to be opened:", opened_depots)
```

```
Depots to be opened: ['D1', 'D2', 'D4', 'D13']
```

[ ]:
```
print(f"Total Cost: {m.ObjVal}")
```

```
Total Cost: 4367932.967101574
```

[ ]:
```
total_depot_costs = 0
for d in depots:
        depot_cost = fixed_costs_dict[d] * D[d].X
        if D[d].X == 1:
            print(f"Cost for opening Depot {d}: {depot_cost}")
        total_depot_costs += depot_cost

    # Print the sum of the depot costs
print(f"Sum of Depot Costs: {total_depot_costs}")
```

```
Cost for opening Depot D1: 431365.0
Cost for opening Depot D2: 387232.0
```

```
Cost for opening Depot D4: 172474.0
Cost for opening Depot D13: 180916.0
Sum of Depot Costs: 1171987.5389950348
```

[ ]:
```
total_cost_from_suppliers = sum(Xpsd[p, s, d].X * weights_dict[p] *␣
 ↪trans_cost_in_dict[s][d]
                                    for p in parts for s in suppliers for d in␣
 ↪depots)
print(f"Total Cost of Shipments from Suppliers to Depots:␣
 ↪{total_cost_from_suppliers}")

total_cost_to_customers = sum(Xpdcc[p, d, cc].X * weights_dict[p] *␣
 ↪trans_costs_dict[d][cc]
                                  for p in parts for d in depots for cc in␣
 ↪country_codes)
print(f"Total Cost of Shipments from Depots to Customers:␣
 ↪{total_cost_to_customers}")

print(f'Total Variable operation costs:
 ↪',(total_cost_from_suppliers+total_cost_to_customers))
```

```
Total Cost of Shipments from Suppliers to Depots: 1636867.945763741
Total Cost of Shipments from Depots to Customers: 1559077.4823427973
Total Variable operation costs: 3195945.428106538
```

[ ]:
```
total_parts_to_depot = {d: 0 for d in depots}

for p in parts:
    for s in suppliers:
        for d in depots:
            if D[d].X ==1:
                quantity_shipped = Xpsd[p, s, d].X
                total_parts_to_depot[d] += quantity_shipped

for d in D:
    if D[d].X == 1:
        print(f"Total amount of parts shipped to Depot {d}:␣
 ↪{total_parts_to_depot[d]}")
```

```
Total amount of parts shipped to Depot D1: 267892.6752642444
Total amount of parts shipped to Depot D2: 140443.7043219401
Total amount of parts shipped to Depot D4: 63227.85062441813
Total amount of parts shipped to Depot D13: 4207.660404054502
```

Part 3:

[ ]:
```
total_parts_per_depot = {d: 0 for d in depots}
total_kg_per_depot = {d: 0 for d in depots}
```

```
for p in parts:
    for s in suppliers:
        for d in depots:
            if D[d].X == 1:
                quantity_shipped = Xpsd[p, s, d].X
                if quantity_shipped > 0:
                    total_parts_per_depot[d] += quantity_shipped
                    total_kg_per_depot[d] += quantity_shipped * weights_dict[p]


for p in parts:
    for d in depots:
        if D[d].X == 1:
            for cc in country_codes:
                quantity_shipped = Xpdcc[p, d, cc].X
                if quantity_shipped > 0:
                    total_kg_per_depot[d] += quantity_shipped * weights_dict[p]


for d in D:
    if D[d].X == 1:
        print(f"Depot {d}: Total Kilograms = {total_kg_per_depot[d]}")
```

```
Depot D1: Total Kilograms = 902241.7544737543
Depot D2: Total Kilograms = 456931.1627915894
Depot D4: Total Kilograms = 209232.65205949018
Depot D13: Total Kilograms = 14373.939851971934

Depot D2: Total Kilograms = 456931.1627915894
Depot D4: Total Kilograms = 209232.65205949018
Depot D13: Total Kilograms = 14373.939851971934
```

[ ]:
```
depot_capacities = {d: 150000 for d in depots}
depot_capacities['D1'] = 300000

for d in depots:
    if D[d].X == 1:
        total_parts = total_parts_per_depot[d]
        capacity = depot_capacities[d]
        utilization_percentage = (total_parts / capacity) * 100
        print(f"Depot {d}: Total Parts = {total_parts}, Capacity = {capacity},␣
 ↪Utilization = {utilization_percentage:.2f}%")
```

```
Depot D1: Total Parts = 267892.6752642444, Capacity = 300000, Utilization =
89.30%
Depot D2: Total Parts = 140443.7043219401, Capacity = 150000, Utilization =
93.63%
```

```
Depot D4: Total Parts = 63227.85062441813, Capacity = 150000, Utilization =
42.15%
Depot D13: Total Parts = 4207.660404054502, Capacity = 150000, Utilization =
2.81%
```

[ ]:
```python
destinations_per_depot = {d: set() for d in depots}

for p in parts:
    for d in depots:
        for cc in country_codes:
            if Xpdcc[p, d, cc].X > 0:
                destinations_per_depot[d].add(cc)


for d in depots:
    if D[d].X == 1:
        destinations = ", ".join(sorted(destinations_per_depot[d]))
        print(f"Depot {d} ships to the following countries: {destinations}")
```

```
Depot D1 ships to the following countries: AG, AN, AW, BB, BF, BG, BS, CA, CO,
CR, CV, EE, ES, FI, GR, GT, IE, IS, JM, KY, LC, LV, LY, MA, MK, MQ, MT, MX, NO,
PA, PR, PT, RO, RU, SE, SN, TT, UA, US, VE, VG
Depot D2 ships to the following countries: AO, AT, BE, BW, CG, CH, CZ, DE, DK,
DZ, FR, GA, GB, HR, HU, IL, IT, JO, KE, LB, LU, NL, PL, SI, SK, TD, TN, UG
Depot D4 ships to the following countries: AE, AF, AU, BD, BH, CN, CY, EG, ET,
GE, ID, IN, IQ, JP, KR, KW, KZ, MO, MU, MV, MY, MZ, NZ, OM, PG, PH, PK, QA, RW,
SA, SG, SZ, TH, TM, TR, TW, TZ, VN, YE, ZA, ZM
Depot D13 ships to the following countries: AR, BR, CK, CL, EC, GU, NG, UY
```

[ ]:
```python
#doing compliance check on all depots for flight time constraints (must be␣
↪under $5/kg)
for d in depots:
    for cc in country_codes:
        if any(Xpdcc[p, d, cc].X > 0 for p in parts):
            cost_per_kg = trans_costs_dict[d][cc]
            if cost_per_kg > 5:
                print(f"Non-compliance found: Shipping from Depot {d} to␣
↪Country {cc} exceeds $5/kg at ${cost_per_kg}/kg")
            else:
                print(f"Compliant: Shipping from Depot {d} to Country {cc} at␣
↪${cost_per_kg}/kg")
```

```
Compliant: Shipping from Depot D1 to Country AG at $4.03/kg
Compliant: Shipping from Depot D1 to Country AN at $4.056/kg
Compliant: Shipping from Depot D1 to Country AW at $4.028/kg
Compliant: Shipping from Depot D1 to Country BB at $4.166/kg
Compliant: Shipping from Depot D1 to Country BF at $4.549/kg
Compliant: Shipping from Depot D1 to Country BG at $4.354/kg
```

```
Compliant: Shipping from Depot D1 to Country BS at $3.534/kg
Compliant: Shipping from Depot D1 to Country CA at $3.162/kg
Compliant: Shipping from Depot D1 to Country CO at $4.12/kg
Compliant: Shipping from Depot D1 to Country CR at $3.899/kg
Compliant: Shipping from Depot D1 to Country CV at $4.772/kg
Compliant: Shipping from Depot D1 to Country EE at $4.946/kg
Compliant: Shipping from Depot D1 to Country ES at $4.76/kg
Compliant: Shipping from Depot D1 to Country FI at $4.935/kg
Compliant: Shipping from Depot D1 to Country GR at $4.541/kg
Compliant: Shipping from Depot D1 to Country GT at $3.836/kg
Compliant: Shipping from Depot D1 to Country IE at $4.631/kg
Compliant: Shipping from Depot D1 to Country IS at $4.332/kg
Compliant: Shipping from Depot D1 to Country JM at $3.776/kg
Compliant: Shipping from Depot D1 to Country KY at $3.69/kg
Compliant: Shipping from Depot D1 to Country LC at $4.127/kg
Compliant: Shipping from Depot D1 to Country LV at $4.982/kg
Compliant: Shipping from Depot D1 to Country LY at $4.451/kg
Compliant: Shipping from Depot D1 to Country MA at $4.875/kg
Compliant: Shipping from Depot D1 to Country MK at $4.423/kg
Compliant: Shipping from Depot D1 to Country MQ at $4.111/kg
Compliant: Shipping from Depot D1 to Country MT at $4.425/kg
Compliant: Shipping from Depot D1 to Country MX at $3.634/kg
Compliant: Shipping from Depot D1 to Country NO at $4.783/kg
Compliant: Shipping from Depot D1 to Country PA at $4.046/kg
Compliant: Shipping from Depot D1 to Country PR at $3.916/kg
Compliant: Shipping from Depot D1 to Country PT at $4.764/kg
Compliant: Shipping from Depot D1 to Country RO at $4.358/kg
Compliant: Shipping from Depot D1 to Country RU at $4.343/kg
Compliant: Shipping from Depot D1 to Country SE at $4.875/kg
Compliant: Shipping from Depot D1 to Country SN at $4.975/kg
Compliant: Shipping from Depot D1 to Country TT at $4.206/kg
Compliant: Shipping from Depot D1 to Country UA at $4.378/kg
Compliant: Shipping from Depot D1 to Country US at $0.872/kg
Compliant: Shipping from Depot D1 to Country VE at $4.12/kg
Compliant: Shipping from Depot D1 to Country VG at $3.945/kg
Compliant: Shipping from Depot D2 to Country AO at $4.799/kg
Compliant: Shipping from Depot D2 to Country AT at $1.7/kg
Compliant: Shipping from Depot D2 to Country BE at $0.872/kg
Compliant: Shipping from Depot D2 to Country BW at $4.474/kg
Compliant: Shipping from Depot D2 to Country CG at $4.678/kg
Compliant: Shipping from Depot D2 to Country CH at $1.7/kg
Compliant: Shipping from Depot D2 to Country CZ at $1.7/kg
Compliant: Shipping from Depot D2 to Country DE at $0.872/kg
Compliant: Shipping from Depot D2 to Country DK at $1.7/kg
Compliant: Shipping from Depot D2 to Country DZ at $3.336/kg
Compliant: Shipping from Depot D2 to Country FR at $1.7/kg
Compliant: Shipping from Depot D2 to Country GA at $4.537/kg
Compliant: Shipping from Depot D2 to Country GB at $1.7/kg
```

```
Compliant: Shipping from Depot D2 to Country HR at $1.7/kg
Compliant: Shipping from Depot D2 to Country HU at $3.077/kg
Compliant: Shipping from Depot D2 to Country IL at $3.811/kg
Compliant: Shipping from Depot D2 to Country IT at $3.132/kg
Compliant: Shipping from Depot D2 to Country JO at $3.828/kg
Compliant: Shipping from Depot D2 to Country KE at $4.736/kg
Compliant: Shipping from Depot D2 to Country LB at $3.761/kg
Compliant: Shipping from Depot D2 to Country LU at $1.7/kg
Compliant: Shipping from Depot D2 to Country NL at $1.7/kg
Compliant: Shipping from Depot D2 to Country PL at $3.105/kg
Compliant: Shipping from Depot D2 to Country SI at $1.7/kg
Compliant: Shipping from Depot D2 to Country SK at $1.7/kg
Compliant: Shipping from Depot D2 to Country TD at $4.195/kg
Compliant: Shipping from Depot D2 to Country TN at $3.325/kg
Compliant: Shipping from Depot D2 to Country UG at $4.572/kg
Compliant: Shipping from Depot D4 to Country AE at $4.619/kg
Compliant: Shipping from Depot D4 to Country AF at $4.343/kg
Compliant: Shipping from Depot D4 to Country AU at $4.734/kg
Compliant: Shipping from Depot D4 to Country BD at $3.789/kg
Compliant: Shipping from Depot D4 to Country BH at $4.74/kg
Compliant: Shipping from Depot D4 to Country CN at $4.256/kg
Compliant: Shipping from Depot D4 to Country CY at $4.386/kg
Compliant: Shipping from Depot D4 to Country EG at $4.414/kg
Compliant: Shipping from Depot D4 to Country ET at $4.965/kg
Compliant: Shipping from Depot D4 to Country GE at $4.999/kg
Compliant: Shipping from Depot D4 to Country ID at $3.105/kg
Compliant: Shipping from Depot D4 to Country IN at $4.164/kg
Compliant: Shipping from Depot D4 to Country IQ at $4.935/kg
Compliant: Shipping from Depot D4 to Country JP at $4.482/kg
Compliant: Shipping from Depot D4 to Country KR at $4.311/kg
Compliant: Shipping from Depot D4 to Country KW at $4.817/kg
Compliant: Shipping from Depot D4 to Country KZ at $4.738/kg
Compliant: Shipping from Depot D4 to Country MO at $3.683/kg
Compliant: Shipping from Depot D4 to Country MU at $4.551/kg
Compliant: Shipping from Depot D4 to Country MV at $3.94/kg
Compliant: Shipping from Depot D4 to Country MY at $0.872/kg
Compliant: Shipping from Depot D4 to Country MZ at $4.399/kg
Compliant: Shipping from Depot D4 to Country NZ at $4.484/kg
Compliant: Shipping from Depot D4 to Country OM at $4.52/kg
Compliant: Shipping from Depot D4 to Country PG at $4.382/kg
Compliant: Shipping from Depot D4 to Country PH at $3.631/kg
Compliant: Shipping from Depot D4 to Country PK at $4.348/kg
Compliant: Shipping from Depot D4 to Country QA at $4.718/kg
Compliant: Shipping from Depot D4 to Country RW at $4.399/kg
Compliant: Shipping from Depot D4 to Country SA at $4.821/kg
Compliant: Shipping from Depot D4 to Country SG at $0.872/kg
Compliant: Shipping from Depot D4 to Country SZ at $4.354/kg
Compliant: Shipping from Depot D4 to Country TH at $3.305/kg
```

```
Compliant: Shipping from Depot D4 to Country TM at $4.7/kg
Compliant: Shipping from Depot D4 to Country TR at $4.425/kg
Compliant: Shipping from Depot D4 to Country TW at $3.899/kg
Compliant: Shipping from Depot D4 to Country TZ at $4.958/kg
Compliant: Shipping from Depot D4 to Country VN at $3.571/kg
Compliant: Shipping from Depot D4 to Country YE at $4.758/kg
Compliant: Shipping from Depot D4 to Country ZA at $4.514/kg
Compliant: Shipping from Depot D4 to Country ZM at $4.478/kg
Compliant: Shipping from Depot D13 to Country AR at $3.074/kg
Compliant: Shipping from Depot D13 to Country BR at $3.693/kg
Compliant: Shipping from Depot D13 to Country CK at $4.516/kg
Compliant: Shipping from Depot D13 to Country CL at $0.872/kg
Compliant: Shipping from Depot D13 to Country EC at $4.058/kg
Compliant: Shipping from Depot D13 to Country GU at $4.404/kg
Compliant: Shipping from Depot D13 to Country NG at $4.404/kg
Compliant: Shipping from Depot D13 to Country UY at $3.273/kg
Compliant: Shipping from Depot D15 to Country MH at $4.744/kg
```

As it stands, only depots 4 and 13 could handle a 20% demand increase, to make sure we are providing a flexible and robost recomendation, lets run a model with demand increased by 20%

[ ]:
```
p3 = Model("Increased Demand Model")

#making depot vars (will be binary to reflect open/close decision)
D_p3 = p3.addVars(depots, vtype=GRB.BINARY, name="D_p3")
#units of part p from supplier s to depot d
Xpsd_p3 = p3.addVars(parts, suppliers, depots, vtype=GRB.CONTINUOUS,␣
 ↪name="Xpsd_p3")
#units of part p from depot d to country code cc
Xpdcc_p3 = p3.addVars(parts, depots, country_codes, vtype=GRB.CONTINUOUS,␣
 ↪name="Xpdcc_p3")
#non neg constarints
p3.addConstrs((Xpsd_p3[p, s, d] >= 0 for p in parts for s in suppliers for d in␣
 ↪depots), "NonNeg_Xpsd_p3")
p3.addConstrs((Xpdcc_p3[p, d, cc] >= 0 for p in parts for d in depots for cc in␣
 ↪country_codes), "NonNeg_Xpdcc_p3")
#supplier capacity for S2 through S14 (not incldуing s1 as it does not have␣
 ↪same cap, it is unlimited)
p3.addConstrs((
    quicksum(weights_dict[p] * Xpsd_p3[p, s, d] for p in parts for d in depots)␣
 ↪<= 200000
    for s in suppliers if s != 'S1'), "SupplierCapacity")
#depot capacities, 300000 for Chicago, 150000 for all else
p3.addConstrs((
    quicksum(Xpsd_p3[p, s, d] for p in parts for s in suppliers) <= D_p3[d] *␣
 ↪(300000 if d == 'D1' else 150000)
    for d in depots), "DepotCapacity")
```

```
#equality constraints, amount of parts into depot must be equal to amount of␣
 ↪parts out
p3.addConstrs((
    quicksum(Xpsd_p3[p, s, d] for s in suppliers) == quicksum(Xpdcc_p3[p, d,␣
 ↪cc] for cc in country_codes)
    for p in parts for d in depots), "TransshipmentEquality")
#demand elevated by 20%
p3.addConstrs((
    quicksum(Xpdcc_p3[p, d, cc] for d in depots) == 1.2*demand_dict[p][cc]
    for p in parts for cc in country_codes if cc in demand_dict[p]),␣
 ↪"DemandSatisfaction")
#service level constraints
p3.addConstrs((
    Xpdcc_p3[p, d, cc] * weights_dict[p] * trans_costs_dict[d][cc] <=␣
 ↪Xpdcc_p3[p, d, cc] * weights_dict[p] * 5
    for p in parts for d in depots for cc in country_codes), "ServiceLevel")

p3.update()

#setting up objective function
objective_func_p3 = quicksum(fixed_costs_dict[d] * D_p3[d] for d in depots) + \
            quicksum(weights_dict[p] * trans_cost_in_dict[s][d] * Xpsd_p3[p, s,␣
 ↪d] for p in parts for s in suppliers for d in depots) + \
            quicksum(weights_dict[p] * trans_costs_dict[d][cc] * Xpdcc_p3[p, d,␣
 ↪cc] for p in parts for d in depots for cc in country_codes)

p3.setObjective(objective_func_p3, GRB.MINIMIZE)

p3.optimize()
```

```
Gurobi Optimizer version 11.0.0 build v11.0.0rc2 (win64 - Windows 11+.0
(22631.2))

CPU model: 11th Gen Intel(R) Core(TM) i5-1155G7 @ 2.50GHz, instruction set
[SSE2|AVX|AVX2|AVX512]
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 625634 rows, 320948 columns and 1281322 nonzeros

CPU model: 11th Gen Intel(R) Core(TM) i5-1155G7 @ 2.50GHz, instruction set
[SSE2|AVX|AVX2|AVX512]
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 625634 rows, 320948 columns and 1281322 nonzeros
Model fingerprint: 0x2a77c579
Variable types: 320929 continuous, 19 integer (19 binary)
Coefficient statistics:
```

```
  Matrix range     [1e-03, 3e+05]
  Objective range  [1e+00, 4e+05]
  Bounds range     [1e+00, 1e+00]
  RHS range        [4e-03, 2e+05]
Presolve removed 620245 rows and 260136 columns
Presolve time: 0.29s
Presolved: 5389 rows, 60812 columns, 152974 nonzeros
Variable types: 60793 continuous, 19 integer (19 binary)

Root relaxation: objective 4.272329e+06, 4685 iterations, 0.07 seconds (0.04
work units)

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

     0     0 4272329.07    0   17          - 4272329.07      -     -    1s
H    0     0                     8179762.0876 4272329.07  47.8%     -    1s
H    0     0                     7936254.2147 4272329.07  46.2%     -    1s
H    0     0                     5400103.1620 4272329.07  20.9%     -    1s
     0     0 4855699.49    0   18 5400103.16 4855699.49  10.1%     -    2s
H    0     0                     5386334.1949 4855699.49  9.85%     -    5s
     0     0 5004115.23    0   17 5386334.19 5004115.23  7.10%     -    5s
     0     0 5057312.37    0   18 5386334.19 5057312.37  6.11%     -    6s
     0     0 5069952.99    0   18 5386334.19 5069952.99  5.87%     -    7s
     0     0 5073813.23    0   18 5386334.19 5073813.23  5.80%     -    7s
H    0     0                     5256708.5070 5073813.23  3.48%     -    7s
H    0     0                     5093056.5853 5080576.12  0.25%     -    9s
     0     0 5093056.59    0   17 5093056.59 5093056.59  0.00%     -    9s

Cutting planes:
  Cover: 2
  Implied bound: 3111
  MIR: 6
  Flow cover: 246
  Flow path: 408
  Network: 42
  Relax-and-lift: 17

Explored 1 nodes (11018 simplex iterations) in 10.09 seconds (6.00 work units)
Thread count was 8 (of 8 available processors)

Solution count 7: 5.09306e+06 5.09306e+06 5.25671e+06 … 8.17976e+06
No other solutions better than 5.09306e+06

Optimal solution found (tolerance 1.00e-04)
Best objective 5.093056081775e+06, best bound 5.093056081775e+06, gap 0.0000%
```

[ ]:
```
opened_depots_p3 = [d for d in depots if D_p3[d].X == 1]
print("Depots to be opened in 20% demand increase model:", opened_depots_p3)
```

```
Depots to be opened in 20% demand increase model: ['D1', 'D2', 'D4', 'D13',
'D17']
 ['D1', 'D2', 'D4', 'D13', 'D17']
```

[ ]:
```
print(f"Total Cost of 20% increase demand model: {p3.ObjVal}")
```

```
Total Cost of 20% increase demand model: 5093056.08177539
```

[ ]:
```
total_depot_costs_p3 = 0
for d in depots:
        depot_cost_p3 = fixed_costs_dict[d] * D_p3[d].X
        if D_p3[d].X == 1:
            print(f"Cost for opening Depot {d}: {depot_cost_p3}")
        total_depot_costs_p3 += depot_cost_p3
print(f"Sum of Depot Costs for 20% demand increase: {total_depot_costs_p3}")
```

```
Cost for opening Depot D1: 431365.0
Cost for opening Depot D2: 387232.0
Cost for opening Depot D4: 172474.0
Cost for opening Depot D13: 180916.0
Cost for opening Depot D17: 318570.0
Sum of Depot Costs for 20% demand increase: 1490558.3006850018
```

[ ]:
```
total_parts_per_depot_p3 = {d: 0 for d in depots}
total_kg_per_depot_p3 = {d: 0 for d in depots}


for p in parts:
    for s in suppliers:
        for d in depots:
            if D_p3[d].X == 1:
                quantity_shipped_p3 = Xpsd_p3[p, s, d].X
                if quantity_shipped_p3 > 0:
                    total_parts_per_depot_p3[d] += quantity_shipped_p3
                    total_kg_per_depot_p3[d] += quantity_shipped_p3 *␣
 ↪weights_dict[p]


for p in parts:
    for d in depots:
        if D_p3[d].X == 1:
            for cc in country_codes:
                quantity_shipped_p3 = Xpdcc_p3[p, d, cc].X
                if quantity_shipped_p3 > 0:
                    total_kg_per_depot_p3[d] += quantity_shipped_p3 *␣
 ↪weights_dict[p]
```

```
for d in D_p3:
    if D_p3[d].X == 1:
        print(f"Depot {d}: Total Kilograms = {total_kg_per_depot_p3[d]}")
```

```
Depot D1: Total Kilograms = 1012544.491607464
Depot D2: Total Kilograms = 306476.8630629728
Depot D4: Total Kilograms = 228113.65863525332
Depot D13: Total Kilograms = 27460.76386605602
Depot D17: Total Kilograms = 324692.7357904784
```

[ ]:
```
for d in depots:
    if D_p3[d].X == 1:
        total_parts_p3 = total_parts_per_depot_p3[d]
        capacity = depot_capacities[d]
        utilization_percentage_p3 = (total_parts_p3 / capacity) * 100
        print(f"Depot {d}: Total Parts = {total_parts_p3}, Capacity =␣
 ↪{capacity}, Utilization = {utilization_percentage_p3:.2f}%")
```

```
Depot D1: Total Parts = 300000.00000000023, Capacity = 300000, Utilization =
100.00%
Depot D2: Total Parts = 90756.98715885452, Capacity = 150000, Utilization =
60.50%
Depot D4: Total Parts = 69394.05038995243, Capacity = 150000, Utilization =
46.26%
Depot D13: Total Parts = 8286.166055308302, Capacity = 150000, Utilization =
5.52%
Depot D17: Total Parts = 102488.78773379838, Capacity = 150000, Utilization =
68.33%
```

[ ]:
```
destinations_per_depot_p3 = {d: set() for d in depots}

for p in parts:
    for d in depots:
        for cc in country_codes:
            if Xpdcc_p3[p, d, cc].X > 0:
                destinations_per_depot_p3[d].add(cc)


for d in depots:
    if D_p3[d].X == 1:
        destinations_p3 = ", ".join(sorted(destinations_per_depot_p3[d]))
        print(f"Depot {d} ships to the following countries: {destinations_p3}")
```

```
Depot D1 ships to the following countries: AG, AN, AW, BB, BS, CA, CR, GT, JM,
KY, LC, MX, PA, PR, TT, US, VE, VG
Depot D2 ships to the following countries: AE, AN, AO, AT, BE, BF, BG, BH, BW,
CG, CH, CY, CZ, DE, DK, DZ, EE, EG, ES, ET, FI, FR, GA, GE, GR, HR, HU, IL, IQ,
```

```
IT, JO, KE, KW, KZ, LB, LU, LV, LY, MA, MK, MT, NL, OM, PL, QA, RO, RU, SA, SE,
SI, SK, SN, TD, TM, TN, TR, UA, UG, VE, YE
Depot D4 ships to the following countries: AF, AU, BD, CN, ID, IN, JP, KR, MO,
MU, MV, MY, MZ, NZ, PG, PH, PK, RW, SG, SZ, TH, TW, VN, ZA
Depot D13 ships to the following countries: AG, AR, BB, BR, CK, CL, CO, CR, EC,
GU, LC, MQ, NG, PA, TT, UY
Depot D17 ships to the following countries: CV, GB, IE, IS, NL, NO, PT, TZ, ZM
```

Part 4: A sensitivty analysis to the flight time constraint

[ ]:
```
#Initializing array that will store the results of each model
results = []
flight_time_rule = 5
#Adding results to array
results.append({
    "flight_time_rule": flight_time_rule,
    "opened_depots": len(opened_depots),
    "objective_value": m.ObjVal,
    "total_depot_costs": total_depot_costs
})
```

[ ]:
```
results
```

[ ]:
```
[{'flight_time_rule': 5,
  'opened_depots': 4,
  'objective_value': 4367932.967101574,
  'total_depot_costs': 1171987.5389950348}]
```

Making model to reflect 15 hour flight time rule ($6/kg)

[ ]:
```
#Re-runing model but with flight_time_rule=6:
flight_time_rule=6

m6 = Model("Group Assignment6")

#Setting up Decsion Variables
depots = fixed_costs_dict.keys()
parts = weights_dict.keys()
suppliers = trans_cost_in_dict.keys()
country_codes = list(next(iter(trans_costs_dict.values())).keys())

#making depot vars (will be binary to reflect open/close decision)
D = m6.addVars(depots, vtype=GRB.BINARY, name="D")

#units of part p from supplier s to depot d
Xpsd = m6.addVars(parts, suppliers, depots, vtype=GRB.CONTINUOUS, name="Xpsd")

#units of part p from depot d to country code cc
```

```
Xpdcc = m6.addVars(parts, depots, country_codes, vtype=GRB.CONTINUOUS,␣
 ↪name="Xpdcc")

#non neg constraints
m6.addConstrs((Xpsd[p, s, d] >= 0 for p in parts for s in suppliers for d in␣
 ↪depots), "NonNeg_Xpsd")
m6.addConstrs((Xpdcc[p, d, cc] >= 0 for p in parts for d in depots for cc in␣
 ↪country_codes), "NonNeg_Xpdcc")

m6.update()

#supplier capacity for S2 through S14 (not incldуing s1 as it does not have␣
 ↪same cap, it is unlimited)
m6.addConstrs((
    quicksum(weights_dict[p] * Xpsd[p, s, d] for p in parts for d in depots) <=␣
 ↪200000
    for s in suppliers if s != 'S1'), "SupplierCapacity")


m6.update()

#depot capacities, 300000 for Chicago, 150000 for all else
m6.addConstrs((
    quicksum(Xpsd[p, s, d] for p in parts for s in suppliers) <= D[d] * (300000␣
 ↪if d == 'D1' else 150000)
    for d in depots), "DepotCapacity")

m6.update()

#equality constraints, amount of parts into depot must be equal to amount of␣
 ↪parts out
m6.addConstrs((
    quicksum(Xpsd[p, s, d] for s in suppliers) == quicksum(Xpdcc[p, d, cc] for␣
 ↪cc in country_codes)
    for p in parts for d in depots), "TransshipmentEquality")

m6.update()

#demand satisfaction constraints

m6.addConstrs((
    quicksum(Xpdcc[p, d, cc] for d in depots) == demand_dict[p][cc]
    for p in parts for cc in country_codes if cc in demand_dict[p]),␣
 ↪"DemandSatisfaction")

m6.update()
```

```
#service level constraints

m6.addConstrs((
    Xpdcc[p, d, cc] * weights_dict[p] * trans_costs_dict[d][cc] <= Xpdcc[p, d,␣
 ↪cc] * weights_dict[p] * flight_time_rule
    for p in parts for d in depots for cc in country_codes), "ServiceLevel")

m6.update()

#setting up objective function
objective_func = quicksum(fixed_costs_dict[d] * D[d] for d in depots) + \
            quicksum(weights_dict[p] * trans_cost_in_dict[s][d] * Xpsd[p, s, d]␣
 ↪for p in parts for s in suppliers for d in depots) + \
            quicksum(weights_dict[p] * trans_costs_dict[d][cc] * Xpdcc[p, d,␣
 ↪cc] for p in parts for d in depots for cc in country_codes)

m6.setObjective(objective_func, GRB.MINIMIZE)

m6.optimize()

print()
opened_depots = [d for d in depots if D[d].X == 1]
print("Depots to be opened:", opened_depots)
print(f"Total Cost: {m6.ObjVal}")
print()
total_depot_costs = 0
for d in depots:
        depot_cost = fixed_costs_dict[d] * D[d].X
        if D[d].X == 1:
            print(f"Cost for opening Depot {d}: {depot_cost}")
        total_depot_costs += depot_cost

    # Print the sum of the depot costs
print(f"Sum of Depot Costs: {total_depot_costs}")
print()
total_parts_to_depot = {d: 0 for d in depots}

for p in parts:
    for s in suppliers:
        for d in depots:
            quantity_shipped = Xpsd[p, s, d].X
            total_parts_to_depot[d] += quantity_shipped

for d in D:
    if D[d].X == 1:
```

```
        print(f"Total amount of parts shipped to Depot {d} (all parts must also␣
 ↪leave the depot): {total_parts_to_depot[d]}")
print()
total_parts_per_depot = {d: 0 for d in depots}
total_kg_per_depot = {d: 0 for d in depots}


for p in parts:
    for s in suppliers:
        for d in depots:
            if D[d].X == 1:
                quantity_shipped = Xpsd[p, s, d].X
                if quantity_shipped > 0:
                    total_parts_per_depot[d] += quantity_shipped
                    total_kg_per_depot[d] += quantity_shipped * weights_dict[p]


for p in parts:
    for d in depots:
        if D[d].X == 1:
            for cc in country_codes:
                quantity_shipped = Xpdcc[p, d, cc].X
                if quantity_shipped > 0:
                    total_kg_per_depot[d] += quantity_shipped * weights_dict[p]


for d in D:
    if D[d].X == 1:
        print(f"Depot {d}: Total Kilograms = {total_kg_per_depot[d]}")
print()
depot_capacities = {d: 150000 for d in depots}
depot_capacities['D1'] = 300000

for d in depots:
    if D[d].X == 1:
        total_parts = total_parts_per_depot[d]
        capacity = depot_capacities[d]
        utilization_percentage = (total_parts / capacity) * 100
        print(f"Depot {d}: Total Parts = {total_parts}, Capacity = {capacity},␣
 ↪Utilization = {utilization_percentage:.2f}%")
print()
destinations_per_depot = {d: set() for d in depots}

for p in parts:
    for d in depots:
        for cc in country_codes:
            if Xpdcc[p, d, cc].X > 0:
```

```
                destinations_per_depot[d].add(cc)


for d in depots:
    if D[d].X == 1:
        destinations = ", ".join(sorted(destinations_per_depot[d]))
        print(f"Depot {d} ships to the following countries: {destinations}")
print()
for d in depots:
    for cc in country_codes:
            # Check if there's any shipping from depot d to country code cc
        if any(Xpdcc[p, d, cc].X > 0 for p in parts):
            cost_per_kg = trans_costs_dict[d][cc]
            if cost_per_kg > flight_time_rule:
                print(f"Non-compliance found: Shipping from Depot {d} to␣
 ↪Country {cc} exceeds ${flight_time_rule}/kg at ${cost_per_kg}/kg")
            else:
                print(f"Compliant: Shipping from Depot {d} to Country {cc} at␣
 ↪${cost_per_kg}/kg")
print()
for d in depots:
    print(f"Depot {d} open status: {D[d].X}")
print()
#adding results to 'results' array
results.append({
    "flight_time_rule": flight_time_rule,
    "opened_depots": len(opened_depots),
    "objective_value": m6.ObjVal,
    "total_depot_costs": total_depot_costs
})
```

```
Gurobi Optimizer version 11.0.0 build v11.0.0rc2 (win64 - Windows 11+.0
(22631.2))

CPU model: 11th Gen Intel(R) Core(TM) i5-1155G7 @ 2.50GHz, instruction set
[SSE2|AVX|AVX2|AVX512]
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 625634 rows, 320948 columns and 1281322 nonzeros

CPU model: 11th Gen Intel(R) Core(TM) i5-1155G7 @ 2.50GHz, instruction set
[SSE2|AVX|AVX2|AVX512]
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 625634 rows, 320948 columns and 1281322 nonzeros
Model fingerprint: 0x9befc075
Variable types: 320929 continuous, 19 integer (19 binary)
```

```
Coefficient statistics:
  Matrix range     [1e-03, 3e+05]
  Objective range  [1e+00, 4e+05]
  Bounds range     [1e+00, 1e+00]
  RHS range        [4e-03, 2e+05]
Presolve removed 620207 rows and 241725 columns
Presolve time: 0.68s
Presolved: 5427 rows, 79223 columns, 189796 nonzeros
Variable types: 79204 continuous, 19 integer (19 binary)
Deterministic concurrent LP optimizer: primal and dual simplex
Showing primal log only…

Concurrent spin time: 0.00s

Solved with primal simplex
Extra simplex iterations after uncrush: 1

Root relaxation: objective 3.439609e+06, 13181 iterations, 0.26 seconds (0.13
work units)

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

     0     0 3439609.06    0   17          - 3439609.06      -     -    1s
H    0     0                     7680785.0721 3439609.06  55.2%     -    1s
H    0     0                     7637283.5685 3439609.06  55.0%     -    2s
H    0     0                     7261062.2057 3439609.06  52.6%     -    2s
H    0     0                     7260204.7671 3439609.06  52.6%     -    2s
H    0     0                     7034225.8216 3439609.06  51.1%     -    2s
H    0     0                     7023813.4930 3439609.06  51.0%     -    2s
H    0     0                     6947294.8090 3439609.06  50.5%     -    2s
H    0     0                     6842955.5068 3439609.06  49.7%     -    2s
H    0     0                     6834579.9404 3439609.06  49.7%     -    2s
H    0     0                     6834571.8753 3860922.56  43.5%     -    4s
H    0     0                     4487512.1210 3860922.56  14.0%     -    4s
H    0     0                     4188434.3541 4067343.54  2.89%     -    4s
     0     0 4067343.54    0   16 4188434.35 4067343.54  2.89%     -    4s
H    0     0                     4133330.2727 4067343.54  1.60%     -    8s
     0     0 4133330.27    0   15 4133330.27 4133330.27  0.00%     -    8s

Cutting planes:
  Cover: 1
  Implied bound: 2044
  Flow cover: 145
  Flow path: 183
  Network: 38
  Relax-and-lift: 2
```

```
Explored 1 nodes (16640 simplex iterations) in 8.40 seconds (4.38 work units)
Thread count was 8 (of 8 available processors)

Solution count 10: 4.13333e+06 4.13333e+06 4.18843e+06 … 7.63728e+06

Optimal solution found (tolerance 1.00e-04)
Best objective 4.133330272693e+06, best bound 4.133330272693e+06, gap 0.0000%

Depots to be opened: ['D1', 'D8', 'D18']
Total Cost: 4133330.272693277

Cost for opening Depot D1: 431365.0
Cost for opening Depot D8: 304631.0
Cost for opening Depot D18: 201235.0
Sum of Depot Costs: 937231.0

Total amount of parts shipped to Depot D1 (all parts must also leave the depot):
268989.587474553
Total amount of parts shipped to Depot D8 (all parts must also leave the depot):
127821.12735098171
Total amount of parts shipped to Depot D18 (all parts must also leave the
depot): 78961.49812274048

Depot D1: Total Kilograms = 907448.8301977894
Depot D8: Total Kilograms = 414332.14611775085
Depot D18: Total Kilograms = 260999.54455299827

Depot D1: Total Parts = 268989.587474553, Capacity = 300000, Utilization =
89.66%
Depot D8: Total Parts = 127821.12735148973, Capacity = 150000, Utilization =
85.21%
Depot D18: Total Parts = 78961.49812274048, Capacity = 150000, Utilization =
52.64%

Depot D1 ships to the following countries: AG, AN, AO, AR, AW, BB, BF, BG, BR,
BS, CA, CK, CL, CO, CR, CV, DZ, EC, EE, ES, FI, GA, GT, GU, IS, IT, JM, KY, LC,
LV, LY, MA, MK, MQ, MT, MX, NG, PA, PR, PT, RU, SI, SN, TD, TN, TT, UA, US, UY,
VE, VG
Depot D8 ships to the following countries: BE, CH, CZ, DE, DK, FR, GB, IE, LU,
NL
Depot D18 ships to the following countries: AE, AF, AT, AU, BD, BH, BW, CG, CN,
CY, EG, ET, GE, GR, HR, HU, ID, IL, IN, IQ, JO, JP, KE, KR, KW, KZ, LB, MH, MO,
MU, MV, MY, MZ, NO, NZ, OM, PG, PH, PK, PL, QA, RO, RW, SA, SE, SG, SK, SZ, TH,
TM, TR, TW, TZ, UG, VN, YE, ZA, ZM

Compliant: Shipping from Depot D1 to Country AG at $4.03/kg
Compliant: Shipping from Depot D1 to Country AN at $4.056/kg
Compliant: Shipping from Depot D1 to Country AO at $5.835/kg
```

```
Compliant: Shipping from Depot D1 to Country AR at $5.365/kg
Compliant: Shipping from Depot D1 to Country AW at $4.028/kg
Compliant: Shipping from Depot D1 to Country BB at $4.166/kg
Compliant: Shipping from Depot D1 to Country BF at $4.549/kg
Compliant: Shipping from Depot D1 to Country BG at $4.354/kg
Compliant: Shipping from Depot D1 to Country BR at $4.453/kg
Compliant: Shipping from Depot D1 to Country BS at $3.534/kg
Compliant: Shipping from Depot D1 to Country CA at $3.162/kg
Compliant: Shipping from Depot D1 to Country CK at $5.618/kg
Compliant: Shipping from Depot D1 to Country CL at $4.491/kg
Compliant: Shipping from Depot D1 to Country CO at $4.12/kg
Compliant: Shipping from Depot D1 to Country CR at $3.899/kg
Compliant: Shipping from Depot D1 to Country CV at $4.772/kg
Compliant: Shipping from Depot D1 to Country DZ at $5.009/kg
Compliant: Shipping from Depot D1 to Country EC at $4.335/kg
Compliant: Shipping from Depot D1 to Country EE at $4.946/kg
Compliant: Shipping from Depot D1 to Country ES at $4.76/kg
Compliant: Shipping from Depot D1 to Country FI at $4.935/kg
Compliant: Shipping from Depot D1 to Country GA at $5.698/kg
Compliant: Shipping from Depot D1 to Country GT at $3.836/kg
Compliant: Shipping from Depot D1 to Country GU at $5.54/kg
Compliant: Shipping from Depot D1 to Country IS at $4.332/kg
Compliant: Shipping from Depot D1 to Country IT at $5.08/kg
Compliant: Shipping from Depot D1 to Country JM at $3.776/kg
Compliant: Shipping from Depot D1 to Country KY at $3.69/kg
Compliant: Shipping from Depot D1 to Country LC at $4.127/kg
Compliant: Shipping from Depot D1 to Country LV at $4.982/kg
Compliant: Shipping from Depot D1 to Country LY at $4.451/kg
Compliant: Shipping from Depot D1 to Country MA at $4.875/kg
Compliant: Shipping from Depot D1 to Country MK at $4.423/kg
Compliant: Shipping from Depot D1 to Country MQ at $4.111/kg
Compliant: Shipping from Depot D1 to Country MT at $4.425/kg
Compliant: Shipping from Depot D1 to Country MX at $3.634/kg
Compliant: Shipping from Depot D1 to Country NG at $5.54/kg
Compliant: Shipping from Depot D1 to Country PA at $4.046/kg
Compliant: Shipping from Depot D1 to Country PR at $3.916/kg
Compliant: Shipping from Depot D1 to Country PT at $4.764/kg
Compliant: Shipping from Depot D1 to Country RU at $4.343/kg
Compliant: Shipping from Depot D1 to Country SI at $4.967/kg
Compliant: Shipping from Depot D1 to Country SN at $4.975/kg
Compliant: Shipping from Depot D1 to Country TD at $5.608/kg
Compliant: Shipping from Depot D1 to Country TN at $5.126/kg
Compliant: Shipping from Depot D1 to Country TT at $4.206/kg
Compliant: Shipping from Depot D1 to Country UA at $4.378/kg
Compliant: Shipping from Depot D1 to Country US at $0.872/kg
Compliant: Shipping from Depot D1 to Country UY at $5.319/kg
Compliant: Shipping from Depot D1 to Country VE at $4.12/kg
Compliant: Shipping from Depot D1 to Country VG at $3.945/kg
```

```
Compliant: Shipping from Depot D8 to Country BE at $1.7/kg
Compliant: Shipping from Depot D8 to Country CH at $1.7/kg
Compliant: Shipping from Depot D8 to Country CZ at $1.7/kg
Compliant: Shipping from Depot D8 to Country DE at $1.7/kg
Compliant: Shipping from Depot D8 to Country DK at $1.7/kg
Compliant: Shipping from Depot D8 to Country FR at $1.7/kg
Compliant: Shipping from Depot D8 to Country GB at $1.7/kg
Compliant: Shipping from Depot D8 to Country IE at $1.7/kg
Compliant: Shipping from Depot D8 to Country LU at $0.872/kg
Compliant: Shipping from Depot D8 to Country NL at $0.872/kg
Compliant: Shipping from Depot D18 to Country AE at $4.617/kg
Compliant: Shipping from Depot D18 to Country AF at $4.244/kg
Compliant: Shipping from Depot D18 to Country AT at $4.503/kg
Compliant: Shipping from Depot D18 to Country AU at $5.024/kg
Compliant: Shipping from Depot D18 to Country BD at $3.61/kg
Compliant: Shipping from Depot D18 to Country BH at $4.728/kg
Compliant: Shipping from Depot D18 to Country BW at $5.747/kg
Compliant: Shipping from Depot D18 to Country CG at $5.788/kg
Compliant: Shipping from Depot D18 to Country CN at $0.872/kg
Compliant: Shipping from Depot D18 to Country CY at $5.063/kg
Compliant: Shipping from Depot D18 to Country EG at $4.348/kg
Compliant: Shipping from Depot D18 to Country ET at $4.354/kg
Compliant: Shipping from Depot D18 to Country GE at $4.801/kg
Compliant: Shipping from Depot D18 to Country GR at $4.455/kg
Compliant: Shipping from Depot D18 to Country HR at $4.534/kg
Compliant: Shipping from Depot D18 to Country HU at $4.451/kg
Compliant: Shipping from Depot D18 to Country ID at $3.923/kg
Compliant: Shipping from Depot D18 to Country IL at $5.046/kg
Compliant: Shipping from Depot D18 to Country IN at $4.016/kg
Compliant: Shipping from Depot D18 to Country IQ at $4.758/kg
Compliant: Shipping from Depot D18 to Country JO at $4.954/kg
Compliant: Shipping from Depot D18 to Country JP at $3.786/kg
Compliant: Shipping from Depot D18 to Country KE at $4.524/kg
Compliant: Shipping from Depot D18 to Country KR at $3.526/kg
Compliant: Shipping from Depot D18 to Country KW at $4.774/kg
Compliant: Shipping from Depot D18 to Country KZ at $4.326/kg
Compliant: Shipping from Depot D18 to Country LB at $4.952/kg
Compliant: Shipping from Depot D18 to Country MH at $4.77/kg
Compliant: Shipping from Depot D18 to Country MO at $1.7/kg
Compliant: Shipping from Depot D18 to Country MU at $5.076/kg
Compliant: Shipping from Depot D18 to Country MV at $4.332/kg
Compliant: Shipping from Depot D18 to Country MY at $3.68/kg
Compliant: Shipping from Depot D18 to Country MZ at $5.631/kg
Compliant: Shipping from Depot D18 to Country NO at $4.467/kg
Compliant: Shipping from Depot D18 to Country NZ at $5.485/kg
Compliant: Shipping from Depot D18 to Country OM at $4.541/kg
Compliant: Shipping from Depot D18 to Country PG at $4.358/kg
Compliant: Shipping from Depot D18 to Country PH at $3.239/kg
```

```
Compliant: Shipping from Depot D18 to Country PK at $4.139/kg
Compliant: Shipping from Depot D18 to Country PL at $4.382/kg
Compliant: Shipping from Depot D18 to Country QA at $4.72/kg
Compliant: Shipping from Depot D18 to Country RO at $4.348/kg
Compliant: Shipping from Depot D18 to Country RW at $5.451/kg
Compliant: Shipping from Depot D18 to Country SA at $4.831/kg
Compliant: Shipping from Depot D18 to Country SE at $4.369/kg
Compliant: Shipping from Depot D18 to Country SG at $3.708/kg
Compliant: Shipping from Depot D18 to Country SK at $4.491/kg
Compliant: Shipping from Depot D18 to Country SZ at $5.663/kg
Compliant: Shipping from Depot D18 to Country TH at $3.398/kg
Compliant: Shipping from Depot D18 to Country TM at $4.509/kg
Compliant: Shipping from Depot D18 to Country TR at $4.969/kg
Compliant: Shipping from Depot D18 to Country TW at $3.093/kg
Compliant: Shipping from Depot D18 to Country TZ at $4.526/kg
Compliant: Shipping from Depot D18 to Country UG at $5.372/kg
Compliant: Shipping from Depot D18 to Country VN at $3.071/kg
Compliant: Shipping from Depot D18 to Country YE at $4.962/kg
Compliant: Shipping from Depot D18 to Country ZA at $5.713/kg
Compliant: Shipping from Depot D18 to Country ZM at $5.616/kg

Depot D1 open status: 1.0
Depot D2 open status: 0.0
Depot D3 open status: 0.0
Depot D4 open status: 0.0
Depot D5 open status: 0.0
Depot D6 open status: 0.0
Depot D7 open status: 0.0
Depot D8 open status: 1.0
Depot D9 open status: 0.0
Depot D10 open status: 0.0
Depot D11 open status: 0.0
Depot D12 open status: 0.0
Depot D13 open status: 0.0
Depot D14 open status: 0.0
Depot D15 open status: 0.0
Depot D16 open status: 0.0
Depot D17 open status: 0.0
Depot D18 open status: 1.0
Depot D19 open status: 0.0
```

Making model to test 6 hour flight rule ($4.50/kg)

[ ]:
```
#Re-run the model but with flight_time_rule=4.5 (MODEL IS INFEASIBLE)
flight_time_rule=4.5

m4 = Model("Group Assignment4")
```

```
# Setting up Decsion Variables

depots = fixed_costs_dict.keys()
parts = weights_dict.keys()
suppliers = trans_cost_in_dict.keys()
country_codes = list(next(iter(trans_costs_dict.values())).keys())

#making depot vars (will be binary to reflect open/close decision)
D = m4.addVars(depots, vtype=GRB.BINARY, name="D")

#units of part p from supplier s to depot d
Xpsd = m4.addVars(parts, suppliers, depots, vtype=GRB.CONTINUOUS, name="Xpsd")

#units of part p from depot d to country code cc
Xpdcc = m4.addVars(parts, depots, country_codes, vtype=GRB.CONTINUOUS,␣
 ↪name="Xpdcc")

#non neg constraints
m4.addConstrs((Xpsd[p, s, d] >= 0 for p in parts for s in suppliers for d in␣
 ↪depots), "NonNeg_Xpsd")
m4.addConstrs((Xpdcc[p, d, cc] >= 0 for p in parts for d in depots for cc in␣
 ↪country_codes), "NonNeg_Xpdcc")

m4.update()

#supplier capacity for S2 through S14 (not incldуing s1 as it does not have␣
 ↪same cap, it is unlimited)
m4.addConstrs((
    quicksum(weights_dict[p] * Xpsd[p, s, d] for p in parts for d in depots) <=␣
 ↪200000
    for s in suppliers if s != 'S1'), "SupplierCapacity")


m4.update()

#depot capacities, 300000 for Chicago, 150000 for all else
m4.addConstrs((
    quicksum(Xpsd[p, s, d] for p in parts for s in suppliers) <= D[d] * (300000␣
 ↪if d == 'D1' else 150000)
    for d in depots), "DepotCapacity")

m4.update()

#equality constraints, amount of parts into depot must be equal to amount of␣
 ↪parts out
```

```
m4.addConstrs((
    quicksum(Xpsd[p, s, d] for s in suppliers) == quicksum(Xpdcc[p, d, cc] for␣
 ↪cc in country_codes)
    for p in parts for d in depots), "TransshipmentEquality")

m4.update()

#demand satisfaction constraints

m4.addConstrs((
    quicksum(Xpdcc[p, d, cc] for d in depots) == demand_dict[p][cc]
    for p in parts for cc in country_codes if cc in demand_dict[p]),␣
 ↪"DemandSatisfaction")

m4.update()

#service level constraints

m4.addConstrs((
    Xpdcc[p, d, cc] * weights_dict[p] * trans_costs_dict[d][cc] <= Xpdcc[p, d,␣
 ↪cc] * weights_dict[p] * flight_time_rule
    for p in parts for d in depots for cc in country_codes), "ServiceLevel")

m4.update()

#setting up objective function
objective_func = quicksum(fixed_costs_dict[d] * D[d] for d in depots) + \
            quicksum(weights_dict[p] * trans_cost_in_dict[s][d] * Xpsd[p, s, d]␣
 ↪for p in parts for s in suppliers for d in depots) + \
            quicksum(weights_dict[p] * trans_costs_dict[d][cc] * Xpdcc[p, d,␣
 ↪cc] for p in parts for d in depots for cc in country_codes)

m4.setObjective(objective_func, GRB.MINIMIZE)

m4.optimize()
```

```
Gurobi Optimizer version 11.0.0 build v11.0.0rc2 (win64 - Windows 11+.0
(22631.2))

CPU model: 11th Gen Intel(R) Core(TM) i5-1155G7 @ 2.50GHz, instruction set
[SSE2|AVX|AVX2|AVX512]
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 625634 rows, 320948 columns and 1281322 nonzeros

CPU model: 11th Gen Intel(R) Core(TM) i5-1155G7 @ 2.50GHz, instruction set
[SSE2|AVX|AVX2|AVX512]
```

```
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 625634 rows, 320948 columns and 1281322 nonzeros
Model fingerprint: 0x80fc9290
Variable types: 320929 continuous, 19 integer (19 binary)
Coefficient statistics:
  Matrix range     [1e-03, 3e+05]
  Objective range  [1e+00, 4e+05]
  Bounds range     [1e+00, 1e+00]
  RHS range        [4e-03, 2e+05]
Presolve removed 608076 rows and 269519 columns
Presolve time: 0.33s

Explored 0 nodes (0 simplex iterations) in 0.50 seconds (0.37 work units)
Thread count was 1 (of 8 available processors)

Solution count 0

Model is infeasible
Best objective -, best bound -, gap -
```

Making model for 12 hour rule ($5.50/kg)

[ ]:
```
#Re-run the model but with flight_time_rule=5.5
flight_time_rule=5.5

m5 = Model("Group Assignment5")

# Setting up Decsion Variables

depots = fixed_costs_dict.keys()
parts = weights_dict.keys()
suppliers = trans_cost_in_dict.keys()
country_codes = list(next(iter(trans_costs_dict.values())).keys())

#making depot vars (will be binary to reflect open/close decision)
D = m5.addVars(depots, vtype=GRB.BINARY, name="D")

#units of part p from supplier s to depot d
Xpsd = m5.addVars(parts, suppliers, depots, vtype=GRB.CONTINUOUS, name="Xpsd")

#units of part p from depot d to country code cc
Xpdcc = m5.addVars(parts, depots, country_codes, vtype=GRB.CONTINUOUS,␣
 ↪name="Xpdcc")

#non neg constraints
m5.addConstrs((Xpsd[p, s, d] >= 0 for p in parts for s in suppliers for d in␣
 ↪depots), "NonNeg_Xpsd")
```

```
m5.addConstrs((Xpdcc[p, d, cc] >= 0 for p in parts for d in depots for cc in␣
 ↪country_codes), "NonNeg_Xpdcc")

m5.update()

#supplier capacity for S2 through S14 (not incldуing s1 as it does not have␣
 ↪same cap, it is unlimited)
m5.addConstrs((
    quicksum(weights_dict[p] * Xpsd[p, s, d] for p in parts for d in depots) <=␣
 ↪200000
    for s in suppliers if s != 'S1'), "SupplierCapacity")


m5.update()

#depot capacities, 300000 for Chicago, 150000 for all else
m5.addConstrs((
    quicksum(Xpsd[p, s, d] for p in parts for s in suppliers) <= D[d] * (300000␣
 ↪if d == 'D1' else 150000)
    for d in depots), "DepotCapacity")

m5.update()

#equality constraints, amount of parts into depot must be equal to amount of␣
 ↪parts out
m5.addConstrs((
    quicksum(Xpsd[p, s, d] for s in suppliers) == quicksum(Xpdcc[p, d, cc] for␣
 ↪cc in country_codes)
    for p in parts for d in depots), "TransshipmentEquality")

m5.update()

#demand satisfaction constraints

m5.addConstrs((
    quicksum(Xpdcc[p, d, cc] for d in depots) == demand_dict[p][cc]
    for p in parts for cc in country_codes if cc in demand_dict[p]),␣
 ↪"DemandSatisfaction")

m5.update()

#service level constraints

m5.addConstrs((
    Xpdcc[p, d, cc] * weights_dict[p] * trans_costs_dict[d][cc] <= Xpdcc[p, d,␣
 ↪cc] * weights_dict[p] * flight_time_rule
```

```
    for p in parts for d in depots for cc in country_codes), "ServiceLevel")

m5.update()

#setting up objective function
objective_func = quicksum(fixed_costs_dict[d] * D[d] for d in depots) + \
            quicksum(weights_dict[p] * trans_cost_in_dict[s][d] * Xpsd[p, s, d]␣
 ↪for p in parts for s in suppliers for d in depots) + \
            quicksum(weights_dict[p] * trans_costs_dict[d][cc] * Xpdcc[p, d,␣
 ↪cc] for p in parts for d in depots for cc in country_codes)

m5.setObjective(objective_func, GRB.MINIMIZE)

m5.optimize()

print()
opened_depots = [d for d in depots if D[d].X == 1]
print("Depots to be opened:", opened_depots)

print(f"Total Cost: {m5.ObjVal}")
print()
total_depot_costs = 0
for d in depots:
        depot_cost = fixed_costs_dict[d] * D[d].X
        if D[d].X == 1:
            print(f"Cost for opening Depot {d}: {depot_cost}")
        total_depot_costs += depot_cost

print(f"Sum of Depot Costs: {total_depot_costs}")
print()
total_parts_to_depot = {d: 0 for d in depots}

for p in parts:
    for s in suppliers:
        for d in depots:
            quantity_shipped = Xpsd[p, s, d].X
            total_parts_to_depot[d] += quantity_shipped

for d in D:
    if D[d].X == 1:
        print(f"Total amount of parts shipped to Depot {d} (all parts must also␣
 ↪leave the depot): {total_parts_to_depot[d]}")
print()
total_parts_per_depot = {d: 0 for d in depots}
total_kg_per_depot = {d: 0 for d in depots}
```

```
for p in parts:
    for s in suppliers:
        for d in depots:
            if D[d].X == 1:
                quantity_shipped = Xpsd[p, s, d].X
                if quantity_shipped > 0:
                    total_parts_per_depot[d] += quantity_shipped
                    total_kg_per_depot[d] += quantity_shipped * weights_dict[p]


for p in parts:
    for d in depots:
        if D[d].X == 1:
            for cc in country_codes:
                quantity_shipped = Xpdcc[p, d, cc].X
                if quantity_shipped > 0:
                    total_kg_per_depot[d] += quantity_shipped * weights_dict[p]


for d in D:
    if D[d].X == 1:
        print(f"Depot {d}: Total Kilograms = {total_kg_per_depot[d]}")
print()
depot_capacities = {d: 150000 for d in depots}
depot_capacities['D1'] = 300000

for d in depots:
    if D[d].X == 1:
        total_parts = total_parts_per_depot[d]
        capacity = depot_capacities[d]
        utilization_percentage = (total_parts / capacity) * 100
        print(f"Depot {d}: Total Parts = {total_parts}, Capacity = {capacity},␣
 ↪Utilization = {utilization_percentage:.2f}%")
print()
destinations_per_depot = {d: set() for d in depots}

for p in parts:
    for d in depots:
        for cc in country_codes:
            if Xpdcc[p, d, cc].X > 0:
                destinations_per_depot[d].add(cc)


for d in depots:
    if D[d].X == 1:
        destinations = ", ".join(sorted(destinations_per_depot[d]))
        print(f"Depot {d} ships to the following countries: {destinations}")
```

```
print()
for d in depots:
    for cc in country_codes:
            # Check if there's any shipping from depot d to country code cc
        if any(Xpdcc[p, d, cc].X > 0 for p in parts):
            cost_per_kg = trans_costs_dict[d][cc]
            if cost_per_kg > flight_time_rule:
                print(f"Non-compliance found: Shipping from Depot {d} to␣
 ↪Country {cc} exceeds ${flight_time_rule}/kg at ${cost_per_kg}/kg")
            else:
                print(f"Compliant: Shipping from Depot {d} to Country {cc} at␣
 ↪${cost_per_kg}/kg")
print()
for d in depots:
    print(f"Depot {d} open status: {D[d].X}")
print()
#Adding results to 'results' array
results.append({
    "flight_time_rule": flight_time_rule,
    "opened_depots": len(opened_depots),
    "objective_value": m5.ObjVal,
    "total_depot_costs": total_depot_costs
})
```

```
Gurobi Optimizer version 11.0.0 build v11.0.0rc2 (win64 - Windows 11+.0
(22631.2))

CPU model: 11th Gen Intel(R) Core(TM) i5-1155G7 @ 2.50GHz, instruction set
[SSE2|AVX|AVX2|AVX512]
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 625634 rows, 320948 columns and 1281322 nonzeros

CPU model: 11th Gen Intel(R) Core(TM) i5-1155G7 @ 2.50GHz, instruction set
[SSE2|AVX|AVX2|AVX512]
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 625634 rows, 320948 columns and 1281322 nonzeros
Model fingerprint: 0x4971f4f0
Variable types: 320929 continuous, 19 integer (19 binary)
Coefficient statistics:
  Matrix range     [1e-03, 3e+05]
  Objective range  [1e+00, 4e+05]
  Bounds range     [1e+00, 1e+00]
  RHS range        [4e-03, 2e+05]
Presolve removed 620207 rows and 251451 columns
Presolve time: 0.67s
```

```
Presolved: 5427 rows, 69497 columns, 170344 nonzeros
Variable types: 69478 continuous, 19 integer (19 binary)
Deterministic concurrent LP optimizer: primal and dual simplex
Showing primal log only…

Concurrent spin time: 0.00s

Solved with primal simplex

Root relaxation: objective 3.465020e+06, 11920 iterations, 0.25 seconds (0.10
work units)

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

     0     0 3465020.44    0   17          - 3465020.44      -     -    1s
H    0     0                     7680785.0716 3465020.44  54.9%     -    1s
H    0     0                     7637283.5680 3465020.44  54.6%     -    2s
H    0     0                     7261358.4152 3465020.44  52.3%     -    2s
H    0     0                     7260500.9766 3465020.44  52.3%     -    2s
H    0     0                     7023020.1534 3465020.44  50.7%     -    2s
H    0     0                     7014644.5869 3465020.44  50.6%     -    2s
H    0     0                     6926843.8848 3857429.76  44.3%     -    3s
H    0     0                     6833767.0917 3857429.76  43.6%     -    3s
H    0     0                     6824881.4206 3857429.76  43.5%     -    3s
H    0     0                     4567515.3052 3857429.76  15.5%     -    3s
H    0     0                     4483174.9143 4085147.34  8.88%     -    3s
     0     0 4085147.34    0   16 4483174.91 4085147.34  8.88%     -    3s
H    0     0                     4344933.2869 4085147.34  5.98%     -    4s
     0     0 4195856.45    0   16 4344933.29 4195856.45  3.43%     -    7s
H    0     0                     4231350.9857 4195856.45  0.84%     -    9s
     0     0 4231350.99    0   16 4231350.99 4231350.99  0.00%     -    9s

Cutting planes:
  Cover: 1
  Implied bound: 2199
  Flow cover: 232
  Flow path: 300
  Network: 39
  Relax-and-lift: 13

Explored 1 nodes (16547 simplex iterations) in 9.77 seconds (5.37 work units)
Thread count was 8 (of 8 available processors)

Solution count 10: 4.23135e+06 4.23135e+06 4.34493e+06 … 7.63728e+06

Optimal solution found (tolerance 1.00e-04)
Best objective 4.231350987319e+06, best bound 4.231350985685e+06, gap 0.0000%
```

```
Depots to be opened: ['D1', 'D3', 'D8']
Total Cost: 4231350.987318838

Cost for opening Depot D1: 431365.0
Cost for opening Depot D3: 272284.0
Cost for opening Depot D8: 304631.0
Sum of Depot Costs: 1008280.0

Total amount of parts shipped to Depot D1 (all parts must also leave the depot):
280578.3811055288
Total amount of parts shipped to Depot D3 (all parts must also leave the depot):
51275.364437918994
Total amount of parts shipped to Depot D8 (all parts must also leave the depot):
143918.46740484895

Depot D1: Total Kilograms = 944828.5820747288
Depot D3: Total Kilograms = 168234.94452365814
Depot D8: Total Kilograms = 469716.99415814795

Depot D1: Total Parts = 280578.3811055288, Capacity = 300000, Utilization =
93.53%
Depot D3: Total Parts = 51275.364437918994, Capacity = 150000, Utilization =
34.18%
Depot D8: Total Parts = 143918.46740484895, Capacity = 150000, Utilization =
95.95%

Depot D1 ships to the following countries: AG, AN, AR, AT, AW, BB, BF, BG, BR,
BS, CA, CL, CO, CR, CV, CY, DZ, EC, EE, ES, FI, GE, GR, GT, HR, IS, IT, JM, KY,
KZ, LC, LV, LY, MA, MK, MQ, MT, MX, NO, PA, PL, PR, PT, RO, RU, SE, SI, SK, SN,
TN, TR, TT, UA, US, UY, VE, VG
Depot D3 ships to the following countries: AU, BD, CK, CN, ID, JP, KR, MH, MO,
MU, MV, MY, NZ, PG, PH, SG, TH, TW, VN
Depot D8 ships to the following countries: AE, AF, AO, BE, BH, BW, CG, CH, CZ,
DE, DK, EG, ET, FR, GA, GB, GU, HU, IE, IL, IN, IQ, JO, KE, KW, LB, LU, MZ, NG,
NL, OM, PK, QA, RW, SA, SZ, TD, TM, TZ, UG, YE, ZA, ZM

Compliant: Shipping from Depot D1 to Country AG at $4.03/kg
Compliant: Shipping from Depot D1 to Country AN at $4.056/kg
Compliant: Shipping from Depot D1 to Country AR at $5.365/kg
Compliant: Shipping from Depot D1 to Country AT at $4.958/kg
Compliant: Shipping from Depot D1 to Country AW at $4.028/kg
Compliant: Shipping from Depot D1 to Country BB at $4.166/kg
Compliant: Shipping from Depot D1 to Country BF at $4.549/kg
Compliant: Shipping from Depot D1 to Country BG at $4.354/kg
Compliant: Shipping from Depot D1 to Country BR at $4.453/kg
Compliant: Shipping from Depot D1 to Country BS at $3.534/kg
Compliant: Shipping from Depot D1 to Country CA at $3.162/kg
```

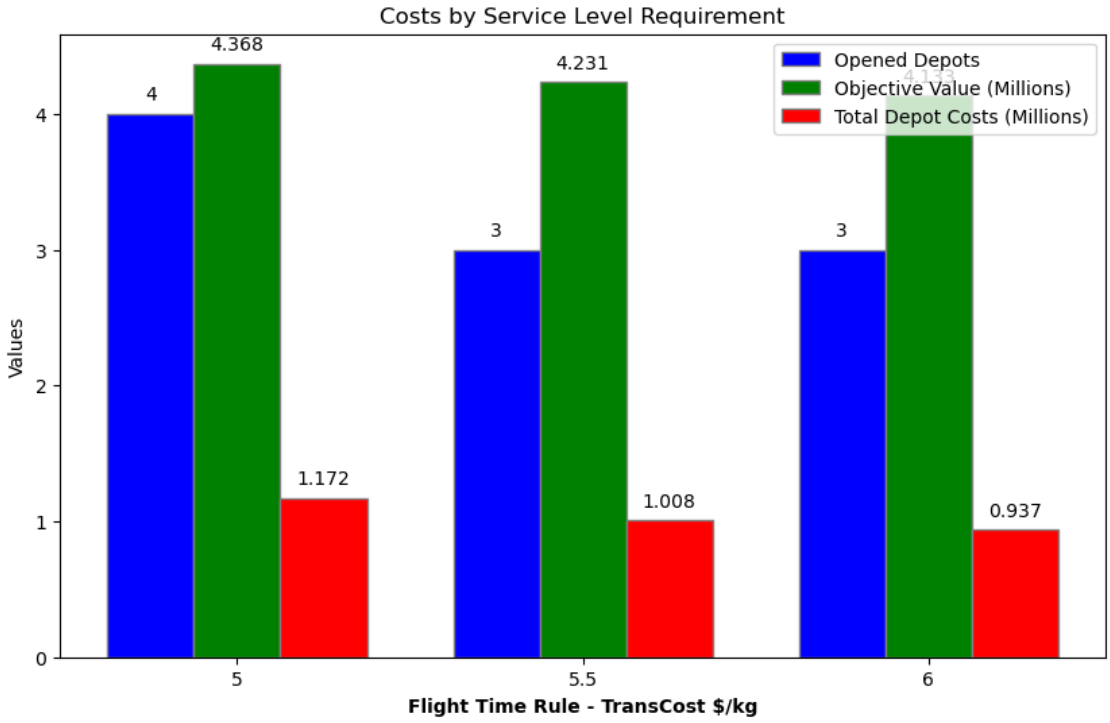
Compliant: Shipping from Depot D1 to Country CL at $4.491/kg
Compliant: Shipping from Depot D1 to Country CO at $4.12/kg
Compliant: Shipping from Depot D1 to Country CR at $3.899/kg
Compliant: Shipping from Depot D1 to Country CV at $4.772/kg
Compliant: Shipping from Depot D1 to Country CY at $5.485/kg
Compliant: Shipping from Depot D1 to Country DZ at $5.009/kg
Compliant: Shipping from Depot D1 to Country EC at $4.335/kg
Compliant: Shipping from Depot D1 to Country EE at $4.946/kg
Compliant: Shipping from Depot D1 to Country ES at $4.76/kg
Compliant: Shipping from Depot D1 to Country FI at $4.935/kg
Compliant: Shipping from Depot D1 to Country GE at $5.487/kg
Compliant: Shipping from Depot D1 to Country GR at $4.541/kg
Compliant: Shipping from Depot D1 to Country GT at $3.836/kg
Compliant: Shipping from Depot D1 to Country HR at $5.067/kg
Compliant: Shipping from Depot D1 to Country IS at $4.332/kg
Compliant: Shipping from Depot D1 to Country IT at $5.08/kg
Compliant: Shipping from Depot D1 to Country JM at $3.776/kg
Compliant: Shipping from Depot D1 to Country KY at $3.69/kg
Compliant: Shipping from Depot D1 to Country KZ at $5.459/kg
Compliant: Shipping from Depot D1 to Country LC at $4.127/kg
Compliant: Shipping from Depot D1 to Country LV at $4.982/kg
Compliant: Shipping from Depot D1 to Country LY at $4.451/kg
Compliant: Shipping from Depot D1 to Country MA at $4.875/kg
Compliant: Shipping from Depot D1 to Country MK at $4.423/kg
Compliant: Shipping from Depot D1 to Country MQ at $4.111/kg
Compliant: Shipping from Depot D1 to Country MT at $4.425/kg
Compliant: Shipping from Depot D1 to Country MX at $3.634/kg
Compliant: Shipping from Depot D1 to Country NO at $4.783/kg
Compliant: Shipping from Depot D1 to Country PA at $4.046/kg
Compliant: Shipping from Depot D1 to Country PL at $5.026/kg
Compliant: Shipping from Depot D1 to Country PR at $3.916/kg
Compliant: Shipping from Depot D1 to Country PT at $4.764/kg
Compliant: Shipping from Depot D1 to Country RO at $4.358/kg
Compliant: Shipping from Depot D1 to Country RU at $4.343/kg
Compliant: Shipping from Depot D1 to Country SE at $4.875/kg
Compliant: Shipping from Depot D1 to Country SI at $4.967/kg
Compliant: Shipping from Depot D1 to Country SK at $4.969/kg
Compliant: Shipping from Depot D1 to Country SN at $4.975/kg
Compliant: Shipping from Depot D1 to Country TN at $5.126/kg
Compliant: Shipping from Depot D1 to Country TR at $5.321/kg
Compliant: Shipping from Depot D1 to Country TT at $4.206/kg
Compliant: Shipping from Depot D1 to Country UA at $4.378/kg
Compliant: Shipping from Depot D1 to Country US at $0.872/kg
Compliant: Shipping from Depot D1 to Country UY at $5.319/kg
Compliant: Shipping from Depot D1 to Country VE at $4.12/kg
Compliant: Shipping from Depot D1 to Country VG at $3.945/kg
Compliant: Shipping from Depot D3 to Country AU at $0.872/kg
Compliant: Shipping from Depot D3 to Country BD at $5.372/kg

```
Compliant: Shipping from Depot D3 to Country CK at $4.393/kg
Compliant: Shipping from Depot D3 to Country CN at $5.353/kg
Compliant: Shipping from Depot D3 to Country ID at $4.53/kg
Compliant: Shipping from Depot D3 to Country JP at $5.1/kg
Compliant: Shipping from Depot D3 to Country KR at $4.345/kg
Compliant: Shipping from Depot D3 to Country MH at $4.406/kg
Compliant: Shipping from Depot D3 to Country MO at $4.999/kg
Compliant: Shipping from Depot D3 to Country MU at $5.386/kg
Compliant: Shipping from Depot D3 to Country MV at $5.325/kg
Compliant: Shipping from Depot D3 to Country MY at $4.811/kg
Compliant: Shipping from Depot D3 to Country NZ at $3.576/kg
Compliant: Shipping from Depot D3 to Country PG at $3.642/kg
Compliant: Shipping from Depot D3 to Country PH at $4.724/kg
Compliant: Shipping from Depot D3 to Country SG at $4.734/kg
Compliant: Shipping from Depot D3 to Country TH at $4.956/kg
Compliant: Shipping from Depot D3 to Country TW at $4.967/kg
Compliant: Shipping from Depot D3 to Country VN at $5.086/kg
Compliant: Shipping from Depot D8 to Country AE at $4.354/kg
Compliant: Shipping from Depot D8 to Country AF at $4.491/kg
Compliant: Shipping from Depot D8 to Country AO at $4.868/kg
Compliant: Shipping from Depot D8 to Country BE at $1.7/kg
Compliant: Shipping from Depot D8 to Country BH at $4.335/kg
Compliant: Shipping from Depot D8 to Country BW at $5.321/kg
Compliant: Shipping from Depot D8 to Country CG at $4.75/kg
Compliant: Shipping from Depot D8 to Country CH at $1.7/kg
Compliant: Shipping from Depot D8 to Country CZ at $1.7/kg
Compliant: Shipping from Depot D8 to Country DE at $1.7/kg
Compliant: Shipping from Depot D8 to Country DK at $1.7/kg
Compliant: Shipping from Depot D8 to Country EG at $3.909/kg
Compliant: Shipping from Depot D8 to Country ET at $4.582/kg
Compliant: Shipping from Depot D8 to Country FR at $1.7/kg
Compliant: Shipping from Depot D8 to Country GA at $4.596/kg
Compliant: Shipping from Depot D8 to Country GB at $1.7/kg
Compliant: Shipping from Depot D8 to Country GU at $4.617/kg
Compliant: Shipping from Depot D8 to Country HU at $3.08/kg
Compliant: Shipping from Depot D8 to Country IE at $1.7/kg
Compliant: Shipping from Depot D8 to Country IL at $3.928/kg
Compliant: Shipping from Depot D8 to Country IN at $4.738/kg
Compliant: Shipping from Depot D8 to Country IQ at $4.058/kg
Compliant: Shipping from Depot D8 to Country JO at $3.938/kg
Compliant: Shipping from Depot D8 to Country KE at $4.823/kg
Compliant: Shipping from Depot D8 to Country KW at $4.125/kg
Compliant: Shipping from Depot D8 to Country LB at $3.88/kg
Compliant: Shipping from Depot D8 to Country LU at $0.872/kg
Compliant: Shipping from Depot D8 to Country MZ at $5.388/kg
Compliant: Shipping from Depot D8 to Country NG at $4.617/kg
Compliant: Shipping from Depot D8 to Country NL at $0.872/kg
Compliant: Shipping from Depot D8 to Country OM at $4.532/kg
```

```
Compliant: Shipping from Depot D8 to Country PK at $4.578/kg
Compliant: Shipping from Depot D8 to Country QA at $4.365/kg
Compliant: Shipping from Depot D8 to Country RW at $4.778/kg
Compliant: Shipping from Depot D8 to Country SA at $4.304/kg
Compliant: Shipping from Depot D8 to Country SZ at $5.388/kg
Compliant: Shipping from Depot D8 to Country TD at $4.278/kg
Compliant: Shipping from Depot D8 to Country TM at $4.136/kg
Compliant: Shipping from Depot D8 to Country TZ at $4.984/kg
Compliant: Shipping from Depot D8 to Country UG at $4.742/kg
Compliant: Shipping from Depot D8 to Country YE at $4.495/kg
Compliant: Shipping from Depot D8 to Country ZA at $5.358/kg
Compliant: Shipping from Depot D8 to Country ZM at $5.108/kg

Depot D1 open status: 1.0
Depot D2 open status: 0.0
Depot D3 open status: 1.0
Depot D4 open status: 0.0
Depot D5 open status: 0.0
Depot D6 open status: 0.0
Depot D7 open status: 0.0
Depot D8 open status: 1.0
Depot D9 open status: 0.0
Depot D10 open status: 0.0
Depot D11 open status: 0.0
Depot D12 open status: 0.0
Depot D13 open status: 0.0
Depot D14 open status: 0.0
Depot D15 open status: 0.0
Depot D16 open status: 0.0
Depot D17 open status: 0.0
Depot D18 open status: 0.0
Depot D19 open status: 0.0
```

[ ]:
```python
import matplotlib.pyplot as plt

# Sorting the results based on flight_time_rule
sorted_results = sorted(results, key=lambda x: x["flight_time_rule"])

flight_time_rules_sorted = [result["flight_time_rule"] for result in␣
 ↪sorted_results]
opened_depots_sorted = [result["opened_depots"] for result in sorted_results]
objective_values_sorted = [result["objective_value"] / 1e6 for result in␣
 ↪sorted_results]
total_depot_costs_sorted = [result["total_depot_costs"] / 1e6 for result in␣
 ↪sorted_results]
```

```
barWidth = 0.25

r1 = range(len(flight_time_rules_sorted))
r2 = [x + barWidth for x in r1]
r3 = [x + barWidth for x in r2]

plt.figure(figsize=(10, 6))
plt.bar(r1, opened_depots_sorted, color='b', width=barWidth, edgecolor='grey',␣
 ↪label='Opened Depots')
plt.bar(r2, objective_values_sorted, color='g', width=barWidth,␣
 ↪edgecolor='grey', label='Objective Value (Millions)')
plt.bar(r3, total_depot_costs_sorted, color='r', width=barWidth,␣
 ↪edgecolor='grey', label='Total Depot Costs (Millions)')

for i in range(len(flight_time_rules_sorted)):
    plt.text(i, opened_depots_sorted[i] + 0.1, f"{opened_depots_sorted[i]:.
 ↪0f}", ha='center')
    plt.text(i + barWidth, objective_values_sorted[i] + 0.1,␣
 ↪f"{objective_values_sorted[i]:.3f}", ha='center')
    plt.text(i + 2*barWidth, total_depot_costs_sorted[i] + 0.1,␣
 ↪f"{total_depot_costs_sorted[i]:.3f}", ha='center')

plt.xlabel('Flight Time Rule - TransCost $/kg', fontweight='bold')
plt.xticks([r + barWidth for r in range(len(flight_time_rules_sorted))],␣
 ↪flight_time_rules_sorted)

plt.ylabel('Values')
plt.title('Costs by Service Level Requirement')
plt.legend()
plt.show()
```


Costs by Service Level Requirement
Opened Depots
Objective Value (Millions)
Total Depot Costs (Millions)
Values
Flight Time Rule - TransCost $/kg
4
4.368
1.172
3
4.231
1.008
3
4.133
0.937
0
1
2
3
4
5
5.5
6