Task 3: Identification of the Objective Function and Constraints

Keniyah Chestnut

Optimization — D605

SID: 012601305

A: OPTIMIZATION PROBLEM PROGRAM

This submission includes the program used to run the Amazon shipping optimization problem in Python using the PuLP library. The problem was modeled using decision variables, constraints, and an objective function developed in Task 2.

A1: DEMONSTRATE SOLVER SOLUTION

Here is the solution provided by the optimization program:

```
Ship 85000.0 tons from CVG to Leipzig
Ship 4397.0 tons from CVG to Hyderabad
Ship 8350.0 tons from AFW to Hyderabad
Ship 36000.0 tons from AFW to San Bernardino
Ship 6500.0 tons from Leipzig to Paris
Ship 640.0 tons from Leipzig to Cologne
Ship 180.0 tons from Leipzig to Hanover
Ship 9100.0 tons from Leipzig to Bangalore
Ship 570.0 tons from Leipzig to Coimbatore
Ship 19000.0 tons from Leipzig to Delhi
Ship 14800.0 tons from Leipzig to Mumbai
Ship 90.0 tons from Leipzig to Cagliari
Ship 185.0 tons from Leipzig to Catania
Ship 800.0 tons from Leipzig to Milan
Ship 1700.0 tons from Leipzig to Rome
Ship 170.0 tons from Leipzig to Katowice
Ship 2800.0 tons from Leipzig to Barcelona
Ship 3700.0 tons from Leipzig to Madrid
Ship 30.0 tons from Leipzig to Castle Donington
Ship 6700.0 tons from Leipzig to London
Ship 190.0 tons from Leipzig to Mobile
Ship 175.0 tons from Leipzig to Anchorage
Ship 1927.0 tons from Leipzig to Phoenix
Ship 7200.0 tons from Leipzig to Los Angeles
Ship 1100.0 tons from Leipzig to Sacramento
Ship 540.0 tons from Leipzig to Hartford
Ship 185.0 tons from Leipzig to Lakeland
Ship 173.0 tons from Leipzig to South Bend
Ship 1700.0 tons from Leipzig to Minneapolis
Ship 480.0 tons from Leipzig to Omaha
Ship 290.0 tons from Leipzig to Toledo
Ship 1100.0 tons from Leipzig to San Juan
Ship 975.0 tons from Leipzig to Austin
Ship 2000.0 tons from Leipzig to Seattle/Tacoma
Ship 38.0 tons from Hyderabad to Fairbanks
Ship 1900.0 tons from Hyderabad to San Francisco
Ship 240.0 tons from Hyderabad to Stockton
Ship 1500.0 tons from Hyderabad to Denver
Ship 1600.0 tons from Hyderabad to Tampa
Ship 500.0 tons from Hyderabad to Honolulu
Ship 16.0 tons from Hyderabad to Kahului/Maui
Ship 63.0 tons from Hyderabad to Kona
Ship 172.0 tons from Hyderabad to Rockford
Ship 550.0 tons from Hyderabad to New Orleans
Ship 1300.0 tons from Hyderabad to Baltimore
```

```
Ship 1300.0 tons from Hyderabad to Baltimore
Ship 975.0 tons from Hyderabad to Kansas City
Ship 450.0 tons from Hyderabad to Albuquerque
Ship 150.0 tons from Hyderabad to Wilmington
Ship 1200.0 tons from Hyderabad to Portland
Ship 420.0 tons from Hyderabad to Allentown
Ship 650.0 tons from Hyderabad to Nashville
Ship 423.0 tons from Hyderabad to Houston
Ship 600.0 tons from Hyderabad to Richmond
Ship 473.0 tons from San Bernardino to Phoenix
Ship 100.0 tons from San Bernardino to Ontario
Ship 1200.0 tons from San Bernardino to Riverside
Ship 3400.0 tons from San Bernardino to Miami
Ship 3000.0 tons from San Bernardino to Atlanta
Ship 5100.0 tons from San Bernardino to Chicago
Ship 200.0 tons from San Bernardino to Fort Wayne
Ship 300.0 tons from San Bernardino to Des Moines
Ship 290.0 tons from San Bernardino to Wichita
Ship 1200.0 tons from San Bernardino to St. Louis
Ship 100.0 tons from San Bernardino to Manchester
Ship 11200.0 tons from San Bernardino to New York
Ship 900.0 tons from San Bernardino to Charlotte
Ship 1000.0 tons from San Bernardino to Pittsburgh
Ship 3300.0 tons from San Bernardino to Dallas
Ship 2877.0 tons from San Bernardino to Houston
Ship 1100.0 tons from San Bernardino to San Antonio
Ship 260.0 tons from San Bernardino to Spokane
```

B: OUTPUT ANALYSIS

The output of the program generated a list of shipping directives that optimize the routes and amounts between hubs, focus cities, and fulfillment centers. Each directive includes the amount of cargo to be shipped along a specific path. In total, the model selected the most cost-effective options based on the objective function and constraints.

Each hub was assigned a specific number of tons to ship, either directly to a center or through a focus city. The values ranged from smaller shipments under 100 tons to larger assignments exceeding several thousand tons. These assignments were based on demand values, capacity limits, and shipping costs as defined in the model.

B1: SATISFYING OPTIMIZATION PROBLEM CONSTRAINTS

In the code, four constraints were checked after the optimization:

[16]:

```python
#Checking constraints: hub capacity
for h in hubs:
    total_shipment_from_hub = sum(x[h][f].varValue for f in focus_cities)
    if total_shipment_from_hub <= hubs[h]["capacity"]:
        print(f"Hub {h} capacity constraint satisfied: {total_shipment_from_hub} <= {hubs[h]['capacity']}")
    else:
        print(f"Hub {h} capacity constraint violated: {total_shipment_from_hub} > {hubs[h]['capacity']}")
```

```
Hub CVG capacity constraint satisfied: 89397.0 <= 95650
Hub AFW capacity constraint satisfied: 44350.0 <= 44350
```

[18]:

```python
#Checking constraints: focus city capacity
for f in focus_cities:
    total_shipment_to_focus_city = sum(x[h][f].varValue for h in hubs)
    if total_shipment_to_focus_city <= focus_cities[f]["capacity"]:
        print(f"Focus city {f} capacity constraint satisfied: {total_shipment_to_focus_city} <= {focus_cities[f]['capacity']}")
    else:
        print(f"Focus city {f} capacity constraint violated: {total_shipment_to_focus_city} > {focus_cities[f]['capacity']}")
```

```
Focus city Leipzig capacity constraint satisfied: 85000.0 <= 85000
Focus city Hyderabad capacity constraint satisfied: 12747.0 <= 19000
Focus city San Bernardino capacity constraint satisfied: 36000.0 <= 36000
```

[20]:

```python
#Checking constraints: center demand
for city, demand in flattened_centers.items():
    total_shipment_to_center = sum(y[f][city].varValue for f in focus_cities)
    if total_shipment_to_center == demand:
        print(f"Center {city} demand constraint satisfied: {total_shipment_to_center} == {demand}")
    else:
        print(f"Center {city} demand constraint violated: {total_shipment_to_center} != {demand}")
```

```
Center Paris demand constraint satisfied: 6500.0 == 6500
Center Cologne demand constraint satisfied: 640.0 == 640
Center Hanover demand constraint satisfied: 180.0 == 180
Center Bangalore demand constraint satisfied: 9100.0 == 9100
Center Coimbatore demand constraint satisfied: 570.0 == 570
Center Delhi demand constraint satisfied: 19000.0 == 19000
Center Mumbai demand constraint satisfied: 14800.0 == 14800
Center Cagliari demand constraint satisfied: 90.0 == 90
Center Catania demand constraint satisfied: 185.0 == 185
Center Milan demand constraint satisfied: 800.0 == 800
Center Rome demand constraint satisfied: 1700.0 == 1700
Center Katowice demand constraint satisfied: 170.0 == 170
Center Barcelona demand constraint satisfied: 2800.0 == 2800
Center Madrid demand constraint satisfied: 3700.0 == 3700
Center Castle Donington demand constraint satisfied: 30.0 == 30
Center London demand constraint satisfied: 6700.0 == 6700
Center Mobile demand constraint satisfied: 190.0 == 190
Center Anchorage demand constraint satisfied: 175.0 == 175
Center Fairbanks demand constraint satisfied: 38.0 == 38
Center Phoenix demand constraint satisfied: 2400.0 == 2400
Center Los Angeles demand constraint satisfied: 7200.0 == 7200
Center Ontario demand constraint satisfied: 100.0 == 100
Center Riverside demand constraint satisfied: 1200.0 == 1200
Center Sacramento demand constraint satisfied: 1100.0 == 1100
Center San Francisco demand constraint satisfied: 1900.0 == 1900
Center Stockton demand constraint satisfied: 240.0 == 240
Center Denver demand constraint satisfied: 1500.0 == 1500
Center Hartford demand constraint satisfied: 540.0 == 540
Center Miami demand constraint satisfied: 3400.0 == 3400
Center Lakeland demand constraint satisfied: 185.0 == 185
Center Tampa demand constraint satisfied: 1600.0 == 1600
Center Atlanta demand constraint satisfied: 3000.0 == 3000
Center Honolulu demand constraint satisfied: 500.0 == 500
Center Kahului/Maui demand constraint satisfied: 16.0 == 16
Center Kona demand constraint satisfied: 63.0 == 63
Center Chicago demand constraint satisfied: 5100.0 == 5100
Center Rockford demand constraint satisfied: 172.0 == 172
```

```
Center Rockford demand constraint satisfied: 172.0 == 172
Center Fort Wayne demand constraint satisfied: 200.0 == 200
Center South Bend demand constraint satisfied: 173.0 == 173
Center Des Moines demand constraint satisfied: 300.0 == 300
Center Wichita demand constraint satisfied: 290.0 == 290
Center New Orleans demand constraint satisfied: 550.0 == 550
Center Baltimore demand constraint satisfied: 1300.0 == 1300
Center Minneapolis demand constraint satisfied: 1700.0 == 1700
Center Kansas City demand constraint satisfied: 975.0 == 975
Center St. Louis demand constraint satisfied: 1200.0 == 1200
Center Omaha demand constraint satisfied: 480.0 == 480
Center Manchester demand constraint satisfied: 100.0 == 100
Center Albuquerque demand constraint satisfied: 450.0 == 450
Center New York demand constraint satisfied: 11200.0 == 11200
Center Charlotte demand constraint satisfied: 900.0 == 900
Center Toledo demand constraint satisfied: 290.0 == 290
Center Wilmington demand constraint satisfied: 150.0 == 150
Center Portland demand constraint satisfied: 1200.0 == 1200
Center Allentown demand constraint satisfied: 420.0 == 420
Center Pittsburgh demand constraint satisfied: 1000.0 == 1000
Center San Juan demand constraint satisfied: 1100.0 == 1100
Center Nashville demand constraint satisfied: 650.0 == 650
Center Austin demand constraint satisfied: 975.0 == 975
Center Dallas demand constraint satisfied: 3300.0 == 3300
Center Houston demand constraint satisfied: 3300.0 == 3300
Center San Antonio demand constraint satisfied: 1100.0 == 1100
Center Richmond demand constraint satisfied: 600.0 == 600
Center Seattle/Tacoma demand constraint satisfied: 2000.0 == 2000
Center Spokane demand constraint satisfied: 260.0 == 260
```

```
[22]: #Checking constraints: flow balance for focus cities
for f in focus_cities:
    total_flow_in = sum(x[h][f].varValue for h in hubs)
    total_flow_out = sum(y[f][city].varValue for city in flattened_centers)
    if total_flow_in == total_flow_out:
        print(f"Flow balance for focus city {f} satisfied: {total_flow_in} == {total_flow_out}")
    else:
        print(f"Flow balance for focus city {f} violated: {total_flow_in} != {total_flow_out}")
```

```
Flow balance for focus city Leipzig satisfied: 85000.0 == 85000.0
Flow balance for focus city Hyderabad satisfied: 12747.0 == 12747.0
Flow balance for focus city San Bernardino satisfied: 36000.0 == 36000.0
```

Note that all of the four above constraints were returned as satisfied and the calculation for each was provided.

B2: DECISION VARIABLES, CONSTRAINTS AND OBJECTIVE FUNCTION

Here is my code defining the decision variables, the constraints, and the objective function:

```python
[12]: #Defining the optimization and variables
prob = LpProblem("Cargo_Optimization", LpMinimize)

x = LpVariable.dicts("shipment_from_hub_to_focus", (hubs.keys(), focus_cities.keys()), lowBound=0, cat='Continuous')
y = LpVariable.dicts("shipment_from_focus_to_center", (focus_cities.keys(), flattened_centers.keys()), lowBound=0, cat='Continuous')

#Defining the objective function to minimize cost
prob += lpSum([get_cost(h, f, cost) * x[h][f] for h in hubs for f in focus_cities]) + \
        lpSum([get_cost(f, c, cost) * y[f][c] for f in focus_cities for c in flattened_centers]), "Total_Shipping_Cost"
```

```python
[14]: #Defining a list of constraints

#1. Hub capacity
for h in hubs:
    prob += lpSum([x[h][f] for f in focus_cities]) <= hubs[h]["capacity"], f"Hub_{h}_Capacity"

#2. Focus city capacity
for f in focus_cities:
    prob += lpSum([x[h][f] for h in hubs]) <= focus_cities[f]["capacity"], f"FocusCity_{f}_Capacity"

#3. Center demand
for city, demand in flattened_centers.items():
    prob += lpSum([y[f][city] for f in focus_cities]) == demand, f"Demand_{city}"

#4. Flow balance for focus cities
for f in focus_cities:
    prob += lpSum([x[h][f] for h in hubs]) == lpSum([y[f][city] for city in flattened_centers]), f"Flow_Balance_{f}"

#Solve the optimization and output results
prob.solve()

if LpStatus[prob.status] == "Optimal":
    for h in hubs:
        for f in focus_cities:
            if x[h][f].varValue > 0:
                print(f"Ship {x[h][f].varValue} tons from {h} to {f}")
    for f in focus_cities:
        for city in flattened_centers:
            if y[f][city].varValue > 0:
                print(f"Ship {y[f][city].varValue} tons from {f} to {city}")
else:
    print("No optimal solution found")
```

As referenced in Task 2, the constraints are hub capacity, focus city capacity, flow conservation, and center requirement. The decision variables were quantity in and quantity out of focus cities. Lastly, here is the objective function, just as it is defined above.

$$\min \left(\sum_{i=1}^{2} \sum_{j=1}^{3} c_{ij} x_{ij} + \sum_{i=1}^{2} \sum_{k=1}^{65} c_{ik} y_{ik} + \sum_{j=1}^{3} \sum_{k=1}^{65} c_{jk} z_{jk} \right)$$

Where:

- c_{ij} is the cost per ton from hub i to focus city j
- c_{ik} is the cost per ton from hub i to center k
- c_{jk} is the cost per ton from focus city j to center k

B3:EXPECTED OUTPUT SOLUTION

Because the decision variables were the shipping amounts between cities, this is the expected output. The output of the optimization, as shown above, consists of shipping routes and the amount of tonnage that is to be sent. After testing the constraints, we can confirm that each center will be at or under capacity and that each center will meet its requirements.

C: REFLECTION

When I first approached this problem, I had already defined the decision variables, constraints, and the objective function and I assumed the optimization calculations would be quite simple. However, I quickly ran into problems with inputting the data. The data was contained in a Word document and when I copied it over to a CSV file, I ran into problems. The disorganization of the data and the lack of a clean dataset was the biggest problem I had to overcome.

However, the data in this project was surprisingly limited and not nearly as big as in past assignments. As a workaround, I was able to just define the data in the program rather than import it, which fixed the problem. I also had trouble with N/A values in the data as the optimizer wouldn't allow them to be processed, so instead I had to replace those values with massive costs to prevent the optimizer from assuming unavailable shipping routes were feasible.

Essentially, my approach developed and changed over time and I had to adjust my expectations of the process of the program as I went. It was a balance between finding ways to get the code to work with the way I had planned to do the project originally.

References

No sources were used besides WGU official course materials.