IE407 - Homework 2 Report

Burak Yıldız - 2449049
Ülkü Aktürk - 2450062

Due Date: 16.05.2024

Abstract

This report outlines the steps taken to solve a transportation optimization problem using Pyomo. The problem involves determining the optimal transportation plan from warehouses to distribution centers and from distribution centers to neighborhoods to minimize total costs. Different cost structures based on delivery volumes are also considered.

Introduction

The transportation problem consists of multiple warehouses with specific capacities, several distribution centers, and several neighborhoods with given demands. The goal is to minimize the total transportation cost while meeting the demands of all neighborhoods. We approach this problem in three parts:

- Part A: Basic transportation optimization.
- Part B: Inclusion of operating costs.
- Part C: Cost adjustments based on delivery volumes.

Part A: Basic Transportation Optimization

Model

The model includes the following components:

- **Sets:** Warehouses, Distribution Centers, Neighborhoods.
- **Parameters:** Supply capacities, demand requirements, transportation costs.
- **Variables:** Amount transported from warehouses to distribution centers, and from distribution centers to neighborhoods.
- **Objective:** Minimize total transportation cost.
- **Constraints:** Supply limits, demand fulfillment.

Sets

$$\begin{aligned} W &: \text{Set of warehouses} \\ DC &: \text{Set of distribution centers} \\ NH &: \text{Set of neighborhoods} \end{aligned}$$

Parameters

$$\begin{aligned} pS[w] &: \text{Supply capacity of warehouse } w \in W \\ pD[nh] &: \text{Demand of neighborhood } nh \in NH \\ pC_{wh\_dc}[w, dc] &: \text{Cost from warehouse } w \text{ to distribution center } dc \\ pC_{dc\_nh}[dc, nh] &: \text{Cost from distribution center } dc \text{ to neighborhood } nh \end{aligned}$$

Variables

$$vX_{wh\_dc}[w, dc] : \text{Amount transported from warehouse } w \text{ to distribution center } dc$$
$$vX_{dc\_nh}[dc, nh] : \text{Amount transported from distribution center } dc \text{ to neighborhood } nh$$

Objective Function

$$\text{Minimize} \sum_{w \in W} \sum_{dc \in DC} pC_{wh\_dc}[w, dc] \cdot vX_{wh\_dc}[w, dc] + \sum_{dc \in DC} \sum_{nh \in NH} pC_{dc\_nh}[dc, nh] \cdot vX_{dc\_nh}[dc, nh]$$

Constraints

$$\sum_{dc \in DC} vX_{wh\_dc}[w, dc] \leq pS[w] \qquad \forall w \in W$$
$$\sum_{dc \in DC} vX_{dc\_nh}[dc, nh] \geq pD[nh] \qquad \forall nh \in NH$$

Code

Pyomo is used to solve, the code can be found in appendix and the pyomo file can be found under the folder Part-A named as TransportModelPartA.py and the data used is under same folder named as TransportData.py which makes use of the file named TransportationData.xlsx

Results for Part A

Optimal Transportation Plan

- Transportation amounts from warehouses to distribution centers can be found in the txt file under the folder Part-A
- Transportation amounts from warehouses to distribution centers can be found in the txt file under the folder Part-A
- Total transportation cost is 179760.0

Part B: Inclusion of Operating Costs

Model

In addition to the basic model, we include fixed operating costs for warehouses and distribution centers.

Additional Parameters

$$pOperating[w] : \text{Operating cost of warehouse } w$$
$$pOperating[dc] : \text{Operating cost of distribution center } dc$$

Additional Variables

$$yW[w] : \text{Binary variable indicating if warehouse } w \text{ is operational}$$
$$yDC[dc] : \text{Binary variable indicating if distribution center } dc \text{ is operational}$$

Objective Function

$$\begin{aligned}
\text{Minimize } & \sum_{w \in W} \sum_{dc \in DC} pC_{wh_dc}[w, dc] \cdot vX_{wh_dc}[w, dc] \\
& + \sum_{dc \in DC} \sum_{nh \in NH} pC_{dc_nh}[dc, nh] \cdot vX_{dc_nh}[dc, nh] \\
& + \sum_{w \in W} pOperating[w] \cdot yW[w] \\
& + \sum_{dc \in DC} pOperating[dc] \cdot yDC[dc]
\end{aligned}$$

Additional Constraints

$$\begin{aligned}
vX_{wh_dc}[w, dc] &\leq pS[w] \cdot yW[w] & \forall w \in W, \forall dc \in DC \\
vX_{dc_nh}[dc, nh] &\leq pCouriers[dc] \cdot yDC[dc] & \forall dc \in DC, \forall nh \in NH
\end{aligned}$$

Code

Pyomo is used to solve , the code can be found in appendix and the pyomo file can be found under the folder Part-B named as TransportModelPartB.py and the data used is under same folder named as TransportData.py which makes use of the file named TransportationData.xlsx. Since the problem is an MIP the sensitivity analysis is not created for this part.

Results for Part B

Optimal Transportation Plan with Operating Costs

- Transportation amounts from warehouses to distribution centers are printed when the code is run , it is provided in the appendix of this report.
- Transportation amounts from warehouses to distribution centers printed when the code is run , it is provided in the appendix of this report.
- Total transportation cost is 748691.0

Part C: Cost Adjustments Based on Delivery Volumes

Model

In this part, the delivery cost from DC1 to any neighborhood decreases from \$2.5 to \$1.5 per kilometer once deliveries exceed 2500 units. The cost adjustment applies only to the amount exceeding the threshold of 2500 units.

Additional Parameters

$$
\begin{aligned}
threshold &: \text{Threshold for cost adjustment (2500 units)} \\
original\_cost\_per\_km &: \text{Original cost per kilometer (\$2.5)} \\
reduced\_cost\_per\_km &: \text{Reduced cost per kilometer (\$1.5)}
\end{aligned}
$$

Additional Variables

$$
\begin{aligned}
vX_{dc1\_nh\_below}[nh] &: \text{Amount transported from DC1 to neighborhoods below threshold} \\
vX_{dc1\_nh\_above}[nh] &: \text{Amount transported from DC1 to neighborhoods above threshold}
\end{aligned}
$$

Additional Constraints

$$
\begin{aligned}
vX_{dc\_nh}[1, nh] = vX_{dc1\_nh\_below}[nh] + vX_{dc1\_nh\_above}[nh] && \forall nh \in NH \\
vX_{dc1\_nh\_below}[nh] \leq threshold && \forall nh \in NH \\
vX_{dc1\_nh\_above}[nh] \geq vX_{dc\_nh}[1, nh] - threshold && \forall nh \in NH
\end{aligned}
$$

Objective Function

$$
\begin{aligned}
\text{Minimize } & \sum_{w \in W} \sum_{dc \in DC} pC_{wh\_dc}[w, dc] \cdot vX_{wh\_dc}[w, dc] \\
& + \sum_{dc \in DC} \sum_{nh \in NH} (pC_{dc\_nh}[dc, nh] \cdot vX_{dc\_nh}[dc, nh]) \\
& + \sum_{nh \in NH} (original\_cost\_per\_km \cdot vX_{dc1\_nh\_below}[nh]) \\
& + \sum_{nh \in NH} (reduced\_cost\_per\_km \cdot vX_{dc1\_nh\_above}[nh]) \\
& + \sum_{w \in W} pOperating_w[w] \cdot yW[w] \\
& + \sum_{dc \in DC} pOperating_dc[dc] \cdot yDC[dc] \\
& + \sum_{dc \in DC} \sum_{dc \in DC} pCouriers[dc] \cdot moto_courier_salary \cdot yDC[dc]
\end{aligned}
$$

Code

Pyomo is used to solve the problem. The code can be found in the appendix and the Pyomo file can be found under the folder Part-C named as TransportModelPartC.py. The data used is in the same folder named TransportData.py, which makes use of the file TransportationData.xlsx. Since the problem is an MIP, the sensitivity analysis is not created for this part.

Results for Part C

Optimal Transportation Plan with Adjusted Costs

- Transportation amounts from warehouses to distribution centers are printed when the code is run and are provided in the appendix of this report.
- Transportation amounts from distribution centers to neighborhoods are printed when the code is run and are provided in the appendix of this report.
- Total transportation cost is 748691.0

Comments

- **Discussion of the effectiveness of the model:**
  - The model effectively identifies the optimal transportation plan, minimizing the total cost while ensuring that the demands of all neighborhoods are met.
  - The inclusion of cost adjustments in Part C based on delivery volumes demonstrates the model's flexibility in handling different cost structures, which is a realistic consideration in logistics planning.
  - The binary variables and additional constraints used in Part C to manage cost reductions were successfully integrated into the model without introducing non-linearities, maintaining the linear structure required for efficient solution using linear programming solvers.
  - The results obtained from each part of the model (A, B, and C) provide a comprehensive view of how different factors (basic costs, operating costs, and volume-based cost adjustments) impact the overall transportation cost.

Conclusion

The model successfully identifies the optimal transportation plan while minimizing costs. Adjustments in cost structures based on delivery volumes significantly contribute to cost savings. Future work may involve more dynamic models and real-time data integration.

Appendix

.1 Part A: Initial Model Code

```
import numpy as np
import TransportData
import pandas as pd
import pyomo.environ as pyo
from pyomo.opt import SolverFactory

# Import the exported dictionaries from TransportData
supplies = TransportData.exports['supplies']
costs_wh_dc = TransportData.exports['costs_wh_dc']
costs_dc_nh = TransportData.exports['costs_dc_nh']
demands = TransportData.exports['demands']
couriers = TransportData.exports['couriers']
moto_courier_capacity = TransportData.exports['moto_courier_capacity']

# Print the sets and dictionary keys for debugging
print("Warehouses:", supplies.keys())
print("Distribution Centers:", couriers.keys())
print("Neighborhoods:", demands.keys())
print("Costs WH to DC keys:", costs_wh_dc.keys())
print("Costs DC to NH keys:", costs_dc_nh.keys())
print("Moto Courier Capacity:", moto_courier_capacity)
print("Moto Couriers:", couriers)

# Construct the model
mdl = pyo.ConcreteModel('TransportModel')

# Define sets
mdl.W = pyo.Set(initialize=supplies.keys(), doc='Warehouses')
mdl.DC = pyo.Set(initialize=couriers.keys(), doc='Distribution Centers'
    )
mdl.NH = pyo.Set(initialize=demands.keys(), doc='Neighborhoods')

# Define parameters
mdl.pS = pyo.Param(mdl.W, initialize=supplies, doc='Capacity of each
    warehouse')
mdl.pD = pyo.Param(mdl.NH, initialize=demands, doc='Demand of each
    neighborhood')
mdl.pC_wh_dc = pyo.Param(mdl.W, mdl.DC, initialize=costs_wh_dc, doc='
    Cost from warehouse to distribution center')
mdl.pC_dc_nh = pyo.Param(mdl.DC, mdl.NH, initialize=costs_dc_nh, doc='
    Cost from distribution center to neighborhood')
mdl.pCouriers = pyo.Param(mdl.DC, initialize=couriers, doc='Number of
    moto couriers')

# Define variables
mdl.vX_wh_dc = pyo.Var(mdl.W, mdl.DC, within=pyo.NonNegativeReals, doc=
    'Amount transported from warehouse to distribution center')
mdl.vX_dc_nh = pyo.Var(mdl.DC, mdl.NH, within=pyo.NonNegativeReals, doc
    ='Amount transported from distribution center to neighborhood')

# Define constraints
# Demand levels at neighborhoods
def eDemandLevel(mdl, nh):
```

```python
    return sum(mdl.vX_dc_nh[dc, nh] for dc in mdl.DC) >= mdl.pD[nh]
mdl.eDemandLevel = pyo.Constraint(mdl.NH, rule=eDemandLevel, doc='Demand level constraint')

# Supply capacities at warehouses
def eSupplyCap(mdl, w):
    return sum(mdl.vX_wh_dc[w, dc] for dc in mdl.DC) <= mdl.pS[w]
mdl.eSupplyCap = pyo.Constraint(mdl.W, rule=eSupplyCap, doc='Supplier capacity constraint')

# Moto courier capacities at distribution centers
def eCourierCap(mdl, dc):
    return sum(mdl.vX_dc_nh[dc, nh] for nh in mdl.NH) <= mdl.pCouriers[dc] * moto_courier_capacity
mdl.eCourierCap = pyo.Constraint(mdl.DC, rule=eCourierCap, doc='Courier capacity constraint')

# Ensure flow conservation at distribution centers
def eFlowConservation(mdl, dc):
    return sum(mdl.vX_wh_dc[w, dc] for w in mdl.W) == sum(mdl.vX_dc_nh[dc, nh] for nh in mdl.NH)
mdl.eFlowConservation = pyo.Constraint(mdl.DC, rule=eFlowConservation, doc='Flow conservation at distribution centers')

# Define objective function to find the  minimum cost of serving neighborhoods.
def oTotal_Cost(mdl):
    transportation_cost = sum(mdl.pC_wh_dc[w, dc] * mdl.vX_wh_dc[w, dc] for w in mdl.W for dc in mdl.DC)
    delivery_cost = sum(mdl.pC_dc_nh[dc, nh] * mdl.vX_dc_nh[dc, nh] for dc in mdl.DC for nh in mdl.NH)

    return transportation_cost + delivery_cost
mdl.oTotal_Cost = pyo.Objective(rule=oTotal_Cost, sense=pyo.minimize, doc='Total Transportation Cost')

# Export the open form of the model (optional)
mdl.write('mdl.lp', io_options={'symbolic_solver_labels': True})
mdl.dual = pyo.Suffix(direction=pyo.Suffix.IMPORT)  # shadow prices of the constraints
mdl.rc = pyo.Suffix(direction=pyo.Suffix.IMPORT)  # reduced costs of the objective function coefficients

# Solve the model
Solver = SolverFactory('glpk')

# Print the sensitivity analysis and output report
Solver.options['ranges'] = '/Users/burak/Desktop/IE-HW2/Part-A/Output_PartA.txt'

# The orders of constraints and variables are as in the Suffix Declarations
SolverResults = Solver.solve(mdl, tee=True)
SolverResults.write()

# Print the results
mdl.pprint()
mdl.vX_wh_dc.display()
```

```
mdl.vX_dc_nh.display()
mdl.oTotal_Cost.display()
```

TransportModelPartA.py

.2 Part B: Initial Model Code

```
import numpy as np
import pyomo.environ as pyo
from pyomo.opt import SolverFactory
import TransportData

# Data from the problem statement
# Import the exported dictionaries from TransportData
supplies = TransportData.exports['supplies']
costs_wh_dc = TransportData.exports['costs_wh_dc']
costs_dc_nh = TransportData.exports['costs_dc_nh']
demands = TransportData.exports['demands']
couriers = TransportData.exports['couriers']
warehouse_operating_costs = TransportData.exports['
    warehouse_operating_costs']
dc_operating_costs = TransportData.exports['dc_operating_costs']
moto_courier_salary = TransportData.exports['moto_courier_salary']
moto_courier_capacity = TransportData.exports['moto_courier_capacity']

# Construct the model
mdl = pyo.ConcreteModel('TransportModel')

# Define sets
mdl.W = pyo.Set(initialize=supplies.keys(), doc='Warehouses')
mdl.DC = pyo.Set(initialize=couriers.keys(), doc='Distribution Centers'
    )
mdl.NH = pyo.Set(initialize=demands.keys(), doc='Neighborhoods')

# Define parameters
mdl.pS = pyo.Param(mdl.W, initialize=supplies, doc='Capacity of each
    warehouse')
mdl.pD = pyo.Param(mdl.NH, initialize=demands, doc='Demand of each
    neighborhood')
mdl.pC_wh_dc = pyo.Param(mdl.W, mdl.DC, initialize=costs_wh_dc, doc='
    Cost from warehouse to distribution center')
mdl.pC_dc_nh = pyo.Param(mdl.DC, mdl.NH, initialize=costs_dc_nh, doc='
    Cost from distribution center to neighborhood')
mdl.pCouriers = pyo.Param(mdl.DC, initialize=couriers, doc='Number of
    moto couriers')
mdl.pOperating_w = pyo.Param(mdl.W, initialize=
    warehouse_operating_costs, doc='Operating costs of warehouses')
mdl.pOperating_dc = pyo.Param(mdl.DC, initialize=dc_operating_costs,
    doc='Operating costs of distribution centers')

# Define variables
mdl.vX_wh_dc = pyo.Var(mdl.W, mdl.DC, within=pyo.NonNegativeReals, doc=
    'Amount transported from warehouse to distribution center')
mdl.vX_dc_nh = pyo.Var(mdl.DC, mdl.NH, within=pyo.NonNegativeReals, doc
    ='Amount transported from distribution center to neighborhood')

```

```python
# Part B: Define continuous variables for operating warehouses and distribution centers (0 to 1)
mdl.yW = pyo.Var(mdl.W, within=pyo.Binary, doc='Warehouse operating decision')
mdl.yDC = pyo.Var(mdl.DC, within=pyo.Binary, doc='Distribution center operating decision')

# Define constraints

# Demand levels at neighborhoods
def eDemandLevel(mdl, nh):
    return sum(mdl.vX_dc_nh[dc, nh] for dc in mdl.DC) >= mdl.pD[nh]
mdl.eDemandLevel = pyo.Constraint(mdl.NH, rule=eDemandLevel, doc='Demand level constraint')

# Supply capacities at warehouses
def eSupplyCap(mdl, w):
    return sum(mdl.vX_wh_dc[w, dc] for dc in mdl.DC) <= mdl.pS[w] * mdl.yW[w]
mdl.eSupplyCap = pyo.Constraint(mdl.W, rule=eSupplyCap, doc='Supplier capacity constraint')

# Moto courier capacities at distribution centers
def eCourierCap(mdl, dc):
    return sum(mdl.vX_dc_nh[dc, nh] for nh in mdl.NH) <= mdl.pCouriers[dc] * moto_courier_capacity * mdl.yDC[dc]
mdl.eCourierCap = pyo.Constraint(mdl.DC, rule=eCourierCap, doc='Courier capacity constraint')

# Ensure flow conservation only for operating distribution centers
def eFlowConservation(mdl, dc):
    return sum(mdl.vX_wh_dc[w, dc] for w in mdl.W) == sum(mdl.vX_dc_nh[dc, nh] for nh in mdl.NH)
mdl.eFlowConservation = pyo.Constraint(mdl.DC, rule=eFlowConservation, doc='Flow conservation at operating distribution centers')

# Ensure flow conservation at operating warehouses
def eFlowConservationWH(mdl, w):
    return sum(mdl.vX_wh_dc[w, dc] for dc in mdl.DC) <= sum(mdl.vX_dc_nh[dc, nh] for dc in mdl.DC for nh in mdl.NH) + 1e6 * (1 - mdl.yW[w])
mdl.eFlowConservationWH = pyo.Constraint(mdl.W, rule=eFlowConservationWH, doc='Flow conservation at operating warehouses')

# Part B: Ensure that transport only happens if the facility is operational
def eTranspFromWhToDc(mdl, w, dc):
    return mdl.vX_wh_dc[w, dc] <= mdl.pS[w] * mdl.yW[w]
mdl.eTranspFromWhToDc = pyo.Constraint(mdl.W, mdl.DC, rule=eTranspFromWhToDc, doc='Transport from warehouse to DC only if operational')

def eTranspFromDcToNh(mdl, dc, nh):
    return mdl.vX_dc_nh[dc, nh] <= mdl.pCouriers[dc] * moto_courier_capacity * mdl.yDC[dc]
mdl.eTranspFromDcToNh = pyo.Constraint(mdl.DC, mdl.NH, rule=eTranspFromDcToNh, doc='Transport from DC to NH only if operational'
```

```
    )

# Define objective function to find the minimum cost of serving
    neighborhoods, including fixed costs and salaries for part b.
def oTotal_Cost(mdl):
    # Transportation cost from warehouses to distribution centers
    transportation_cost = sum(mdl.pC_wh_dc[w, dc] * mdl.vX_wh_dc[w, dc]
        for w in mdl.W for dc in mdl.DC)

    # Delivery cost from distribution centers to neighborhoods
    delivery_cost = sum(mdl.pC_dc_nh[dc, nh] * mdl.vX_dc_nh[dc, nh] for
        dc in mdl.DC for nh in mdl.NH)

    # Operating costs for warehouses and distribution centers
    operating_cost = sum(mdl.pOperating_w[w] * mdl.yW[w] for w in mdl.W
        ) + sum(mdl.pOperating_dc[dc] * mdl.yDC[dc] for dc in mdl.DC)

    # Part B: Include the salaries of moto couriers
    courier_salaries = sum(mdl.pCouriers[dc] * moto_courier_salary *
        mdl.yDC[dc] for dc in mdl.DC)

    return transportation_cost + delivery_cost + operating_cost +
        courier_salaries

mdl.oTotal_Cost = pyo.Objective(rule=oTotal_Cost, sense=pyo.minimize,
    doc='Total Transportation Cost')

# Export the open form of the model (optional)
mdl.write('mdl.lp', io_options={'symbolic_solver_labels': True})
mdl.dual = pyo.Suffix(direction=pyo.Suffix.IMPORT)  # shadow prices of
    the constraints
mdl.rc = pyo.Suffix(direction=pyo.Suffix.IMPORT)  # reduced costs of
    the objective function coefficients

# Solve the model
Solver = SolverFactory('glpk')

# Can not Print the sensitivity analysis and output report since this
    is an MIP

# The orders of constraints and variables are as in the Suffix
    Declarations
SolverResults = Solver.solve(mdl, tee=True)
SolverResults.write()

# Print the results
mdl.pprint()
mdl.vX_wh_dc.display()
mdl.vX_dc_nh.display()
mdl.oTotal_Cost.display()
```

TransportModelPartB.py

Transportation amounts from warehouses to distribution centers and distribution centers to neighborhoods:

```
vX_wh_dc : Amount transported from warehouse to distribution center
    Size=15, Index=W*DC
```

```
    Key    : Lower : Value  : Upper : Fixed : Stale : Domain
    (1, 1) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (1, 2) :     0 : 9000.0 :  None : False : False : NonNegativeReals
    (1, 3) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (1, 4) :     0 : 5400.0 :  None : False : False : NonNegativeReals
    (1, 5) :     0 : 7100.0 :  None : False : False : NonNegativeReals
    (2, 1) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (2, 2) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (2, 3) :     0 : 8100.0 :  None : False : False : NonNegativeReals
    (2, 4) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (2, 5) :     0 : 3400.0 :  None : False : False : NonNegativeReals
    (3, 1) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (3, 2) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (3, 3) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (3, 4) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (3, 5) :     0 :    0.0 :  None : False : False : NonNegativeReals
vX_dc_nh : Amount transported from distribution center to neighborhood
    Size=30, Index=DC*NH
    Key    : Lower : Value  : Upper : Fixed : Stale : Domain
    (1, 1) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (1, 2) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (1, 3) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (1, 4) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (1, 5) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (1, 6) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (2, 1) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (2, 2) :     0 : 1500.0 :  None : False : False : NonNegativeReals
    (2, 3) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (2, 4) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (2, 5) :     0 : 4500.0 :  None : False : False : NonNegativeReals
    (2, 6) :     0 : 3000.0 :  None : False : False : NonNegativeReals
    (3, 1) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (3, 2) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (3, 3) :     0 : 4000.0 :  None : False : False : NonNegativeReals
    (3, 4) :     0 :  600.0 :  None : False : False : NonNegativeReals
    (3, 5) :     0 : 3500.0 :  None : False : False : NonNegativeReals
    (3, 6) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (4, 1) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (4, 2) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (4, 3) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (4, 4) :     0 : 5400.0 :  None : False : False : NonNegativeReals
    (4, 5) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (4, 6) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (5, 1) :     0 : 5000.0 :  None : False : False : NonNegativeReals
    (5, 2) :     0 : 5500.0 :  None : False : False : NonNegativeReals
    (5, 3) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (5, 4) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (5, 5) :     0 :    0.0 :  None : False : False : NonNegativeReals
```

```
(5, 6) :     0 :    0.0 :  None : False : False : NonNegativeReals
```

.3 Part C: Initial Model Code

```python
import numpy as np
import TransportData
import pandas as pd
import pyomo.environ as pyo
from pyomo.opt import SolverFactory

# Import the exported dictionaries from TransportData
supplies = TransportData.exports['supplies']
costs_wh_dc = TransportData.exports['costs_wh_dc']
costs_dc_nh = TransportData.exports['costs_dc_nh']
demands = TransportData.exports['demands']
couriers = TransportData.exports['couriers']
moto_courier_capacity = TransportData.exports['moto_courier_capacity']
moto_courier_salary = TransportData.exports['moto_courier_salary']
warehouse_operating_costs = TransportData.exports['warehouse_operating_costs']
dc_operating_costs = TransportData.exports['dc_operating_costs']

# Construct the model
mdl = pyo.ConcreteModel('TransportModel')

# Define sets
mdl.W = pyo.Set(initialize=supplies.keys(), doc='Warehouses')
mdl.DC = pyo.Set(initialize=couriers.keys(), doc='Distribution Centers')
mdl.NH = pyo.Set(initialize=demands.keys(), doc='Neighborhoods')

# Define parameters
mdl.pS = pyo.Param(mdl.W, initialize=supplies, doc='Capacity of each warehouse')
mdl.pD = pyo.Param(mdl.NH, initialize=demands, doc='Demand of each neighborhood')
mdl.pC_wh_dc = pyo.Param(mdl.W, mdl.DC, initialize=costs_wh_dc, doc='Cost from warehouse to distribution center')
mdl.pC_dc_nh = pyo.Param(mdl.DC, mdl.NH, initialize=costs_dc_nh, doc='Cost from distribution center to neighborhood')
mdl.pCouriers = pyo.Param(mdl.DC, initialize=couriers, doc='Number of moto couriers')
mdl.pOperating_w = pyo.Param(mdl.W, initialize=warehouse_operating_costs, doc='Operating costs of warehouses')
mdl.pOperating_dc = pyo.Param(mdl.DC, initialize=dc_operating_costs, doc='Operating costs of distribution centers')

# Define variables
mdl.vX_wh_dc = pyo.Var(mdl.W, mdl.DC, within=pyo.NonNegativeReals, doc='Amount transported from warehouse to distribution center')
mdl.vX_dc_nh = pyo.Var(mdl.DC, mdl.NH, within=pyo.NonNegativeReals, doc='Amount transported from distribution center to neighborhood')

# Additional variables for part C
mdl.vX_dc1_nh_below = pyo.Var(mdl.NH, within=pyo.NonNegativeReals, doc='Amount transported from DC1 to neighborhoods below threshold')
```

```python
mdl.vX_dc1_nh_above = pyo.Var(mdl.NH, within=pyo.NonNegativeReals, doc='Amount transported from DC1 to neighborhoods above threshold')

# Part B: Define binary variables for operating warehouses and distribution centers (0 or 1)
mdl.yW = pyo.Var(mdl.W, within=pyo.Binary, doc='Warehouse operating decision')
mdl.yDC = pyo.Var(mdl.DC, within=pyo.Binary, doc='Distribution center operating decision')

# Define constraints
threshold_deliveries = 2500
original_cost_per_km = 2.5
reduced_cost_per_km = 1.5

# Demand levels at neighborhoods
def eDemandLevel(mdl, nh):
    return sum(mdl.vX_dc_nh[dc, nh] for dc in mdl.DC) >= mdl.pD[nh]
mdl.eDemandLevel = pyo.Constraint(mdl.NH, rule=eDemandLevel, doc='Demand level constraint')

# Supply capacities at warehouses
def eSupplyCap(mdl, w):
    return sum(mdl.vX_wh_dc[w, dc] for dc in mdl.DC) <= mdl.pS[w]
mdl.eSupplyCap = pyo.Constraint(mdl.W, rule=eSupplyCap, doc='Supplier capacity constraint')

# Moto courier capacities at distribution centers
def eCourierCap(mdl, dc):
    return sum(mdl.vX_dc_nh[dc, nh] for nh in mdl.NH) <= mdl.pCouriers[dc] * moto_courier_capacity
mdl.eCourierCap = pyo.Constraint(mdl.DC, rule=eCourierCap, doc='Courier capacity constraint')

# Ensure flow conservation only for operating distribution centers
def eFlowConservation(mdl, dc):
    return sum(mdl.vX_wh_dc[w, dc] for w in mdl.W) == sum(mdl.vX_dc_nh[dc, nh] for nh in mdl.NH)
mdl.eFlowConservation = pyo.Constraint(mdl.DC, rule=eFlowConservation, doc='Flow conservation at operating distribution centers')

# Ensure flow conservation at operating warehouses
def eFlowConservationWH(mdl, w):
    return sum(mdl.vX_wh_dc[w, dc] for dc in mdl.DC) <= sum(mdl.vX_dc_nh[dc, nh] for dc in mdl.DC for nh in mdl.NH) + 1e6 * (1 - mdl.yW[w])
mdl.eFlowConservationWH = pyo.Constraint(mdl.W, rule=eFlowConservationWH, doc='Flow conservation at operating warehouses')

# Part B: Ensure that transport only happens if the facility is operational
def eTranspFromWhToDc(mdl, w, dc):
    return mdl.vX_wh_dc[w, dc] <= mdl.pS[w] * mdl.yW[w]
mdl.eTranspFromWhToDc = pyo.Constraint(mdl.W, mdl.DC, rule=eTranspFromWhToDc, doc='Transport from warehouse to DC only if operational')

```

```python
def eTranspFromDcToNh(mdl, dc, nh):
    return mdl.vX_dc_nh[dc, nh] <= mdl.pCouriers[dc] * moto_courier_capacity * mdl.yDC[dc]
mdl.eTranspFromDcToNh = pyo.Constraint(mdl.DC, mdl.NH, rule=eTranspFromDcToNh, doc='Transport from DC to NH only if operational')

# Splitting the deliveries for DC1 into two parts
def eSplitDeliveries(mdl, nh):
    return mdl.vX_dc_nh[1, nh] == mdl.vX_dc1_nh_below[nh] + mdl.vX_dc1_nh_above[nh]
mdl.eSplitDeliveries = pyo.Constraint(mdl.NH, rule=eSplitDeliveries, doc='Splitting deliveries for DC1')

# Ensuring the correct costs for the split deliveries
def eThresholdCostBelow(mdl, nh):
    return mdl.vX_dc1_nh_below[nh] <= threshold_deliveries
mdl.eThresholdCostBelow = pyo.Constraint(mdl.NH, rule=eThresholdCostBelow, doc='Cost for deliveries below threshold')

def eThresholdCostAbove(mdl, nh):
    return mdl.vX_dc1_nh_above[nh] >= mdl.vX_dc_nh[1, nh] - threshold_deliveries
mdl.eThresholdCostAbove = pyo.Constraint(mdl.NH, rule=eThresholdCostAbove, doc='Cost for deliveries above threshold')

# Define objective function to find the minimum cost of serving neighborhoods
def oTotal_Cost(mdl):
    transportation_cost = sum(mdl.pC_wh_dc[w, dc] * mdl.vX_wh_dc[w, dc] for w in mdl.W for dc in mdl.DC)
    delivery_cost = sum(mdl.pC_dc_nh[dc, nh] * mdl.vX_dc_nh[dc, nh] for dc in mdl.DC for nh in mdl.NH if dc != 'Distribution Center 1')

    # Delivery costs for DC1 split into two parts
    delivery_cost += sum(original_cost_per_km * mdl.vX_dc1_nh_below[nh] for nh in mdl.NH)
    delivery_cost += sum(reduced_cost_per_km * mdl.vX_dc1_nh_above[nh] for nh in mdl.NH)

    # Operating costs for warehouses and distribution centers
    operating_cost = sum(mdl.pOperating_w[w] * mdl.yW[w] for w in mdl.W) + sum(mdl.pOperating_dc[dc] * mdl.yDC[dc] for dc in mdl.DC)

    # Part B: Include the salaries of moto couriers
    courier_salaries = sum(mdl.pCouriers[dc] * moto_courier_salary * mdl.yDC[dc] for dc in mdl.DC)

    return transportation_cost + delivery_cost + operating_cost + courier_salaries

mdl.oTotal_Cost = pyo.Objective(rule=oTotal_Cost, sense=pyo.minimize, doc='Total Transportation Cost')

# Export the open form of the model (optional)
mdl.write('mdl.lp', io_options={'symbolic_solver_labels': True})
mdl.dual = pyo.Suffix(direction=pyo.Suffix.IMPORT)  # shadow prices of the constraints
```

```
mdl.rc = pyo.Suffix(direction=pyo.Suffix.IMPORT)  # reduced costs of the objective function coefficients

# Solve the model
solver = SolverFactory('glpk')

# Cannot print the sensitivity analysis and output report since this is an MIP

# The orders of constraints and variables are as in the Suffix Declarations
solver_results = solver.solve(mdl, tee=True)
solver_results.write()

# Print the results
mdl.pprint()
mdl.vX_wh_dc.display()
mdl.vX_dc_nh.display()
mdl.oTotal_Cost.display()
```

TransportModelPartC.py

Transportation amounts from warehouses to distribution centers and distribution centers to neighborhoods:

```
vX_wh_dc : Amount transported from warehouse to distribution center
    Size=15, Index=W*DC
    Key    : Lower : Value  : Upper : Fixed : Stale : Domain
    (1, 1) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (1, 2) :     0 : 9000.0 :  None : False : False : NonNegativeReals
    (1, 3) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (1, 4) :     0 : 5400.0 :  None : False : False : NonNegativeReals
    (1, 5) :     0 : 7100.0 :  None : False : False : NonNegativeReals
    (2, 1) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (2, 2) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (2, 3) :     0 : 8100.0 :  None : False : False : NonNegativeReals
    (2, 4) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (2, 5) :     0 : 3400.0 :  None : False : False : NonNegativeReals
    (3, 1) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (3, 2) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (3, 3) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (3, 4) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (3, 5) :     0 :    0.0 :  None : False : False : NonNegativeReals
vX_dc_nh : Amount transported from distribution center to neighborhood
    Size=30, Index=DC*NH
    Key    : Lower : Value  : Upper : Fixed : Stale : Domain
    (1, 1) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (1, 2) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (1, 3) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (1, 4) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (1, 5) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (1, 6) :     0 :    0.0 :  None : False : False : NonNegativeReals
    (2, 1) :     0 :    0.0 :  None : False : False : NonNegativeReals
```

```
(2, 2) :     0 : 1500.0 :  None : False : False : NonNegativeReals
(2, 3) :     0 :    0.0 :  None : False : False : NonNegativeReals
(2, 4) :     0 :    0.0 :  None : False : False : NonNegativeReals
(2, 5) :     0 : 4500.0 :  None : False : False : NonNegativeReals
(2, 6) :     0 : 3000.0 :  None : False : False : NonNegativeReals
(3, 1) :     0 :    0.0 :  None : False : False : NonNegativeReals
(3, 2) :     0 :    0.0 :  None : False : False : NonNegativeReals
(3, 3) :     0 : 4000.0 :  None : False : False : NonNegativeReals
(3, 4) :     0 :  600.0 :  None : False : False : NonNegativeReals
(3, 5) :     0 : 3500.0 :  None : False : False : NonNegativeReals
(3, 6) :     0 :    0.0 :  None : False : False : NonNegativeReals
(4, 1) :     0 :    0.0 :  None : False : False : NonNegativeReals
(4, 2) :     0 :    0.0 :  None : False : False : NonNegativeReals
(4, 3) :     0 :    0.0 :  None : False : False : NonNegativeReals
(4, 4) :     0 : 5400.0 :  None : False : False : NonNegativeReals
(4, 5) :     0 :    0.0 :  None : False : False : NonNegativeReals
(4, 6) :     0 :    0.0 :  None : False : False : NonNegativeReals
(5, 1) :     0 : 5000.0 :  None : False : False : NonNegativeReals
(5, 2) :     0 : 5500.0 :  None : False : False : NonNegativeReals
(5, 3) :     0 :    0.0 :  None : False : False : NonNegativeReals
(5, 4) :     0 :    0.0 :  None : False : False : NonNegativeReals
(5, 5) :     0 :    0.0 :  None : False : False : NonNegativeReals
(5, 6) :     0 :    0.0 :  None : False : False : NonNegativeReals
```