Multi-Commodity Transportation Problems:
Modeling and Programming Approach for Solution

Shervin Iranaghideh
SRBIAU
Supervisor: Dr. Farhad Hosseinzadeh Lotfi

Abstract

Multi-commodity Transportation Problems (MCTP) are a type of optimization problems that involves the allocation of resources to transport different commodities from various sources to various destinations. In this study I present a mathematical modeling of MCTP and a step-by-step code for solving them using SciPy Python library. While the real-world scenarios are immensely complex, here I present a simplified example to illustrate the problem with respect to the basic structure of these problems. This study provides a clear and accessible starting point for understanding the fundamental principles of multi-commodity transportation problems as well as how to solve them using programming.

1. Introduction

Multi-commodity transportation problems are one type of Multi-commodity Flow Problems (MCFP), which are optimization problems that the goal is to minimize the total cost of transportation of multiple types of good from different sources to different destinations.

MCTP have many applications in various real-life fields, such as supply chain management, logistics, transportation planning and network design. Here I am describing a simple example of the multi-commodity transportation problems.

2. Mathematical Modeling

To describe the problem let's assume there is a firm producing p kinds of goods at n different supply centers. We label these $\ell = 1, \ldots, p$ and $i = 1, \ldots, n$ respectively. The amount of each ℓ supply produced at each supply center i is $S_{i,\ell}$. There are also m different demand centers that we label them $j = 1, \ldots, m$. The demand of ℓth good at jth center is $D_{j,\ell}$. The objective is transporting goods from the supply centers to the demand centers at a minimum cost. Let's assume that the cost of shipping one unit of product ℓ from supply center i to demand center j is $c_{i,j,\ell}$ and that shipping cost is linear meaning that if we shipped $x_{i,j,\ell}$ units of product ℓ from supply center i to demand center j, then the cost would be $c_{i,j,\ell}\, x_{i,j,\ell}$.

To formulate the problem, lets define $x_{i,j,\ell}$ to be the number of units of the product ℓ shipped from supply center i to demand center j as said above, which in our linear programming problem are called the Decision Variables. The objective is to minimize the total cost of the shipping which we define as our Objective Function:

$$min \sum_{i} \sum_{j} \sum_{\ell} c_{i,j,\ell}\, x_{i,j,\ell}$$

Which $c_{i,j,\ell}$ denotes the cost of shipping one unit of product ℓ from supply center i to demand center j.

The problem is subject to the following constraints: the total amount of product shipped to a center should satisfy its demands, and the total amount of product shipped from each supply center should not exceed the available supply at that center.

Now consider the supply center i, the total product ℓ shipped out of the supply center i is $\sum_{i=1}^{n} x_{i,j,\ell}$ $(for\ all\ j = 1, \ \ldots, \ m)$. This sum is all the shipments from supply center i which cannot surpass the quantity available at that supply center. Hence, we have the constraints

$$\sum_{j} x_{i,j,\ell} \ \leq S_{i,\ell} \ for \ \forall i, \ell$$

in which $S_{i,\ell}$ represents the amount of product ℓ available at supply center i.

Likewise, the constraints that denote that we satisfy the demand of each demand centers are:

$$\sum_{i} x_{i,j,\ell} = D_{j,\ell} \ for \ \forall j, \ell$$

in which $D_{j,\ell}$ is the demand of product ℓ at supply center j.

There is another constraint which is the maximum amount that can be shipped in one visit (assuming each supply center only ships products a maximum of one time to each demand center), this constraint is a lower bound and an upper bound for our decision variables.

The number of units shipped clearly cannot be negative so we have one constraint for that, and also the amount shipped cannot exceed our upper bound (denoting it by u), so to formulate this we have:

$$0 \leq x_{i,j,\ell} \leq u \ for \ \forall i, j, \ell$$

We can formulate this problem as a linear programming problem as follows:

$$min \sum_{i} \sum_{j} \sum_{\ell} c_{i,j,\ell} \, x_{i,j,\ell}$$

$$S.T. \quad \sum_{j} x_{i,j,\ell} \ \leq S_{i,\ell} \ for \ \forall i, \ell$$

$$\sum_{i} x_{i,j,\ell} = D_{j,\ell} \ for \ \forall j, \ell$$

$$0 \leq x_{i,j,\ell} \leq u \ for \ \forall i, j, \ell$$

3. An Illustrative Example

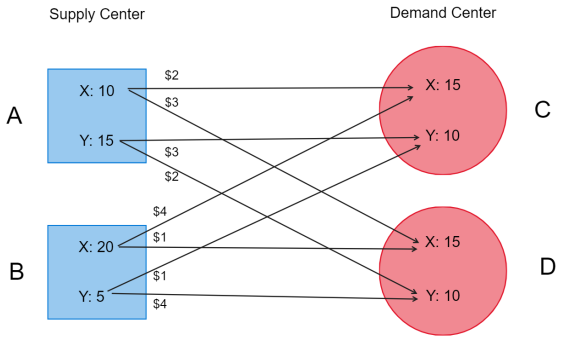
Suppose we have two supply centers, A and B. Center A has 10 units of good X and 15 units of good Y. Center B has 20 units of good X and 5 units of good Y.

We also have two demand centers, C and D. Center C requires 15 units of good X and 10 units of good Y. Center D requires 15 units of good X and 10 units of good Y.

The cost of transporting one unit of good X from center A to center C is $2, from center A to center D is $3, from center B to center C is $4, and from center B to center D is $1.

The cost of transporting one unit of good Y from center A to center C is $3, from center A to center D is $2, from center B to center C is $1, and from center B to center D is $4.

Find the optimal transportation plan that minimizes the total transportation cost.



4. Solving the Problem Using Python SciPy

These problems (MCTP) can be solved manually using Simplex method, but doing so is highly time consuming, so using programming is more reasonable. Here we are using scipy.optimize to solve the problem , the library has a method called linprog which takes different parameters, here we are explaining how to organize multi-commodity problem data so we can use them as a parameter for this function.

First we have to import the libraries we want to work with, here I'm using NumPy [4] for easier data manipulation, so we need to import NumPy in addition to scipy.optimize :

```
import numpy as np
from scipy.optimize import linprog
```

Then we are going to store our data into variables so we can manipulate them. First we define the number of supply centers, demand centers and product types, and we call them m, n and p respectively. We also dedicate a list for each supply center and demand center in which we include the supply and demand of each product in order and respectively.

```
# Number of supply centers
m = 2

# Number of demand centers
n = 2

# Number of goods
p = 2

# Supply limit for each good at each supply
center (m * p matrix)
supply = [[10, 15], [20, 5]]

# Demand amount for each good at each demand
center (n * p matrix)
demand = [[15, 10], [15, 10]]

# Cost of sending each good to each demand
center from each supply center (m * n * p
matrix)
cost = [[[2, 3], [3, 2]], [[4, 1], [1, 4]]]
```

As it is shown [10, 15] in supply means that in supply center 1, the demand for product 1 is 10 and the demand for product 2 is 15.

Likewise [15, 10] in demand means the in the first demand center the demand for product 1 is 15 and the demand for product 2 is 10.

We also added the costs of each transportation, as it is shown in the cost list there are 2 nested lists which represent our supply centers, then in each of the two nested lists there are 2 other nested lists representing our demand centers in which the demand for each product is included respectively.

The lingprog function [2][3] takes the following parameters:

Parameters:

c : 1-D array

The coefficients of the linear objective function to be minimized.

A_ub : 2-D array, optional

The inequality constraint matrix. Each row of A_ub specifies the coefficients of a linear inequality constraint on x.

b_ub : 1-D array, optional

The inequality constraint vector. Each element represents an upper bound on the corresponding value of A_ub @ x.

A_eq : 2-D array, optional

The equality constraint matrix. Each row of A_eq specifies the coefficients of a linear equality constraint on x.

b_eq : 1-D array, optional

The equality constraint vector. Each element of A_eq @ x must equal the corresponding element of b_eq.

As said before c is the list of the coefficients of our objective function which is in order of our decision variables in respect of their index so for making c we simply have:

```
# Objective function coefficients
c = []
for i in range(m):
    for j in range(n):
        for k in range(p):
            c.append(cost[i][j][k])
```

For A_ub and b_ub which are our inequality constraints coefficients we have:

```
# Inequality constraints (supply constraints)
A_ub = []
b_ub = []
for i in range(m):
    for k in range(p):
        zero = np.zeros((m, n, p))
        for j in range(n):
            zero[i][j][k] = 1
        zero = zero.reshape(-1).tolist()
        A_ub.append(zero)
        b_ub.append(supply[i][k])
```

For A_eq and b_eq which are our equality constraints coefficients we have:

```
# Equality constraints:
A_eq = []
b_eq = []
for j in range(n):
    for k in range(p):
        zero = np.zeros((m, n, p))
        for i in range(m):
            zero[i][j][k] = 1
        zero = zero.reshape(-1).tolist()
        A_eq.append(zero)
        b_eq.append(demand[j][k])
```

Now that we organized our data for the function, we can use them:

```
solver = linprog(c, A_ub=A_ub, b_ub=b_ub, A_eq=A_eq, b_eq=b_eq, bounds=(0, 100), method='highs')
```

The program outputs the optimal cost of $95.0 and optimal values [10. 5. 0. 10. 5. 5. 15. 0.] for our decision variables.

Executable code:

```
import numpy as np
from scipy.optimize import linprog

m = 2

n = 2

p = 2

supply = [[10, 15], [20, 5]]

demand = [[15, 10], [15, 10]]

cost = [[[2, 3], [3, 2]], [[4, 1], [1, 4]]]

# Objective function coefficients
c = []
for i in range(m):
    for j in range(n):
        for k in range(p):
            c.append(cost[i][j][k])

# Inequality constraints (supply constraints)
A_ub = []
b_ub = []
for i in range(m):
    for k in range(p):
        zero = np.zeros((m, n, p))
        for j in range(n):
            zero[i][j][k] = 1
        zero = zero.reshape(-1).tolist()
        A_ub.append(zero)
        b_ub.append(supply[i][k])

# Equality constraints:
A_eq = []
b_eq = []
for j in range(n):
    for k in range(p):
        zero = np.zeros((m, n, p))
        for i in range(m):
            zero[i][j][k] = 1
        zero = zero.reshape(-1).tolist()
        A_eq.append(zero)
        b_eq.append(demand[j][k])

solver = linprog(c, A_ub=A_ub, b_ub=b_ub, A_eq=A_eq, b_eq=b_eq, bounds=(0, 100), method='highs')

print(f"Status: {solver.message}")
print(f"Optimal Value Found: {solver.success}")
print(f"Optimal Value: ${solver.fun}")
print(f"Optimal Solution: {solver.x}")
```

4. About

This study is done in purpose of gaining experience in conducting research and enhancing my understanding of the Linear Optimization course I took and also getting familiar with writing a research paper in addition to gaining experience in solving these kinds of problems using Python.

The code for this project which is completed with a simple UI for solving these kind of transportation problems using Python can be found here.

References

- Joel Sobel, Linear Programming Notes VIII: The Transportation Problem, University of California San Diego.
- [2] https://docs.scipy.org/doc/scipy/reference/optimize.linprog-highs.html Scipy Python library linprog API.
- [3] Huangfu, Q., Galabova, I., Feldmeier, M., and Hall, J. A. J. "HiGHS - high performance software for linear optimization." https://highs.dev/
- [4] NumPy (Version: 1.24.3) https://www.numpy.org
- Figure made with Microsoft Whiteboard.