An Improved Transport Optimization Model in Python

Fall 2013

Michael Lee

ABSTRACT

Transportation problem are a classic example of linear optimization problem, appearing in many commercial supply chain models. Building off the work of economists Koopermans (1951) and Dorfman, Samuelson, and Solow (1958) who's collective work form the foundation of transport economics. Expanding upon the original algebraic models, computational methods allow for a rapidly adaptable formula for solving such linear problems. In the case presented, the dynamic language python was chosen to minimize cost between supply nodes and demand nodes given the respective costs of each transport route.

Initial Design

A Brooke, D Kendrick and A Meeraus (1988) develop a simple model of where two suppliers ship to three destinations. Each destination is a predefined distance away, and the cost per mile is a constant value for each route:

Supply\Demand	Chicago	Topeka	New York City
Seattle	1.7	1.8	2.5
San Diego	1.8	1.4	2.5

Table 1: Distance to Between Each Node

Total cost of shipment (G_{ij}) is equal to the number of miles times the cost per mile for each route. Total cost calculations are minimized via linear optimization. The optimization algorithm chooses the quantity shipped from each supply node to each destination node via routes *ij* such that:

$$G_{ij} = C_{ij} * d_{ij}$$
$$\sum_{r=1}^{R} Q_i^{ij} \leq \sum_{i=1}^{n} S_i$$
$$\sum_{r=1}^{R} Q_i^{ij} \geq \sum_{i=1}^{m} D_i$$

where Q_i^{ij} represents the quantity shipped on each route (r) from *i* to *j*, subject to the constraints that the total quantity shipped must be simultaneously less than or equal to the total available supply and greater than or equal to the total demand. To be exhaustive, it is also specified that Q_i^{ij} must be nonnegative.

Kendrick et al.’s original model is programming in the General Algebraic Modeling System (GAMS), a high-level language built around mathematical modeling and optimization. Their model produces the following results:

INSERT RESULTS HERE

Improving the Model

The original model faces serious limitations, including the omission of varying labor costs and retail prices in different markets, and by positing that the shipment costs are constant across all routes. Additionally, any excess supply is unaccounted for, incurring no storage or disposal costa on the supplier. Computationally, GAMS is limited in its applications and cannot be rapidly interfaced with other resources such as databases and the Internet.

Variations in Cost per Mile

In the real world, the quality of roads and the availability of alternative shipping methods create vastly different costs per mile for different routes. For example, shipping via truck from New York City, NY to Boston, MA would be cheaper per mile than shipping from New York City, NY to

Pittsburg, PA since trucks would have to cross mountainous terrain on the latter. This increased cost is related to the sum paid to the drivers, which is a function the time it takes to reach each destination.

$$C_{ij} = w * t_{ij}$$

Thus, a better approximation of the shipment costs can be calculated instead by using tools such as Google Maps to find the time it takes to reach each destination, and multiplying that by a some constant wage (this assumes that wages paid by trucking companies are independent of supply/demand node locations—a valid approximation).

Supply\Demand	Chicago, IS	Topeka, KS	New York, NY
Seattle, WA	29	26	41
San Diego, CA	29	22	40

Table 2: Travel Time Between Each Node (hrs)

Assuming that the company subcontracts its shipping (thus incurring no capital depreciation of the trucks themselves) and hourly wage of a truck driver is 22USD per hour, the total cost decreases from $1613USD to $1400 USD. The cost savings can be attributed to the fall in the effective price per mile from $90USD to $2.73USD (assuming the truck moves at a constant 60 mph). Also, a transport unit (T)—loosely defined as the amount of goods that can be transported from supplier to demander at a time—is used in cost calculation instead of the raw supply. This is meant to replicate the carrying capacity of a single truck or train, and can be parameterized to study how perturbations in packing efficiency or truck capacity can alter the company's costs.

Warehousing

The proposed transportation model adds a warehouse in which surplus supply can be stored. Initially the warehouse is chosen to be located centrally to the demanders, and costs are incurred for the transport to the warehouse. When supply outweighs demand, the warehouse acts as an extra demander, whose demand is equal to the difference between total supply and total demand.

$$D_{warehouse} = \sum_{i=1}^{n} S_i - \sum_{j=1}^{m} D_j$$

However, when the demanded quantity is larger than the available supply, the surplus in the warehouse can be shipped to supplement the factories. In this situation, the model treats the warehouse as an extra supplier, who incurs costs proportional to the distance to each destination.

In the initial model, the warehouse is located equidistance from all demand centers such that the cost to ship from warehouse to any of the possible destinations is minimized. Using the supplied values, the warehouse receives 50 additional units. There is an additional overhead cost associated with leasing the facility. Since the model represents one period, the inventory shipped to the warehouse would be unaccounted in profit calculations if some positive $sellPrice$ were not attributed to the warehouse. While there is no customer demand at the warehouse, the company can warehouse surplus to be used in the next period (not modeled). In the proposed model, the warehouse's price level is found by averaging the $sellPrice$ of each demand node times some depreciation factor δ (originally chosen to be .3).

$$P_{warehouse} = \sum_{j=1}^{n} P_j * (1 - \delta)$$

Labor Costs and Retail Prices

In order to more fully evaluate the profit of the company, labor costs at each supply factor and retail prices at each demand node must be accounted for. From this data, the total profit of the company can be calculated as:

$$\pi = \sum_{j=1}^{n} D_j * P_j - \sum_{i=1}^{m} S_i * w_i^S - \sum_{r=1}^{R} \frac{t_r * Q_r}{T} * w_T - Lease$$

With the above calculation, potential markets can be evaluated on a profit-maximization basis as opposed to a strictly cost-minimization basis (the latter being biased towards short transport times).

Experiments

Experiments were constructed to test the effects of opening new markets. Effects of changing exogenous, specified variables—namely $\{sellPrice\}, \{distance\},$ and $\{costMile\}$— were not studied, as they perturbations in these would alter the profit linearly and will provide no insight into optimization.

New Markets

Two new markets were opened, expanding the transport time matrix shown in **Figure 2** to the following:

Supply\Demand	Chicago, IL	Topeka, KS	New York, NY	Albuquerque, NM	Boston, MA	Warehouse
Seattle, WA	29	26	41	21	44	20
San Diego, CA	29	22	40	11	44	21
Warehouse	6	6	6	6	6	0

Figure 3: Updated Travel Time Between Each Node

The sell prices at Albuquerque, NM and Boston, MA are set to be \$8 and \$13 respectively. The results are tabulated in **Appendix 1**. The profit per route – a measure of your most profitable route per unit good shipped-- can be computed as follows:

$$\pi_{ij} = \frac{P_{j-} w_i^S}{t_{ij} * w_T}$$

Revenue\Route	Chicago, IL	Topeka, KS	New York, NY	Albuquerque, NM	Boston, MA	Warehouse
Seattle, WA	.0109	.0052	.0121	.0108	.0103	.0091
San Diego, CA	.0125	.0082	.0136	.0248	.0113	.0089
Warehouse	.0682	.0378	.0984	.0530	.0919	0

Figure 4: Profit Per Route

Based on the profit per route calculations, the model suggests that the company should pullout of the Topeka, KS market as it is provides the lowest returns per shipped good. Another observation is that the San Diego, CA to Albuquerque, NM route is the most profitable. While Albuquerque's prices are the second lowest in the model, the time it takes to ship along this route is less than half of other routes. The warehouse's proximity to the demand nodes affords it a highly profitable route to all markets.

Building upon the benefits of warehouse centrality, it can be inferred that if the warehouse was instead modeled as a distribution center, and a low cost method of transport was established between supply nodes and distribution center—i.e. a favorable contract with a railroad operator (w_T') or an increased transport capacity (T') — then profits could be increased. So long as the inequality below holds:

$$\frac{w_T' * (t_{S \to dist})}{T'} + \frac{w_T * (t_{dist \to j})}{T} \leq \frac{w_T * (t_{ij})}{T}$$

<u>Further Expansion of the Model</u>

One implication of the availability of warehousing is that it opens the possibility for induced demand via discounting. This means that the company can maximize profit by decreasing price in profitable markets until the price was less than or equal to that of the warehouse. Given some known marginal propensity to consume (MPS), the company can dynamically discount the price it sells the goods as it moves along the demand curve increasing quantity supplied. This can be modeled computationally using the following psuedocode:

```
for p in range(P_eq: P_ware):
   Q_induced = (p - P_max) * MPS
   Revenue = p * Q_induced
```

Given more time, transshipment nodes could be used in place of direct routes. This would create a true transport network, where few direct routes exist. Additionally, the location of the warehouse could be optimized to minimize total costs. By predefining various location options for the next warehouse/distribution center, the algorithm could select whichever location would minimize costs given distance to supply and demand nodes, and the city's labor costs.

Appendix 1: Experimental Results

```
%run
"/var/folders/c0/28r9j3zd3g77n9g93m7fll3r0000gn/T/tmpg_gyGO.py"
main() Excess Supply... shipping to warehouse
('status:', 'Optimal')
```

- ('Route_SD_Albuquerque', ' = ', 100.0)
- ('Route_SD_Boston', ' = ', 125.0)
- ('Route_SD_Chi', ' = ', 0.0)
- ('Route_SD_NY', ' = ', 300.0)
- ('Route_SD_Top', ' = ', 275.0)
- ('Route_SD_Warehouse', ' = ', 0.0)
- ('Route_Sea_Albuquerque', ' = ', 0.0)
- ('Route_Sea_Boston', ' = ', 25.0)
- ('Route_Sea_Chi', ' = ', 325.0)
- ('Route_Sea_NY', ' = ', 0.0)
- ('Route_Sea_Top', ' = ', 0.0)
- ('Route_Sea_Warehouse', ' = 850.0)
- ('Route_Warehouse_Albuquerque', ' = ', 0.0)
- ('Route_Warehouse_Boston', ' = ', 0.0)
- ('Route_Warehouse_Chi', ' = ', 0.0)
- ('Route_Warehouse_NY', ' = ', 0.0)
- ('Route_Warehouse_Top', ' = ', 0.0)
- ('Route_Warehouse_Warehouse', ' = ', 0.0)
- ('total cost of transport = ', 8700.0)
- Profit is equal to: 400.0

Appendix 2: Python Code

```
from pulp import *
import numpy as np

def equilibrium(supply, demand, sellPrice, Lease):
# create list of all supply nodes
Suppliers = ['Sea', 'SD', 'Warehouse']
# create a dict of all demand nodes
Destinations = ['Chi', 'NY', 'Top', 'Warehouse', 'Albuquerque', 'Boston']

excess = sum(supply.values()) - sum(demand.values())
#positve excess values = extra supply, negative = extra demand

# add in excess to a warehouse
if excess > 0:
demand['Warehouse'] = excess
print('Excess Supply... shipping to warehouse')

elif excess < 0:
supply['Warehouse'] = excess
print('Excess Demand... using warehouse inventories')

else:
print('Supply = Demand')

## create a time array
dist = np.array( #Destinations
# Chi NY Top warehouse Alb. Bos
[(29, 41, 26, 10, 21, 44),#Sea
(29, 40, 22, 10, 11, 44),#SD
( 6, 6, 6, 6, 6, 6) #Warehouse
])

# assuming hourly wage is 22USD
costMile = np.array([ #Destinations
#Chi NY Top War Al Bos
(22, 22, 22, 22, 22, 22),#Sea
(22, 22, 22, 22, 22, 22),#SD
(22, 22, 22, 22, 22, 22) #Warehouse
])

# specifiy how much each transport container can hold
capacity = 100

# create labor price dict. & caclulate total labor cost
lab = {}
laborPrice = {'Sea': 3,
'SD': 2,
'Warehouse': 1}
for key, Value in supply.items():
lab[key] = laborPrice[key] * Value
# create revenue matrix
rev = {}
for key, Value in demand.items():
rev[key] = Value * sellPrice[key]
```

```
# create a cost matrix and convert to a list
costs = (costMile * dist) / capacity
costs = costs.tolist()

costs = makeDict([Suppliers, Destinations], costs, 0)
# setup the cost minimization problem
prob = LpProblem('Canning Distribution Problem', LpMinimize)

# create a list of all possible routes
routes = [(S,D) for S in Suppliers for D in Destinations]

# create a dictionary to contain all referenced variables
route_var = LpVariable.dicts("Route", (Suppliers, Destinations), 0, None, LpInteger)

# add the objective function to the problem
prob += lpSum([route_var[S][D]*costs[S][D] for (S,D) in routes])

# constraints functions
for S in Suppliers:
prob += lpSum([route_var[S][D] for D in Destinations]) <= supply[S]
"sum_of_products_out_of_suppliers_%s" %S
for D in Destinations:
prob += lpSum([route_var[S][D] for S in Suppliers]) >= demand[D]
"sum_of_products_into_destinations_%s" %D

prob.writeLP('Transport_problem.lp')
prob.solve()

print( "status:", LpStatus[prob.status])

for v in prob.variables():
print(v.name, ' = ', v.varValue)
# calculate profit
profit = sum(rev.values()) - sum(lab.values()) - value(prob.objective) - Lease

print( 'total cost of transport = ', value(prob.objective))

return profit
```

def main():

```
#assume a MPC
MPC = .5
depreciation = .3
Lease = 100
# create a dict with the inventories of each warehouse
supply_0 = {'Sea': 1200,
'SD' : 800,
'Warehouse': 0}
demand_0 = {'Chi': 325,
'NY' : 300,
'Top': 275,
'Warehouse': 0,
'Albuquerque': 100,
'Boston': 150}
```

```
# create sell price array
sellPrice_0 = {'Chi': 10,
'NY': 14,
'Top': 6,
'Warehouse': ((10+14+6)/3)* depreciation,
'Albuquerque': 8,
'Boston': 13}


prof = equilibrium(supply_0, demand_0, sellPrice_0, Lease)
print('Profit is equal to: ' + str(prof))
```