Applied Electronics Chips Production

Neil Jiang   Icy Wang   Jingyi Huang   Foster Mosden
Team 4

EMORY
GOIZUETA BUSINESS SCHOOL

Problem Outline

Background

Applied Electronics provides semiconductor chips for customers all over the world from its six fabrication plants Mexico, Canada, Chile, Frankfurt, Austin, and Japan. Plants were originally designed to produce the quantity of chips required for the country where they were located, but as demand changes, there is the potential to produce chips in one country and ship them to another.

Project Objective

Shipping products to different regions introduces new factors into cost, including shipping costs and duties. This project helps AE construct a new model to find an efficient production plan for next year, and it also considers the scenarios plants running at 85% capacities, and shrinking down the supply of the most difficult plant for AE to support, Chile, to zero.

Methodology

What is VAM, and why should we use it?

Vogel's Approximation Method, or VAM, is an efficient manner of finding a feasible initial solution to a transportation problem that satisfies all constraints:

i. Identify the smallest cost in each row and column of the cost matrix
ii. Find the difference between the smallest costs within rows and columns
iii. Select cell with the largest difference; allocate maximum possible units
iv. Adjust the matrix by subtracting allocated units, create zeros if necessary
v. Repeat Steps 1-4 until all supplies and demands are met

VAM does not necessarily provide the *best* solution, but it *does* guarantee a balanced, initial solution to transportation problems.

Input Data Structure

- To start, employees at AE can input their data into our clearly labeled and defined Python dictionaries and arrays
- These values can be easily edited and the code re-ran for any changes in this problem or for similar new problems
- We found this method better than having users create a .csv or .xlsx in another program, and then import that
    - Keeping everything in one code file was the simplest approach for usability in deployment

```python
############################################ Input Data ############################################

# Input production data
data = {
    "Country": ["Mexico", "Canada", "Chile", "Frankfurt", "Austin", "Japan"],
    "Capacity (millions of chips)": [22.0, 3.7, 4.5, 47.0, 18.5, 5.0],
    "Demand (millions of chips)": [3.00, 2.60, 16.00, 20.00, 26.40, 11.90],
    "Production Costs (per box of 100 chips)": [92.63, 93.25, 112.31, 73.34, 89.15, 149.24],
    "Import Duties (percent of production cost)": ["60.0%", "0.0%", "50.0%", "9.5%", "4.5%", "6.0%"]
}

# Convert production data
matrix = {key: values for key, values in data.items()}

# Input shipping cost data
shipping_data = {
    "Country": ["Mexico", "Canada", "Chile", "Frankfurt", "Austin", "Japan"],
    "Mexico": ["-", 11.40, 7.00, 11.00, 11.00, 14.00],
    "Canada": [11.00, "-", 9.00, 11.50, 6.00, 13.00],
    "Chile": [7.00, 10.00, "-", 13.00, 10.40, 14.30],
    "Frankfurt": [10.00, 11.50, 12.50, "-", 11.20, 13.30],
    "Austin": [10.00, 6.00, 11.00, 10.00, "-", 12.50],
    "Japan": [14.00, 13.00, 12.50, 14.20, 13.00, "-"]
}

# Input the capacity of each location
supply = np.array([22, 3.7, 4.5, 47, 18.5, 5])

# input the demand of each location
demand = np.array([3, 2.6, 16, 20, 26.4, 11.9])
```

VAM Algorithm

Main VAM function

- Calculate opportunity costs
- Find the highest opportunity cost and allocate to that
- Iterate while there’s remaining capacity and unfulfilled demand

```
############################################ VAM Algorithm ############################################
def vam(cost_matrix, supply, demand):
    allocation = np.zeros_like(cost_matrix)
    while supply.sum() > 0 and demand.sum() > 0:
        penalties = []
        for i in range(len(cost_matrix)):
            if supply[i] > 0:
                sorted_row = sorted([x for x in cost_matrix[i] if x < 1e9])
                if len(sorted_row) > 1:
                    penalties.append((sorted_row[1] - sorted_row[0], (i, -1)))
        for j in range(len(cost_matrix[0])):
            if demand[j] > 0:
                sorted_col = sorted([cost_matrix[i][j] for i in range(len(cost_matrix)) if cost_matrix[i][j] < 1e9])
                if len(sorted_col) > 1:
                    penalties.append((sorted_col[1] - sorted_col[0], (-1, j)))
        if not penalties:
            break
        max_penalty = max(penalties, key=lambda x: x[0])
        if max_penalty[1][0] != -1:
            row = max_penalty[1][0]
            col = np.argmin(cost_matrix[row])
        else:
            col = max_penalty[1][1]
            row = np.argmin(cost_matrix[:, col])
        min_supply_demand = min(supply[row], demand[col])
        allocation[row][col] = min_supply_demand
        supply[row] -= min_supply_demand
        demand[col] -= min_supply_demand
        if supply[row] == 0:
            cost_matrix[row] = 1e9
        if demand[col] == 0:
            cost_matrix[:, col] = 1e9
    return allocation
```

Supplementary function

- When only a few cells left – opportunity cost can no longer be calculated by the main function
- Fulfill demand with cheapest source

```
def fulfill_unmet_demand(allocation, supply, demand, cost_matrix):
    # Create a list to store potential allocations along with their costs
    potential_allocations = []

    # Identify cells with unmet demand and available supply
    for j in range(len(demand)):
        for i in range(len(supply)):
            if demand[j] > 0 and supply[i] > 0 and cost_matrix[i][j] < 1e9:
                potential_allocations.append((cost_matrix[i][j], i, j))

    # Sort potential allocations based on cost
    potential_allocations.sort(key=lambda x: x[0])

    # Allocate supply to these cells based on sorted order
    for cost, i, j in potential_allocations:
        if demand[j] > 0 and supply[i] > 0:
            amt = min(demand[j], supply[i])
            allocation[i][j] += amt
            demand[j] -= amt
            supply[i] -= amt

    return allocation
```

Existing Plan

The plan manually created by the team at AE

Demand / Supply	Mexico	Canada	Chile	Frankfurt	Austin	Japan
Mexico	3.0	-	-	-	12.4	1.8
Canada	-	2.6	-	-	-	-
Chile	-	-	4.1	-	-	-
Frankfurt	-	-	11.9	20	-	6.1
Austin	-	-	-	-	14.0	-
Japan	-	-	-	-	-	4.0

Total Cost: $78.445 million

Now, the team wonders; can a heuristic do a better job? We suggest using Vogel’s Approximation Method to attempt to make a better solution.

Scenario 1: 100% Capacity

Applying the VAM algorithm

Demand / Supply	Mexico	Canada	Chile	Frankfurt	Austin	Japan
Mexico	3	-	-	-	3.2	-
Canada	-	2.6	-	-	1.1	-
Chile	-	-	4.5	-	-	-
Frankfurt	-	-	11.5	20	3.6	11.9
Austin	-	-	-	-	18.5	-
Japan	-	-	-	-	-	-

Total Cost with 100% capacity: $74.09 million

Cost difference from the existing plan: - $4.355 million

Percentage Difference: - 5.55%

If we could run all of our factories at 100% capacity, our algorithm has created a good, initial logistics plan, which would **decrease** our total cost by 5.55% overall.

Scenario 2: 85% Capacity

Applying the VAM algorithm

Demand / Supply	Mexico	Canada	Chile	Frankfurt	Austin	Japan
Mexico	3	-	4.125	-	5.88	-
Canada	-	2.6	-	-	0.545	-
Chile	-	-	3.825	-	-	-
Frankfurt	-	-	8.05	20	-	11.9
Austin	-	-	-	-	15.725	-
Japan	-	-	-	-	4.25	-

Total Cost with 85% capacity: $78.99 million

Cost difference from the existing plan: + 0.545 million

Percentage Difference: + 0.69%

In the more realistic scenario in which our factory capacity is at 85%, our VAM algorithm provides us with an initial solution with a total cost of $78.99 million, a 0.69% **increase** in our total cost when compared to the existing plan made at AE.

Scenario 3: Chileave & 90% Capacity

Applying the VAM algorithm

Demand / Supply	Mexico	Canada	Chile	Frankfurt	Austin	Japan
Mexico	3	-	5.6	-	4.52	-
Canada	-	2.6	-	-	0.73	-
~~**Chile**~~	~~-~~	~~-~~	~~-~~	~~-~~	~~-~~	~~-~~
Frankfurt	-	-	10.4	20	-	11.9
Austin	-	-	-	-	16.65	-
Japan	-	-	-	-	4.5	-

Total Cost with Chileave & 90% capacity: $79.69 million

Cost difference from the existing plan: + 1.245 million

Percentage Difference: + 1.59%

If we were to close the Chile plant, which is the most difficult for AE to support, our VAM algorithm creates an initial solution with a total cost of $79.69 million; an **increase** in total cost by 1.59% compared to the existing plan at AE.

Thank You

EMORY
GOIZUETA BUSINESS SCHOOL