Transportation Problem Optimization

M. Tuluyhan Sözen — Student ID: 2019400039
Ali Ayhan Günder — Student ID: 2021400219

December 18, 2024

1 Generating Problem Instances

We approached the challenge of creating testable transportation problems by designing a function to generate input data based on critical parameters:

- **Number of supply nodes**: Represents the count of supply points available.
- **Number of demand nodes**: Indicates the number of demand points included.
- **Maximum cost**: Defines the highest allowable transportation cost between any supply and demand point.
- **Maximum supply and demand**: Sets the upper limit for supply capacities and demand requirements.

1.1 Process Steps

1. Random integer values were assigned to supply capacities and demand requirements.
2. The total supply was adjusted to match the total demand, eliminating the need for dummy nodes.
3. Transportation costs were randomly generated within a specified range of 1 to the maximum cost.

This method ensures that each generated problem instance is valid and ready for optimization tests.

2 Solving with Optimization Tools

To efficiently solve the transportation problem, we utilized Python's `Pulp` library. The transportation problem was modeled as a linear program with the objective of minimizing the total transportation cost while meeting supply and demand constraints.

2.1 Variables

- **Cost of transporting one unit of goods**: From a supply node to a demand node.
- **Quantity of goods transported**: From a supply node to a demand node.

2.2 Objective

Minimize the total transportation cost:

$$\text{Minimize } Z = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}$$

where:

- c_{ij} = Cost of transporting one unit from supply node i to demand node j.
- x_{ij} = Quantity transported from supply node i to demand node j.

2.3 Constraints

- **Supply Constraints**: The total amount transported from each supply point does not exceed its capacity.

$$\sum_{j=1}^{n} x_{ij} \leq \text{Supply}_i \quad \forall i = 1, \ldots, m$$

- **Demand Constraints**: The total amount received at each demand point meets its requirement.

$$\sum_{i=1}^{m} x_{ij} = \text{Demand}_j \quad \forall j = 1, \ldots, n$$

- **Non-Negativity**: All transportation values must be non-negative.

$$x_{ij} \geq 0 \quad \forall i = 1, \ldots, m; \quad j = 1, \ldots, n$$

2.4 Implementation in Pulp

1. **Define Variables**: Each decision variable represents the quantity transported between supply and demand nodes. These variables are declared as continuous and non-negative.
2. **Set Up the Objective Function**: The total cost was calculated as the sum-product of the cost matrix and decision variables.
3. **Add Constraints**:
   - Supply constraints ensured that the total shipped from each supply node equaled its capacity.
   - Demand constraints ensured that the total received at each demand node equaled its requirement.
4. **Solve the Problem**: `Pulp`'s solver computed the optimal transportation plan that minimized costs while satisfying all constraints.

3 Implementing the Revised Simplex Algorithm

The Revised Simplex Algorithm iteratively identifies the optimal solution to linear programming problems by navigating through feasible solutions.

3.1 Steps in the Algorithm

1. **Initialization**:
   - Slack variables were introduced to convert inequality constraints into equalities.
   - The initial feasible solution used the slack variables as the starting basis.
2. **Compute Reduced Costs**:
   - Reduced costs were calculated to identify if the current solution could be improved.
   - If all reduced costs were zero or negative, the solution was deemed optimal.
3. **Pivoting**:
   - If the solution was not optimal, a variable was selected to enter the basis (incoming variable).
   - The smallest ratio rule determined which variable would leave the basis (outgoing variable).
4. **Update the Basis**:
   - The inverse of the basis matrix was updated to reflect the new basis.
5. **Repeat**:
   - These steps were repeated until the solution was optimal or the problem was unbounded.

3.2 Key Features

- `Numpy` was utilized for efficient matrix operations.
- The algorithm included checks for edge cases such as unbounded problems or degeneracy.

4 Testing and Results

The Revised Simplex Algorithm was evaluated by applying it to transportation problems of varying sizes and comparing its performance to `Pulp`'s solver.

4.1 Testing Setup

- **Problem Sizes**: $n = m = 3, 5, 7, 10$.
- **Maximum transportation cost and supply/demand values**: Fixed at 20 and 100, respectively.

4.2 Observations

4.2.1 Solution Accuracy

Both the Revised Simplex Algorithm and `Pulp`'s solver provided the same optimal transportation plans and costs, verifying the accuracy of the custom implementation.

4.2.2 Integer Solutions

All results were integers, consistent with the inherent structure of transportation problems.

4.2.3 Performance

- `Pulp`'s solver significantly outperformed the Revised Simplex Algorithm on larger problem sizes.
- The runtime of the Revised Simplex Algorithm increased with problem size, aligning with its theoretical complexity of $\mathcal{O}(m^2 n)$.