Design and Implementation of a Transportation Problem using Linear Programming

A Dissertation
Submitted for the partial fulfilment of the
requirements for the award of the degree
of
Master of Technology
in
Computer Science and Engineering

Submitted By
Govind Jaiswal
M.Tech CSE
Enrolment No. 16535013

Department of Computer Science and Engineering,
Indian Institute of Technology, Roorkee,
Roorkee- 247667, India.
May, 2018

Declaration

I declare that the work presented in this dissertation with title **"Design and Implementation of a Transportation Problem using Linear Programming"** towards fulfillment of the requirement for the award of the degree of **Master of Technology** in **Computer Science & Engineering** submitted in the **Department of Computer Science & Engineering, Indian Institute of Technology Roorkee, India** is an authentic record of my own work carried out during the period of **May 2017 to May 2018** under the supervision of **Dr. Rajdeep Niyogi**, Associate Professor, Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, Roorkee, India. The content of this dissertation has not been submitted by me for the award of any other degree of this or any other institute.

Date: ....
Place: ROORKEE

GOVIND JAISWAL
(16535013)
M.TECH (CSE)

Certificate

This is to certify that the statement made by the candidate is correct to the best of my Knowledge and belief.

Date: ....
Place: ....

Sign: ....
Dr. Rajdeep Niyogi
(Associate Professor)
Indian Institute of Technology
Roorkee

Acknowledgement

Dedicated to my family and friends, for standing by me through thick and thin, without whom i would not have gotten this far. I would like to express my sincere gratitude to my advisor **Dr. Rajdeep Niyogi** for the continuous support of my study and research, for his patience, motivation, enthusiasm and immense knowledge. His guidance helped me in all time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my study.

I am also grateful to the Department of Computer Science and Engineering, IIT Roorkee for providing valuable resources to aid my research.

GOVIND JAISWAL

Abstract

Linear Programming is a way of allocating resources in efficient manner so that cost in allocation can be minimized and profit can be maximized. Linear Programming is based on mathematical formulations .Transportation Problem is the special case of Linear Programming in which we have to transport the goods from one location to another location such that the needs of every arrival location are fulfilled. However this problem can also be converted to another problems such as job assignment problem. The transportation problem can be solved by using method such as vogel's approximation Northwest-Corner (NWC),Least-Cost (LC) .We also focus on the optimization of solution of transportation problem.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Transportation problem is very well known problem because it is been used widely.In this problem we need to transport goods from certain set of sources to the other set of destinations.There are certain constraints on supply and demand of the goods .We have to minimize the transportation cost such that demand is fulfilled.The solution of this problem consist of three parts.

1. Linear Programming formulation of the problem
2. Basic solution proposal
3. Optimization of the Basic Problem

This problem can be converted to another problem according to requirement.Initially this problem is developed by Hitchcock [1, 3].Transportation problem can be solved by several methods such as North west corner method .but such methods can be optimized.The main objective is to optimize such methods.Initially Linear Programming formulation of the problem is required.The Linear Programming Problem (LPP) representing the Transportation problem for m sources and n destinations are generally given as: The quality of an Initial Basic Feasible Solution of the Transportation Problem is measured by the computational efforts. These are the certain methods for finding the Initial Basic Feasible Solution for the Transportation Problems are North West Corner Method (NWCM), and Vogels Approximation Method (VAM),Least Cost Method (LCM). There is no such kind of unique method which can be claimed to be the best method for finding an

$$\text{Minimize:} \quad z = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} .$$

$$\text{Subject to:} \quad \sum_{j=1}^{n} x_{ij} \le S_i \ ; \quad i = 1, 2, \dots \dots , m .$$

$$\sum_{i=1}^{m} x_{ij} \ge d_j \ ; \quad j = 1, 2, \dots \dots , n .$$

$$x_{ij} \ge 0 \quad \text{for all} \quad i \quad \text{and} \quad j .$$

Initial Basic Feasible Solution. We should also focus on the procedure for finding an Initial Basic Feasible Solution along with finding the optimal solution. North West Corner Method started with allocating at the most North West cell.Since NWCM is based on position rather than transportation cost,this method usually gives a higher transportation cost than optimal cost.

Chapter 2

Related Work

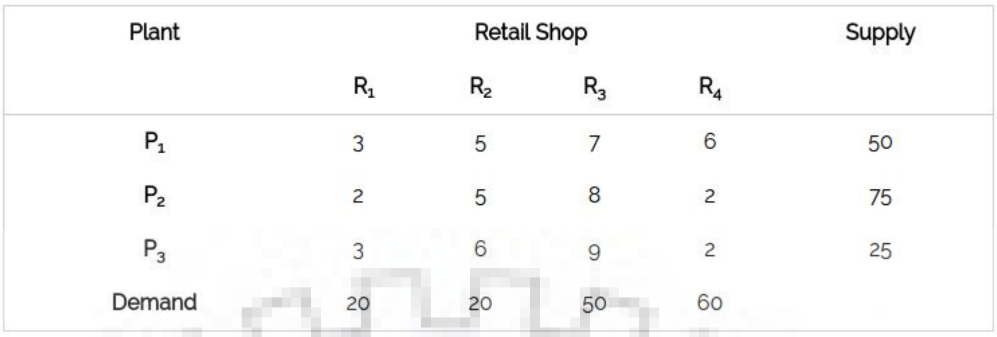
2.1 The North West Corner Rule

The North West corner rule is a method for computing a basic feasible solution of a transportation problem, where the basic variables are selected from the North West corner (i.e., top left corner).

The standard instructions for a transportation model are paraphrased below.

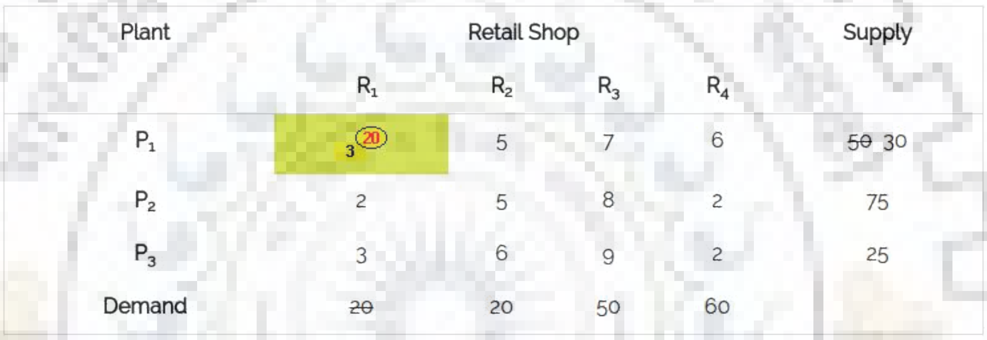
1. Select the upper left-hand corner cell of the transportation table and allocate as many units as possible equal to the minimum between available supply and demand, i.e., min(s1, d1).
2. Adjust the supply and demand numbers in the respective rows and columns.
3. If the demand for the first cell is satisfied, then move horizontally to the next cell in the second column.
4. If the supply for the first row is exhausted, then move down to the first cell in the second row.
5. If for any cell, supply equals demand, then the next allocation can be made in cell either in the next row or column.
6. Continue the process until all supply and demand values are exhausted.

The steps of NWCR are as follows: [2]



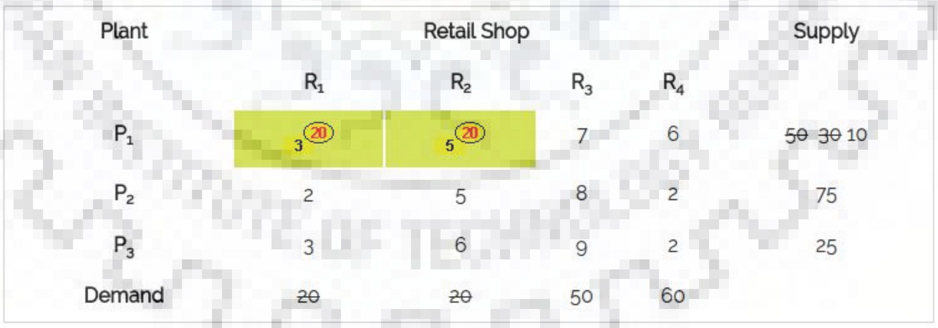
Plant	Retail Shop				Supply
	R_1	R_2	R_3	R_4	
P_1	3	5	7	6	50
P_2	2	5	8	2	75
P_3	3	6	9	2	25
Demand	20	20	50	60	

Figure 2.1: Step 1



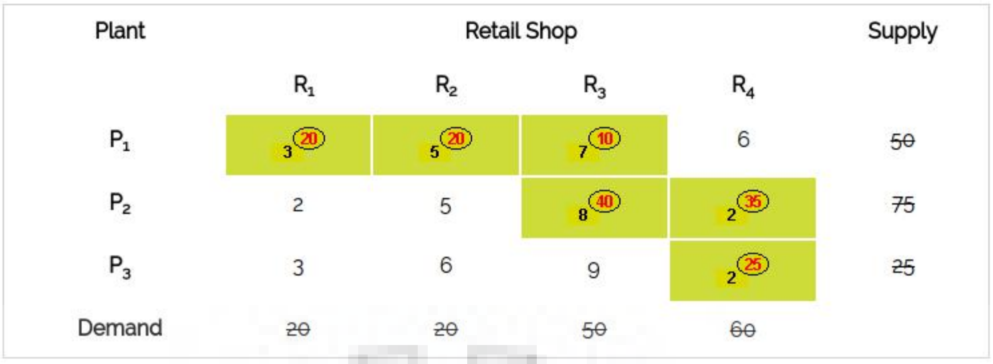
Plant	Retail Shop				Supply
	R_1	R_2	R_3	R_4	
P_1	3 20	5	7	6	~~50~~ 30
P_2	2	5	8	2	75
P_3	3	6	9	2	25
Demand	~~20~~	20	50	60	

Figure 2.2: Step 2



Plant	Retail Shop				Supply
	R_1	R_2	R_3	R_4	
P_1	3 20	5 20	7	6	~~50~~ ~~30~~ 10
P_2	2	5	8	2	75
P_3	3	6	9	2	25
Demand	~~20~~	~~20~~	50	60	

Figure 2.3: Step 3



Plant	Retail Shop				Supply
	R_1	R_2	R_3	R_4	
P_1	3 ⑳	5 ⑳	7 ⑩	6	~~50~~
P_2	2	5	8 ㊵	2 ㉟	~~75~~
P_3	3	6	9	2 ㉕	~~25~~
Demand	~~20~~	~~20~~	~~50~~	~~60~~	

FIGURE 2.4: Step 4

2.2 Stepping Stone

This problem can be completed by using Stepping Stone.This method can be performed if the following rules are considered. The following rules are used to allocate the units based on product delivery path.The rule according to (Render, 2007) is AIJ The number of occupied routers (squares) must always be equal to one less then the sum of the number of rows plus the number of columns. This means occupied shipping routes (squares) = number of rows + number of columns - 1 ". The rules in the application of Stepping Stone method is that The amount of the allocation of delivery routes must equal the number of rows plus the number of columns minus one , in other words we can say that the allocated amount of shipping routes = number of rows + number of columns - 1. To determine whether the allocation of each cell is optimal, it is necessary optimality testing by evaluating the cell are still empty or not (non basis variable) in order to find out if it ever done sending a unit into an empty cell is whether to raise or lower the total costs. This testing process is called Stepping Stone method. Stages of the testing stepping stone method are as

1. Choose one of cells which are empty for the test.
2. Starting from these cells that are still blank,draw a line opposite with clockwise direction and return back to an empty cell was way past of the cells that have been allocated to the units of the product based on shipping routes and its movement is done with the help of horizontal or vertical lines.

3. Start with the positive sign (+) from cells that are still vacant, and proceed using minus sign to the next cell, then use back plus sign to the next cell and continue back to the minus sign to the next cell, is intermittent until returning to original cell was still empty.

4. Calculate the progress by adding all the unit costs contained in each cell with plus sign and then reduce the cost of all units contained in each cell with a minus sign.

5. Repeat the above steps until all the improvement index is obtained in all cells that are still empty. If the results of all calculations improvement index is equal to or greater than zero, then the optimal solution has been achieved. If not, then it should be revised allocations of the cell that already contains the allocation of delivery routes from the source of supply, with the goal to minimize the total cost.

2.3 Vogel's Approximation

In The Vogels Approximation Method (VAM) is an iterative procedure for computing a basic feasible solution of a transportation problem. This method is better than other two methods i.e. North West Corner Rule (NWC) because the basic feasible solution obtained by this method is nearer to the optimal solution. The algorithm for Vogels Approximation Method (VAM) is given below:

1. Identify the cell having minimum and next to minimum transportation cost in each row and write the difference (Penalty) along the side of the table against the corresponding row.

2. Identify the cell having minimum and next to minimum transportation cost in each column and write the difference (Penalty) along the side of the table against the corresponding column. If minimum cost appear in two or more times in a row or column then select these same cost as a minimum and next to minimum cost and penalty will be zero.

3. **a.** Identify the row and column with the largest penalty, breaking ties arbitrarily. Allocate as much as possible to the variable with the least cost in the selected row or column. Adjust the supply and demand and cross out the

satisfied row or column. If a row and column are satisfies simultaneously, only one of them is crossed out and remaining row or column is assigned a zero supply or demand.

b. If two or more penalty costs have same largest magnitude, then select any one of them (or select most top row or extreme left column).

4. **a.** If exactly one row or one column with zero supply or demand remains uncrossed out, Stop.

   b. If only one row or column with positive supply ordemand remains uncrossed out, determine the basic variables in the row or column by the Least-Cost Method.

   c. If all uncrossed out rows or column have (remaining) zero supply or demand, determined the zero basic variables by the Least-Cost Method. Stop.

   d. Otherwise, go to Step-1.

Chapter 3

Problem Formulation

Suppose a company has m warehouses and n retail outlets. A single product is to be shipped from the warehouses to the outlets. Each warehouse has a given level of supply, and each outlet has a given level of demand. We are also given the transportation costs between every pair of warehouse and outlet, and these costs are assumed to be linear.More explicitly, the assumptions are : [4]

1. The total supply of the product from warehouse i is a i , where i = 1, 2, . . ., m.

2. The total demand for the product at outlet j is b_j , where j = 1, 2, . . ., n. 3. The cost of sending one unit of the product from warehouse i to outlet j is equal to c_{ij} , where i = 1, 2, . . ., m and j = 1, 2, . . ., n.

The total cost of a shipment is linear in the size of the shipment. The problem of interest is to determine an optimal transportation scheme between the warehouses and the outlets, subject to the specified supply and demand constraints. Graphically, a transportation problem is often visualized as a network with m source nodes, n sink nodes, and a set of m * n "directed arcs." We now proceed with a linear-programming formulation of this problem.

3.1 The Decision Variables

A transportation scheme is a complete specification of how many units of the product should be shipped from each warehouse to each outlet. Therefore, the decision variables are: x_{ij} = the size of the shipment from warehouse i to outlet j, where i = 1, 2, . . ., m and j = 1, 2, . . ., n. This is a set of m * n variables.

3.2 The Objective Function

Consider the shipment from warehouse i to outlet j. For any i and any j, the transportation cost per unit is c ij ; and the size of the shipment is x ij . Since we assume that the cost function is linear, the total cost of this shipment is given by $c_{ij}x_{ij}$. Summing over all i and all j now yields the overall transportation cost for all warehouse-outlet combinations. That is, our objective function is: [4]

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

3.3 The Constraints

Consider warehouse i. The total outgoing shipment from this warehouse is the sum x i1 + x i2 + . . . + x in . In summation notation, this is written as n_j=1 x_{ij} . Since the total supply from warehouse i is a i , the total outgoing shipment cannot exceed a i . That is, we must require [4] This results in a set of m + n functional constraints. Of course, as physical shipments, the x_{ij} s should be nonnegative.

$$\sum_{j=1}^{n} x_{ij} \le a_i \,, \quad \text{for } i = 1,\, 2,\, \ldots,\, m.$$

3.4 LP Formulation

In summary, we have arrived at the following formulation: This is a linear program with m * n decision variables, m + n functional constraints, and m * n non negativity constraints. [4]

$$
\begin{array}{ll}
\text{Minimize} & \displaystyle\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \\
\text{Subject to:} & \\
& \displaystyle\sum_{j=1}^{n} x_{ij} \quad \leq \quad a_i \qquad \text{for } i = 1, 2, \ldots, m \\
& \displaystyle\sum_{i=1}^{m} x_{ij} \quad \geq \quad b_j \qquad \text{for } j = 1, 2, \ldots, n \\
& x_{ij} \geq 0 \quad \text{for } i = 1, 2, \ldots, m \text{ and } j = 1, 2, \ldots, n.
\end{array}
$$

3.5 A Numerical Example

In an actual instance of the transportation problem, we need to specify m and n, and replace the a_i s, the b_js, and the c_{ij} s with explicit numerical values. As a simple example, suppose we are given: m = 3 and n = 2; a 1 = 45, a 2 = 60, and a 3 = 35; b 1 = 50 and b 2 = 60; and finally, c 11 = 3, c 12 = 2, c 21 = 1, c 22 = 5, c 31 = 5, and c 32 = 4. Then, substitution of these values into the above formulation leads to the following explicit problem: [47] For an example of an explicit transportation scheme, let x_{11} = 20, x_{12} = 20, x_{21} = 20, x_{22} = 20, x_{31} = 10, and x_{32} = 20. It is easily seen that this proposed solution satisfies all of the constraints, and hence it is feasible. In words, the solution calls for shipping 20 units from warehouse 1 to outlet 1, 20 units from warehouse 1 to outlet 2, 20 units from warehouse 2 to outlet 1, . . ., and finally 20 units from warehouse 3 to outlet 2. The total transportation cost associated with this transportation scheme can be computed as: 3 * 20 + 2 * 20 + 1 * 20 + 5 * 20 + 5 * 10 + 4 * 20 = 350. [4]

$$
\begin{array}{llllllllll}
\text{Minimize} & 3x_{11} & +2x_{12} & +x_{21} & +5x_{22} & +5x_{31} & +4x_{32} & & & \\
\text{Subject to:} & & & & & & & & & \\
& x_{11} & +x_{12} & & & & & \leq & 45 & (1) \\
& & & x_{21} & +x_{22} & & & \leq & 60 & (2) \\
& & & & & x_{31} & +x_{32} & \leq & 35 & (3) \\
& x_{11} & & +x_{21} & & +x_{31} & & \geq & 50 & (4) \\
& & x_{12} & & +x_{22} & & +x_{32} & \geq & 60 & (5) \\
& \multicolumn{9}{l}{x_{ij} \geq 0 \quad \text{for } i = 1, 2, 3 \text{ and } j = 1, 2.}
\end{array}
$$

3.6 An Equivalent Formulation

To derive an optimal transportation scheme for the above example, we can of course apply the standard Simplex algorithm. This means that we will first introduce 3 slack variables for constraints (1), (2), and

(3) and 2 surplus variables for constraints (4) and (5), to convert these inequalities into equalities. On top of these, at the onset of Phase I, we also need to introduce 2 more artificial variables to serve as starting basic variables for constraints (4) and (5). We would then go through Phase I and Phase II to arrive at an optimal solution. This routine is standard fare, but the introduction of so many new variables seems tedious. Can we do better? It turns out that we can. The first observation is that for a given problem to have any feasible solution, the total supply must not be less than the total demand. In this numerical example, we have a total supply of 140 and a total demand of 110. Hence, feasible solutions exist; and indeed, we constructed one with ease. The second observation is that if the total supply happens to be equal to the total demand, then any feasible solution must satisfy all of the inequality constraints as equalities. (For example, this would be the case if a 1 , a 2 , and a 3 had been 40, 40, and 30, respectively.) As a consequence, whenever the given total supply and total demand are the same, we can replace all (functional) inequality constraints by equality constraints. Our third, and final, observation is that after such replacements, there is no longer any need for introducing slack or surplus variables. Certainly, we cannot expect every problem to come with identical total supply and total

demand. However, notice that if the total supply is strictly greater than the total demand, then one can artificially create a dummy to absorb the difference between the two. Of course, in order to preserve the original cost structure, the transportation cost for units sent to the dummy sink should be set to zero. Thus, for this specific example, we can introduce a third outlet to serve as the dummy sink; and let b 3 = 30 and c 13 = c 23 = c 33 = 0. This yields the following new linear program: [4]

$$\begin{array}{llllllllllll}
\text{Minimize} & 3x_{11} & +2x_{12} & & +x_{21} & +5x_{22} & & +5x_{31} & +4x_{32} & & & \\
\text{Subject to:} & & & & & & & & & & & \\
& x_{11} & +x_{12} & +x_{13} & & & & & & & = & 45 \\
& & & & x_{21} & +x_{22} & +x_{23} & & & & = & 60 \\
& & & & & & & x_{31} & +x_{32} & +x_{33} & = & 35 \\
& x_{11} & & & +x_{21} & & & +x_{31} & & & = & 50 \\
& & x_{12} & & & +x_{22} & & & +x_{32} & & = & 60 \\
& & & x_{13} & & & +x_{23} & & & +x_{33} & = & 30
\end{array}$$

$x_{ij} \geq 0$ for $i = 1, 2, 3$ and $j = 1, 2, 3$.

It should be clear that by construction, this problem is equivalent to the original one. With the above discussion, we can now assume without loss of generality that every transportation problem comes with identical total supply and total demand. This gives rise to what is called the standard form of the transportation problem. Formally, under the assumption that sum of all a is is equal to sum of all b js.

Chapter 4

Implementation Details

Transportation problem can be solved by several method, namely Northwest-Corner (NWC), Vogels Approximation Method (VAM) and Assignment Method. The method is simply an early solution to the issue of transportation. To find the optimal solution, there are method that can be used, the Stepping-Stone Method and Simplex Method (MODI). Here we are comparing two method of solution of transportation problems, the NWCR method and the stepping-stone method.Here We have implemented the North West Corner Rule algorithm which is further been optimized by Stepping Stone algorihtm.The Implementation is done in Java .The Implementation of North West Corner Rule and Stepping Stone is as follows.

```
import java.io.File;
import java.util.*;
import static java.util.Arrays.stream;
import static java.util.stream.Collectors.toCollection;

public class TransportationProblem {

    private static int[] demand;
    private static int[] supply;
    private static double[][] costs;
    private static Shipment[][] matrix;

    private static class Shipment {
        final double costPerUnit;
        final int r, c;
        double quantity;

        public Shipment(double q, double cpu, int r, int c) {
            quantity = q;
            costPerUnit = cpu;
            this.r = r;
            this.c = c;
        }
    }

    static void init(String filename) throws Exception {

        try (Scanner sc = new Scanner(new File(filename))) {
            int numSources = sc.nextInt();
            int numDestinations = sc.nextInt();

            List<Integer> src = new ArrayList<>();
            List<Integer> dst = new ArrayList<>();

            for (int i = 0; i < numSources; i++)
```

```
                src.add(sc.nextInt());

            for (int i = 0; i < numDestinations; i++)
                dst.add(sc.nextInt());


            int totalSrc = src.stream().mapToInt(i -> i).sum();
            int totalDst = dst.stream().mapToInt(i -> i).sum();
            if (totalSrc > totalDst)
                dst.add(totalSrc - totalDst);
            else if (totalDst > totalSrc)
                src.add(totalDst - totalSrc);

            supply = src.stream().mapToInt(i -> i).toArray();
            demand = dst.stream().mapToInt(i -> i).toArray();

            costs = new double[supply.length][demand.length];
            matrix = new Shipment[supply.length][demand.length];

            for (int i = 0; i < numSources; i++)
                for (int j = 0; j < numDestinations; j++)
                    costs[i][j] = sc.nextDouble();
        }
    }

    static void northWestCornerRule() {

        for (int r = 0, northwest = 0; r < supply.length; r++)
            for (int c = northwest; c < demand.length; c++) {

                int quantity = Math.min(supply[r], demand[c]);
                if (quantity > 0) {
                    matrix[r][c] = new Shipment(quantity, costs[r][c], r, c);

                    supply[r] -= quantity;
                demand[c] -= quantity;

                if (supply[r] == 0) {
                    northwest = c;
                    break;
                }
            }
        }
}

static void steppingStone() {
    double maxReduction = 0;
    Shipment[] move = null;
    Shipment leaving = null;

    fixDegenerateCase();

    for (int r = 0; r < supply.length; r++) {
        for (int c = 0; c < demand.length; c++) {

            if (matrix[r][c] != null)
                continue;

            Shipment trial = new Shipment(0, costs[r][c], r, c);
            Shipment[] path = getClosedPath(trial);

            double reduction = 0;
            double lowestQuantity = Integer.MAX_VALUE;
            Shipment leavingCandidate = null;

            boolean plus = true;
            for (Shipment s : path) {
                if (plus) {
                    reduction += s.costPerUnit;
                } else {
                    if (s.quantity < lowestQuantity) {
                        leavingCandidate = s;
                        lowestQuantity = s.quantity;
                    }
                }
                plus = !plus;
            }
            if (reduction < maxReduction) {
                move = path;
                leaving = leavingCandidate;
                maxReduction = reduction;
            }
        }
    }

    if (move != null) {
        double q = leaving.quantity;
        boolean plus = true;
        for (Shipment s : move) {
            s.quantity += plus ? q : -q;
            matrix[s.r][s.c] = s.quantity == 0 ? null : s;
            plus = !plus;
        }
        steppingStone();
    }
}

static LinkedList<Shipment> matrixToList() {
    return stream(matrix)
            .flatMap(row -> stream(row))
            .filter(s -> s != null)
            .collect(toCollection(LinkedList::new));
}
```

```java
static Shipment[] getClosedPath(Shipment s) {
    LinkedList<Shipment> path = matrixToList();
    path.addFirst(s);


    while (path.removeIf(e -> {
        Shipment[] nbrs = getNeighbors(e, path);
        return nbrs[0] == null || nbrs[1] == null;
    }));


    Shipment[] stones = path.toArray(new Shipment[path.size()]);
    Shipment prev = s;
    for (int i = 0; i < stones.length; i++) {
        stones[i] = prev;
        prev = getNeighbors(prev, path)[i % 2];
    }
    return stones;
}

static Shipment[] getNeighbors(Shipment s, LinkedList<Shipment> lst) {
    Shipment[] nbrs = new Shipment[2];
    for (Shipment o : lst) {
        if (o != s) {
            if (o.r == s.r && nbrs[0] == null)
                nbrs[0] = o;
            else if (o.c == s.c && nbrs[1] == null)
                nbrs[1] = o;
            if (nbrs[0] != null && nbrs[1] != null)
                break;
        }
    }
    return nbrs;
}
```

```
static void fixDegenerateCase() {
    final double eps = Double.MIN_VALUE;

    if (supply.length + demand.length - 1 != matrixToList().size()) {

        for (int r = 0; r < supply.length; r++)
            for (int c = 0; c < demand.length; c++) {
                if (matrix[r][c] == null) {
                    Shipment dummy = new Shipment(eps, costs[r][c], r, c);
                    if (getClosedPath(dummy).length == 0) {
                        matrix[r][c] = dummy;
                        return;
                    }
                }
            }
    }
}

static void printResult(String filename) {
    System.out.printf("Optimal solution %s%n%n", filename);
    double totalCosts = 0;

    for (int r = 0; r < supply.length; r++) {
        for (int c = 0; c < demand.length; c++) {

            Shipment s = matrix[r][c];
            if (s != null && s.r == r && s.c == c) {
                System.out.printf(" %3s ", (int) s.quantity);
                totalCosts += (s.quantity * s.costPerUnit);
            } else
                System.out.printf("  -  ");
        }
        System.out.println();
    }
    System.out.printf("%nTotal costs: %s%n%n", totalCosts);
```

```
                    }
                }
            }
        }

    static void printResult(String filename) {
        System.out.printf("Optimal solution %s%n%n", filename);
        double totalCosts = 0;

        for (int r = 0; r < supply.length; r++) {
            for (int c = 0; c < demand.length; c++) {

                Shipment s = matrix[r][c];
                if (s != null && s.r == r && s.c == c) {
                    System.out.printf(" %3s ", (int) s.quantity);
                    totalCosts += (s.quantity * s.costPerUnit);
                } else
                    System.out.printf("  -  ");
            }
            System.out.println();
        }
        System.out.printf("%nTotal costs: %s%n%n", totalCosts);
    }

    public static void main(String[] args) throws Exception {

        for (String filename : new String[]{"input1.txt", "input2.txt",
            "input3.txt"}) {
            init(filename);
            northWestCornerRule();
            steppingStone();
            printResult(filename);
        }
    }
}
```

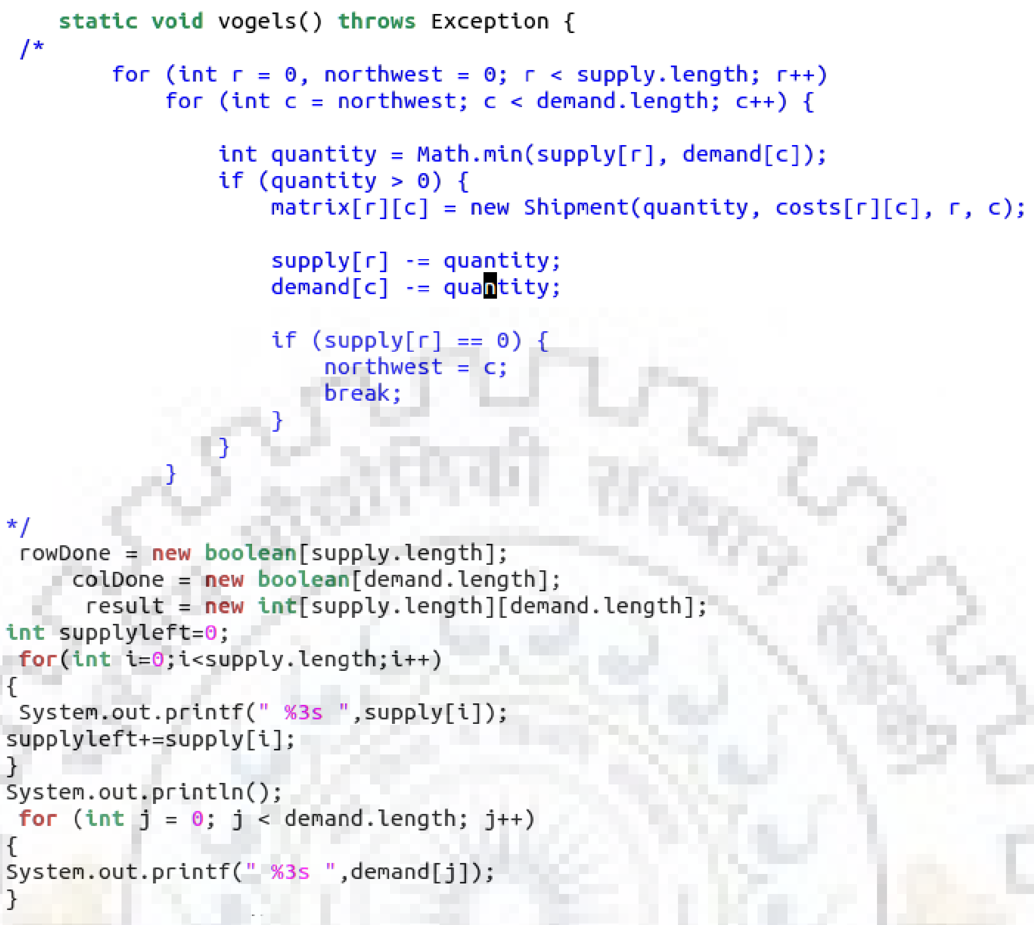
We have also implemented Vogel's approximation and also applied Stepping Stone method on it .The implementation of Stepping Stone on Vogel's approximation is as follow.

```
import java.io.File;
import java.util.*;
import static java.util.Arrays.stream;
import static java.util.stream.Collectors.toCollection;
import java.util.concurrent.*;
public class demo {

    private static int[] demand;
    private static int[] supply;
    private static double[][] costs;
    private static Shipment[][] matrix;
    public static  int[][] result;
static boolean[] rowDone;
static boolean[] colDone;
 static ExecutorService es = Executors.newFixedThreadPool(2);
//static boolean[] rowDone = new boolean[nRows];
  //  static boolean[] colDone = new boolean[nCols];
    //static int[][] result = new int[nRows][nCols];

    private static class Shipment {
        final double costPerUnit;
        final int r, c;
        double quantity;

        public Shipment(double q, double cpu, int r, int c) {
            quantity = q;
            costPerUnit = cpu;
            this.r = r;
            this.c = c;
        }
    }
```

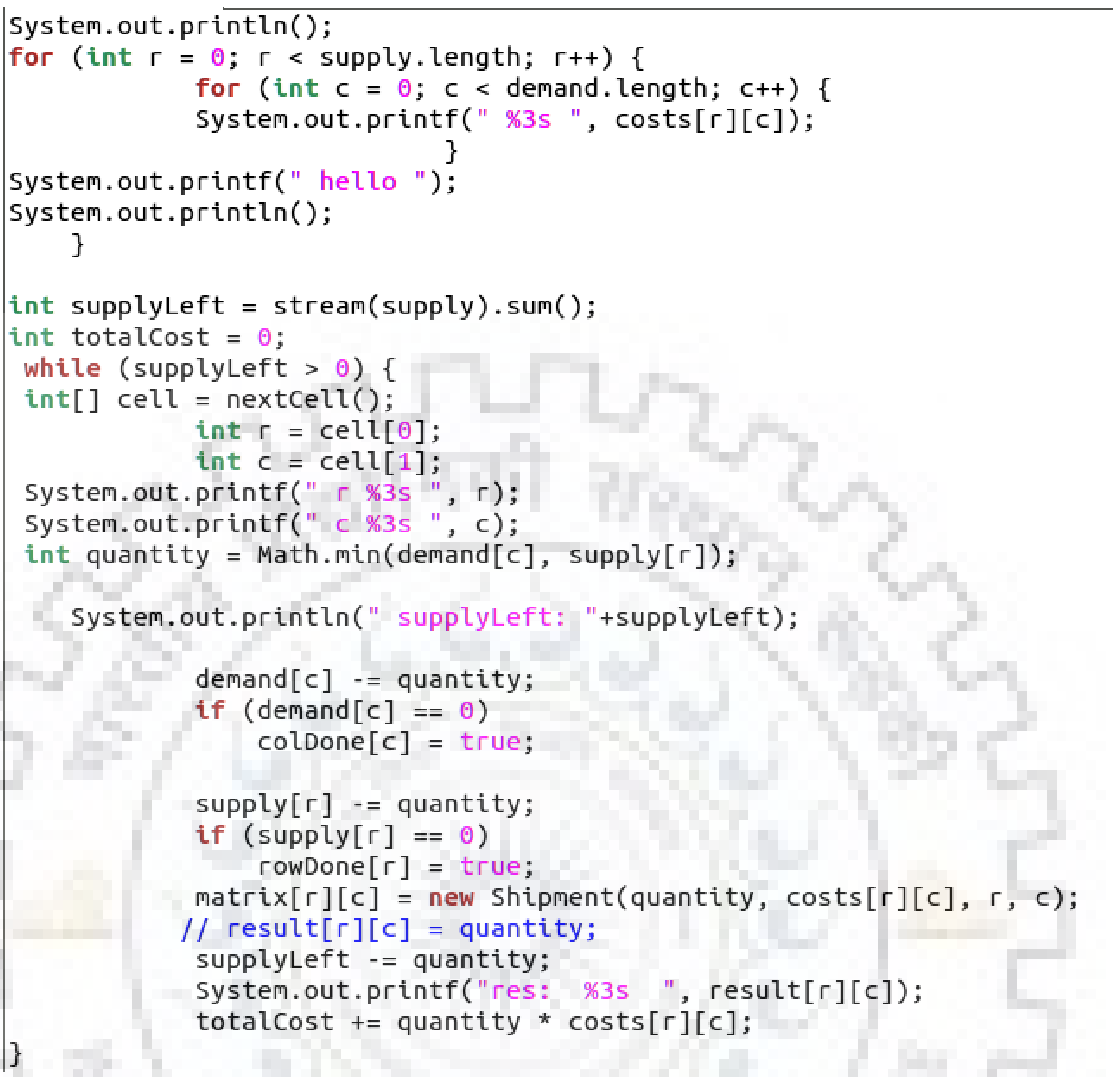
```
import java.io.File;
import java.util.*;
import static java.util.Arrays.stream;
import static java.util.stream.Collectors.toCollection;
import java.util.concurrent.*;
public class demo {

    private static int[] demand;
    private static int[] supply;
    private static double[][] costs;
    private static Shipment[][] matrix;
    public static  int[][] result;
static boolean[] rowDone;
static boolean[] colDone;
 static ExecutorService es = Executors.newFixedThreadPool(2);
//static boolean[] rowDone = new boolean[nRows];
  //  static boolean[] colDone = new boolean[nCols];
    //static int[][] result = new int[nRows][nCols];

    private static class Shipment {
        final double costPerUnit;
        final int r, c;
        double quantity;

        public Shipment(double q, double cpu, int r, int c) {
            quantity = q;
            costPerUnit = cpu;
            this.r = r;
            this.c = c;
        }
    }
```

```
import java.io.File;
import java.util.*;
import static java.util.Arrays.stream;
import static java.util.stream.Collectors.toCollection;
import java.util.concurrent.*;
public class demo {

    private static int[] demand;
    private static int[] supply;
    private static double[][] costs;
    private static Shipment[][] matrix;
    public static  int[][] result;
static boolean[] rowDone;
static boolean[] colDone;
 static ExecutorService es = Executors.newFixedThreadPool(2);
//static boolean[] rowDone = new boolean[nRows];
  //  static boolean[] colDone = new boolean[nCols];
    //static int[][] result = new int[nRows][nCols];

    private static class Shipment {
        final double costPerUnit;
        final int r, c;
        double quantity;

        public Shipment(double q, double cpu, int r, int c) {
            quantity = q;
            costPerUnit = cpu;
            this.r = r;
            this.c = c;
        }
    }
```

```
static void init(String filename) throws Exception {

    try (Scanner sc = new Scanner(new File(filename))) {
        int numSources = sc.nextInt();
        int numDestinations = sc.nextInt();

        List<Integer> src = new ArrayList<>();
        List<Integer> dst = new ArrayList<>();

        for (int i = 0; i < numSources; i++)
            src.add(sc.nextInt());

        for (int i = 0; i < numDestinations; i++)
            dst.add(sc.nextInt());

        // fix imbalance
        int totalSrc = src.stream().mapToInt(i -> i).sum();
        int totalDst = dst.stream().mapToInt(i -> i).sum();
        if (totalSrc > totalDst)
            dst.add(totalSrc - totalDst);
        else if (totalDst > totalSrc)
            src.add(totalDst - totalSrc);

        supply = src.stream().mapToInt(i -> i).toArray();
        demand = dst.stream().mapToInt(i -> i).toArray();

        costs = new double[supply.length][demand.length];
        matrix = new Shipment[supply.length][demand.length];

        for (int i = 0; i < numSources; i++)
            for (int j = 0; j < numDestinations; j++)
                costs[i][j] = sc.nextDouble();
    }
}
```



```
    static void vogels() throws Exception {
 /*
        for (int r = 0, northwest = 0; r < supply.length; r++)
            for (int c = northwest; c < demand.length; c++) {

                int quantity = Math.min(supply[r], demand[c]);
                if (quantity > 0) {
                    matrix[r][c] = new Shipment(quantity, costs[r][c], r, c);

                    supply[r] -= quantity;
                    demand[c] -= quantity;

                    if (supply[r] == 0) {
                        northwest = c;
                        break;
                    }
                }
            }

*/
 rowDone = new boolean[supply.length];
     colDone = new boolean[demand.length];
      result = new int[supply.length][demand.length];
int supplyleft=0;
 for(int i=0;i<supply.length;i++)
{
 System.out.printf(" %3s ",supply[i]);
supplyleft+=supply[i];
}
System.out.println();
 for (int j = 0; j < demand.length; j++)
{
System.out.printf(" %3s ",demand[j]);
}
```



```
System.out.println();
for (int r = 0; r < supply.length; r++) {
            for (int c = 0; c < demand.length; c++) {
            System.out.printf(" %3s ", costs[r][c]);
                            }
System.out.printf(" hello ");
System.out.println();
    }

int supplyLeft = stream(supply).sum();
int totalCost = 0;
 while (supplyLeft > 0) {
 int[] cell = nextCell();
            int r = cell[0];
            int c = cell[1];
 System.out.printf(" r %3s ", r);
 System.out.printf(" c %3s ", c);
 int quantity = Math.min(demand[c], supply[r]);

    System.out.println(" supplyLeft: "+supplyLeft);

            demand[c] -= quantity;
            if (demand[c] == 0)
                colDone[c] = true;

            supply[r] -= quantity;
            if (supply[r] == 0)
                rowDone[r] = true;
            matrix[r][c] = new Shipment(quantity, costs[r][c], r, c);
           // result[r][c] = quantity;
            supplyLeft -= quantity;
            System.out.printf("res:  %3s  ", result[r][c]);
            totalCost += quantity * costs[r][c];
}
```

```
  System.out.println(" tc: "+totalCost);
 steppingStone();
}
static int[] nextCell() throws Exception {
       Future<int[]> f1 = es.submit(() -> maxPenalty(supply.length,demand.length , true));
       Future<int[]> f2 = es.submit(() -> maxPenalty(demand.length,supply.length, false));

       int[] res1 = f1.get();
       int[] res2 = f2.get();

       if (res1[3] == res2[3])
           return res1[2] < res2[2] ? res1 : res2;

       return (res1[3] > res2[3]) ? res2 : res1;
   }
static int[] maxPenalty(int len1, int len2, boolean isRow) {
       int md = Integer.MIN_VALUE;
       int pc = -1, pm = -1, mc = -1;
       for (int i = 0; i < len1; i++) {
           if (isRow ? rowDone[i] : colDone[i])
               continue;
           int[] res = diff(i, len2, isRow);
           if (res[0] > md) {
               md = res[0];  // max diff
               pm = i;       // pos of max diff
               mc = res[1];  // min cost
               pc = res[2];  // pos of min cost
           }
       }
       return isRow ? new int[]{pm, pc, mc, md} : new int[]{pc, pm, mc, md};
   }
static int[] diff(int j, int len, boolean isRow) {
       int min1 = Integer.MAX_VALUE, min2 = Integer.MAX_VALUE;
       int minP = -1;
```

```
        for (int i = 0; i < len; i++) {
            if (isRow ? colDone[i] : rowDone[i])
                continue;
            int c = (int)(isRow ? costs[j][i] : costs[i][j]);
            if (c < min1) {
                min2 = min1;
                min1 = c;
                minP = i;
            } else if (c < min2)
                min2 = c;
        }
        return new int[]{min2 - min1, min1, minP};
    }
    static void steppingStone()throws Exception {
        double maxReduction = 0;
        Shipment[] move = null;
        Shipment leaving = null;

        fixDegenerateCase();

        for (int r = 0; r < supply.length; r++) {
            for (int c = 0; c < demand.length; c++) {
              // if(result[r][c])
              // System.out.printf("%3s yes", matrix[r][c].);
                if (matrix[r][c] != null)
                    continue;

                Shipment trial = new Shipment(0, costs[r][c], r, c);
                Shipment[] path = getClosedPath(trial);

                double reduction = 0;
                double lowestQuantity = Integer.MAX_VALUE;
                Shipment leavingCandidate = null;
                boolean plus = true;
                for (Shipment s : path) {
                    if (plus) {
                        reduction += s.costPerUnit;
                    } else {
                        reduction -= s.costPerUnit;
                        if (s.quantity < lowestQuantity) {
                            leavingCandidate = s;
                            lowestQuantity = s.quantity;
                        }
                    }
                    plus = !plus;
                }
                if (reduction < maxReduction) {
                    move = path;
                    leaving = leavingCandidate;
                    maxReduction = reduction;
                }
            }
        }

        if (move != null) {
            double q = leaving.quantity;
            boolean plus = true;
            for (Shipment s : move) {
                s.quantity += plus ? q : -q;
                matrix[s.r][s.c] = s.quantity == 0 ? null : s;
                plus = !plus;
            }
            steppingStone();
        }
    }
```

```java
static LinkedList<Shipment> matrixToList() {
    return stream(matrix)
            .flatMap(row -> stream(row))
            .filter(s -> s != null)
            .collect(toCollection(LinkedList::new));
}

static Shipment[] getClosedPath(Shipment s) {
    LinkedList<Shipment> path = matrixToList();
    path.addFirst(s);

    // remove (and keep removing) elements that do not have a
    // vertical AND horizontal neighbor
    while (path.removeIf(e -> {
        Shipment[] nbrs = getNeighbors(e, path);
        return nbrs[0] == null || nbrs[1] == null;
    }));

    // place the remaining elements in the correct plus-minus order
    Shipment[] stones = path.toArray(new Shipment[path.size()]);
    Shipment prev = s;
    for (int i = 0; i < stones.length; i++) {
        stones[i] = prev;
        prev = getNeighbors(prev, path)[i % 2];
    }
    return stones;
}
```

```
static Shipment[] getNeighbors(Shipment s, LinkedList<Shipment> lst) {
    Shipment[] nbrs = new Shipment[2];
    for (Shipment o : lst) {
        if (o != s) {
            if (o.r == s.r && nbrs[0] == null)
                nbrs[0] = o;
            else if (o.c == s.c && nbrs[1] == null)
                nbrs[1] = o;
            if (nbrs[0] != null && nbrs[1] != null)
                break;
        }
    }
    return nbrs;
}

static void fixDegenerateCase() {
    final double eps = Double.MIN_VALUE;

    if (supply.length + demand.length - 1 != matrixToList().size()) {

        for (int r = 0; r < supply.length; r++)
            for (int c = 0; c < demand.length; c++) {
                if (matrix[r][c] == null) {
                    Shipment dummy = new Shipment(eps, costs[r][c], r, c);
                    if (getClosedPath(dummy).length == 0) {
                        matrix[r][c] = dummy;
                        return;
                    }
                }
            }
    }
}
```

```
    static void printResult(String filename) {
        System.out.printf("Optimal solution %s%n%n", filename);
        double totalCosts = 0;

        for (int r = 0; r < supply.length; r++) {
            for (int c = 0; c < demand.length; c++) {

                Shipment s = matrix[r][c];
                if (s != null && s.r == r && s.c == c) {
                    System.out.printf(" %3s ", (int) s.quantity);
                    totalCosts += (s.quantity * s.costPerUnit);
                } else
                    System.out.printf("  -  ");
            }
            System.out.println();
        }
       System.out.printf("%nTotal costs: %s%n%n jai shri ram", totalCosts);
    }

    public static void main(String[] args) throws Exception {

        for (String filename : new String[]{"first.txt", "second.txt","input1.txt","input2.txt","input3.txt",
            "third.txt","input4.txt","input5.txt"}) {
            init(filename);
            vogels();
         //   steppingStone();
            printResult(filename);
        }
    }
}
```

4.1 Result Comparison

Example 1:

No of Sources	No of Destination
2	3

Supply 1	Supply 2
2	3

Demand 1	Demand 2	Demand 3
20	30	10

Source/ Destination	d1	d2	d3
s1	3	5	7
s2	3	2	5

NWCL Solution

```
20 5 -

- 25 10

Total costs: 185.0
```

NWCL followed by stepping stone solution

```
20 - 5

- 30 5

Total costs: 180.0
```

Vogel's Approximation solution

```
20 - 5

- 30 5

Total costs: 180.0
```

Vogels approx and stepping stone

Example 1

```
20 - 5

- 30 5

Total costs: 180.0
```

Example 2:

No of Sources	No of Destination
3	3

Supply1	Supply2	Supply3
12	40	33

Demand1	Demand2	Demand3
20	30	10

Source/ Destination	d1	d2	d3
s1	3	5	7
s2	2	4	6
s3	9	1	8

NWCL Solution

12 - - -

8 30 2 -

- - 8 25

Total costs: 248.0

NWCL followed by Stepping Stone solution

- - - 12

20 - 10 10

- 30 - 3

Total costs: 130.0

Vogel's Approximation solution

- - - 12

20 - 7 13

- 30 3 -

Total costs: 136.0

Vogels approx and stepping stone
Example 2

- - - 12

20 - 10 10

- 30 - 3

Total costs: 130.0

Example 3:

No of Sources	No of Destination
10	12

Sup1	Sup2	Sup3	Sup4	Sup5	Sup6	Sup7	Sup8	Sup9	Sup10
25	35	23	24	22	34	32	23	45	56

Dm1	Dm2	Dm3	Dm4	Dm5	Dm6	Dm7	Dm8	Dm9	Dm10	Dem11	Dm12
45	32	12	14	24	13	34	23	12	23	24	63

Source/ Destination	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10	d11	d12
s1	5	2	2	1	4	3	5	3	1	3	4	6
s2	4	2	2	4	4	1	4	3	2	3	2	3
s3	6	32	2	1	24	13	34	23	12	23	24	63
s4	5	3	12	14	24	13	34	23	12	23	24	63
s5	4	2	12	14	24	13	34	23	12	23	24	63
s6	4	2	12	14	24	13	34	23	12	23	24	63
s7	4	2	12	14	24	13	34	23	12	23	24	63
s8	5	3	12	23	24	13	34	23	12	3	24	63
s9	5	32	12	14	24	13	34	23	1	23	24	63
s10	45	32	12	14	24	13	34	23	1	23	2	63

NWCL Solution

25 - - - - - - - - - - -

20 15 - - - - - - - - - -

- 17 6 - - - - - - - - -

- - 6 14 4 - - - - - - -

- - - - 20 2 - - - - - -

```
- - - - - 11 23 - - - - -

- - - - - - 11 21 - - - -

- - - - - - - 2 12 9 - -

- - - - - - - - - 14 24 7

- - - - - - - - - - - 56
```

Total costs: 8527.0

NWCL followed by stepping stone solution
Example 3

```
- - - - - - - - - - - 25

- - - - - - - - - - - 35

- - 9 14 - - - - - - - -

- - - - - - 12 12 - 0 - -

- 22 - - - - - - - - - -

24 10 - - - - - - - - - -

21 - - - - - - 11 - - - -

- - - - - - - - - 23 - -

- - 3 - 24 13 5 - - - - -

- - - - - - 17 - 12 - 24 3
```

Total costs: 3315.0

Vogel's Approximation solution

```
- - - 14 - - - 11 - - - -

- - - - - - - - - - - 35

- - 12 - - 11 - - - - - -

- 24 - - - - - - - - - -
```

```
12 8 - - - 2 - - - - - -

- - - - 24 - 10 - - - - -

- - - - - - 24 8 - - - -

- - - - - - - - - 23 - -

33 - - - - - - - 12 - - -

- - - - - - - 4 - - 24 28

Total costs: 4547.0
```

Vogels approx and stepping stone

Example 3

solution Example-3

```
- - - - - - - - - - - 25

- - - - - - - - - - - 35

- - 9 14 - - - - - - - -

- - - - 1 - 23 - - 0 - -

- 22 - - - - - - - - - -

24 10 - - - - - - - - - -

21 - - - - - 11 - - - - -

- - - - - - - - - 23 - -

- - 3 - 17 13 - - 12 - - -

- - - - 6 - - 23 - - 24 3

Total costs: 3315.0
```

Example 4

No of Sources	No of Destination
3	3

Supply1	**Supply2**	**Supply3**
12	40	33

Demand1	**Demand2**	**Demand3**
20	30	10

Source/ Destination	**d1**	**d2**	**d3**
s1	3	5	7
s2	2	4	6
s3	9	1	8

NWCL Solution

Example 4

```
12 - - -

8 30 2 -

- - 8 25
```

Total costs: 248.0

NWCL followed by Stepping Stone solution

```
- - - 12

20 - 10 10

- 30 - 3
```

Total costs: 130.0

Vogel's Approximation solution

```
- - - 12

20 - 7 13
```

- 30 3 -

Total costs: 136.0

Vogels approx and stepping stone

Example 4

- - - 12

20 - 10 10

- 30 - 3

Total costs: 130.0

Example 5

No of Sources	No of Destination
4	4

Supply1	Supply2	Supply3	Supply4
14	10	15	12

Demand1	Demand2	Demand3	Demand4
10	15	12	15

Source/ Destination	d1	d2	d3	d4
s1	10	30	25	15
s2	20	15	20	10
s3	10	30	20	20
s4	30	40	35	45

NWCL Solution

Example 5

10 4 - -

- 10 - -

```
- 1 12 2

- - - 12

- - - 1

Total costs: 1220.0
```

NWCL followed by stepping stone solution

```
- - - 14

- 9 - 1

10 - 5 -

- 5 7 -

- 1 - -

Total costs: 1000.0
```

Vogel's Approximation solution

```
- - - 14

- 9 - 1

10 - 5 -

- 5 7 -

- 1 - -

Total costs: 1000.0
```

Vogels approx and stepping stone
Example 5

```
- - - 14

- 9 - 1
```

10 - 5 -

- 5 7 -

- 1 - -

Total costs: 1000.0

Example 6

No of Sources	**No of Destination**
4	4

Supply1	**Supply2**	**Supply3**	**Supply4**
15	10	15	12

Demand1	**Demand2**	**Demand3**	**Demand4**
10	15	12	15

Source/ Destination	**d1**	**d2**	**d3**	**d4**
s1	10	30	25	15
s2	20	15	20	10
s3	10	30	20	20
s4	30	40	35	45

NWCL Solution
Example 6
10 5 - -

- 10 - -

- - 12 3

- - - 12

Total costs: 1240.0

NWCL followed by stepping stone solution

0 - - 15

- 10 - -

10 - 5 -

- 5 7 -

Total costs: 1020.0

Vogel's Approximation solution

- - - 15

- 10 - -

10 - 5 -

- 5 7 -

Total costs: 1020.0

Vogels approx and stepping stone
Example 6

0 - - 15

- 10 - -

10 - 5 -

- 5 7 -

Total costs: 1020.0 **Example 7**

No of Sources	**No of Destination**
8	10

Supply1	**Supply2**	**Supply3**	**Supply4**	**Supply5**	**Supply6**	**Supply7**	**Supply8**
25	35	34	21	23	24	12	23

Dem1	**Dem2**	**Dem3**	**Dem4**	**Dem5**	**Dem6**	**Dem7**	**Dem8**	**Dem9**	**Dem10**
20	30	10	5	1	23	34	2	12	60

Source/ Destination	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10
s1	3	5	7	3	4	2	4	2	1	4
s2	2	2	7	3	4	2	4	2	1	4
s3	2	3	3	6	4	3	4	2	1	4
s4	2	3	2	5	4	2	4	2	1	4
s5	4	3	1	4	4	2	4	2	1	4
s6	3	5	7	3	4	3	4	2	1	4
s7	2	2	7	2	4	2	4	2	1	4
s8	1	5	7	2	4	2	4	2	1	4

NWCL Solution

Example 7

20 5 - - - - - - - -

- 25 10 - - - - - - -

- - - 5 1 23 5 - - -

- - - - - - 21 - - -

- - - - - - 8 2 12 1

- - - - - - - - - 24

- - - - - - - - - 12

- - - - - - - - - 23

Total costs: 677.0

NWCL followed by Stepping Stone solution

- - - - 0 - - - 12 13

- 30 - - 1 2 - 2 - -

- - - - - 21 13 - - -

- - - - - - 21 - - -

- - 10 - - - - - - 13

- - - - - - - - - 24

```
- - - 5 - - - - - 7

20 - - - - - - - - 3
```

Total costs: 542.0

Vogel's Approximation solution

```
- - - - 1 23 1 - - -

- 30 - - - - 5 - - -

- - - - - - 28 2 4 -

- - - - - - - - 8 13

- - 10 - - - - - - 13

- - - - - - - - - 24

- - - 5 - - - - - 7

20 - - - - - - - - 3
```

Total costs: 542.0

Vogels approx and stepping stone
Example 7

```
- - - - 1 23 1 - - -

- 30 - - - - 5 - - -

- - - - - - 28 2 4 -

- - - - - - - - 8 13

- - 10 - - - - - - 13

- - - - - - - - - 24

- - - 5 - - - - - 7

20 - - - - - - - - 3
```

Total costs: 542.0

Example 8

No of Sources	No of Destination
4	5

Supply1	Supply2	Supply3	Supply4
100	80	70	90

Demand1	Demand2	Demand3	Demand4	Demand5
60	40	100	50	90

Source/ Destination	d1	d2	d3	d4	d5
s1	10	8	9	5	13
s2	7	9	8	10	4
s3	9	3	7	10	6
s4	11	4	8	3	9

NWCL Solution

Example 8

60 40 - - -

- - 80 - -

- - 20 50 -

- - - - 90

Total costs: 3010.0

NWCL followed by Stepping Stone solution

60 - 40 - -

- - - - 80

- 0 60 - 10

- 40 - 50 -

Total costs: 2070.0

Vogel's Approximation solution

- - 100 - -

60 - - - 20

- - - - 70

- 40 - 50 -

Total costs: 2130.0

Vogels approx and stepping stone
Example 8

60 - 40 - -

- - - - 80

- 0 60 - 10

- 40 - 50 -

Total costs: 2070.0

Comparison

	NWCL	**SS on NWCL**	**Vogel's approximation**	**SS on Vogel'sapp**
Ex-1	185	180	180	180
Ex-2	248	130	136	130
Ex-3	8527	3315	4547	3315
Ex-4	248	130	136	130
Ex-5	1220	1000	1000	1000
Ex-6	1240	1020	1020	1020
Ex-7	677	542	542	542
Ex-8	3010	2070	2130	2070

TABLE 4.1: Comparison of various solutions

	diff NWCL-optimal	**diff Vogel-optimal**	**diff Vogel-NWCL**
1	5	0	5
2	118	6	112
3	5212	1232	3980
4	118	6	112
5	220	0	220
6	220	0	220
7	125	0	135
8	940	60	880

TABLE 4.2: Differences of various solutions

Chapter 5

Conclusion

In todays highly competitive market, various organizations want to deliver products to the customers in a cost effective way, so that the market becomes competitive. To meet this challenge, transportation model provides a powerful framework to determine the best ways to deliver goods to the customer. In this article, we compared different methods for finding an initial basic feasible solution of transportation problems.We can see that vogel's approximation always give better initial basic feasible solution and vogel's approximation solution is near to optimal solution or is itself is the optimal solution. This will help to achieve the goal to those who want to maximize their profit by minimizing the transportation cost.

Chapter 6

References

[1] M. Ary and D. Syarifuddin. Comparison the transportation problem solution between northwest-corner method and steppingstone method with basis tree approach, 10 2011.

[2] M. I. A. Halawa, A. M. Maatuk, H. S. Idrees, and E. H. Ali. An optimal solution for transportation problem using computing modelling. In 2016 International Conference on Engineering MIS (ICEMIS), pages 15, Sept 2016.

[3] F. L. Hitchcock. The distribution of a product from several sources to numerous localities. Journal of Mathematics and Physics, 20(1-4):224230, 1941.

[4] J. Holladay. Some transportation problems and techniques for solving them. Naval Research Logistics Quarterly, 11(1):1542, 1964.

[5] A.Charnes,W.W.Cooper The stepping stone method of explaining linear programming calculations in transportation problems Manage. Sci.,1(1)(1954), pp.49-69

[6] U.K.Das,M.A.Babu,A.R.Khan,M.A.Helal,M.S.Uddin Logical development of vogel's approximation method (LD-VAM): an approach to find basic feasible solution of transportation problem Int. J. Sci. Technol. Res.,3(2)(2014), pp.42-48
[7] N.M.Deshmukh An Innovative method for solving transportation problem Int. J. Phys. Math. Sci.,2(3)(2012), pp.86-91

[8] S.K.GoyalImproving VAM for unbalanced transportation problems J. Oper. Res. Soc.,35(12)(1984), pp.1113-1114