CHAPTER 3

NEW METHOD FOR SOLVING TRANSPORTATION PROBLEMS

3.1 INTRODUCTION

The transportation problem constitutes an important part of logistics management. In addition, logistics problems without shipment of commodities may be formulated as transportation problems. For instance, the decision problem of minimizing dead kilometers given by Raghavendra et al (1987) can be formulated as a transportation problem illustrated by Vasudevan et al (1993), Sridharan (1991). The problem is important in urban transport undertakings, as dead kilometers mean additional losses. It is also possible to approximate certain additional linear programming problems by using a transportation formulated by Dhose et al (1996).

Various methods are available to solve the transportation problem to obtain an optimal solution. Typical/well-known transportation methods include the stepping stone method used by Charnes et al (1954) and the modified distribution method was given by Dantzig (1963), the modified stepping-stone method adopted by Shih (1987), the simplex-type algorithm formulated by Arsham et al (1989) and the dual-matrix method approached by Ji et al (2002). Glover et al (1974) presented a detailed computational comparison of basic solution algorithms for solving the transportation problems. Shafaat et al (1988) proposed a systematic approach for handling the situation of degeneracy encountered in the stepping stone method.
A detailed literature review on the basic solution methods was not presented. All the optimal solution algorithms for solving transportation problems need an initial basic feasible solution to obtain the optimal solution. There are various heuristic methods available to get an initial basic feasible solution, such as "North West Corner" rule, "Best Cell Method," "VAM — Vogel's Approximation Method" explained by Reinfeld et al (1958), Shimshak et al (1981), Goyal (1984), Ramakrishnan (1988) developed earlier versions of VAM. Further Kirca et al (1990) developed a heuristic, called TOM (Total Opportunity-cost Method), for obtaining an initial basic feasible solution for the transportation problem.

Gass (1990) detailed the practical issues for solving transportation problems and offered comments on various aspects of transportation problem methodologies along with discussions on the computational results, by the respective researchers. Recently, Sharma et al (2000) proposed a new heuristic approach for getting good initial solution for dual based approaches used to solve transportation problems.

All the researchers were mainly focused on obtaining initial basic feasible solution with various methods and not discussed about optimal solution. Even all the available methods to solve transportation problems need more iteration to get optimal solution. In this chapter, new method namely Zero Suffix Method is proposed for solving transportation problems. The proposed new method is found to have the following major advantages:

(i) It helps to get optimal solution directly with less iteration.

(ii) The degeneracy problem is not found to arise.

(iii) The new method is a systematic procedure, easy to apply and can be utilized for all types of transportation problems.
3.2 **ZERO SUFFIX METHOD**

The zero suffix method proceeds as follows.

**Step 1** : Construct the transportation table for the given TP and check the balanced condition. If not, convert it into balanced one.

**Step 2** : Subtract each row entries of the transportation table from the row minimum.

**Step 3** : Subtract each column entries of the resulting transportation table after using the Step 2 from the column minimum.

**Step 4** : In the reduced cost matrix there will be at least one zero in each row and column, then find the suffix value of all the zeros in the reduced cost matrix by following simplification, the suffix value is denoted by S,

Therefore $S = \frac{\text{Add the costs of nearest adjacent sides of zero}}{\text{No. of costs added}}$

**Step 5** : Choose the maximum of S, if it has one maximum value then supply to that demand corresponding to the cell. If it has more equal values then select $\{a_i, b_j\}$ and supply to that demand maximum possible.

**Step 6** : After the above step, the exhausted demands (column) or supplies (row) are to be trimmed. If $a_i = b_j$, cross out either $i^{th}$ row or $j^{th}$ column but not both. The resultant matrix must possess at least one zero in each row and each column, else repeat step 2 and step 3.

**Step 7** : Repeat Step 4 to Step 6 until the optimal cost is obtained.
3.3 NUMERICAL EXAMPLE

Consider the following cost minimizing transportation problems.

Example

<table>
<thead>
<tr>
<th></th>
<th>D₁</th>
<th>D₂</th>
<th>D₃</th>
<th>D₄</th>
<th>Supply</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₁</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>S₂</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>S₃</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>

Demand | 4  | 6  | 8  | 6  |

By applying step 1 to step 3, the following matrix is arrived

<table>
<thead>
<tr>
<th></th>
<th>D₁</th>
<th>D₂</th>
<th>D₃</th>
<th>D₄</th>
<th>Supply</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₁</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>S₂</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>S₃</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>

Demand | 4  | 6  | 8  | 6  |

By applying the step 4,

<table>
<thead>
<tr>
<th></th>
<th>D₁</th>
<th>D₂</th>
<th>D₃</th>
<th>D₄</th>
<th>Supply</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₁</td>
<td>1</td>
<td>0₁₃</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>S₂</td>
<td>4</td>
<td>1</td>
<td>0₀.₇₅</td>
<td>0₁₆</td>
<td>8</td>
</tr>
<tr>
<td>S₃</td>
<td>0₂</td>
<td>0₀₃</td>
<td>0₀₃</td>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>

Demand | 4  | 6  | 8  | 4  |

The suffix value of zeros were considered, among the suffix value 2 is maximum. Therefore, the demand first is supplied to (S₃, D₁), after supplying the trimmed matrix.
Consider suffix value of zeros, among the suffix value 1.6 is maximum. Therefore, the demand first is supplied to (S₂, D₄), after supplying the trimmed matrix.

<table>
<thead>
<tr>
<th></th>
<th>D₂</th>
<th>D₃</th>
<th>D₄</th>
<th>Supply</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₁</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>S₂</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>S₃</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Demand</td>
<td>6</td>
<td>8</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>D₂</th>
<th>D₃</th>
<th>D₄</th>
<th>Supply</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₁</td>
<td>0₁₅</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>S₂</td>
<td>1</td>
<td>0₀₇₅</td>
<td>0₁₆</td>
<td>8</td>
</tr>
<tr>
<td>S₃</td>
<td>0₀₅</td>
<td>0₀₃</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Demand</td>
<td>6</td>
<td>8</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>D₂</th>
<th>D₃</th>
<th>Supply</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₁</td>
<td>0₁₅</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>S₂</td>
<td>1</td>
<td>0₁</td>
<td>2</td>
</tr>
<tr>
<td>S₃</td>
<td>0₀₅</td>
<td>0₀₀</td>
<td>6</td>
</tr>
<tr>
<td>Demand</td>
<td>6</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>
In above matrix, among the suffix value 1.5 is maximum. Therefore, the demand first is supplied to \((S_1, D_2)\). Here, supply value and demand value are equal, therefore do not cancel both, cancel either supply or demand, keep the remaining one as zero, after supplying the resultant matrix

\[
\begin{array}{ccc}
S_2 & D_2 & D_3 \\
\hline
S_2 & 1 & 0.5 \\
S_3 & 0.5 & 0 \\
Demand & 0 & 8 \\
\end{array}
\]

By applying the similar procedure, the proposed method gives following allocation \(X_{12} = 6\), \(X_{23} = 2\), \(X_{24} = 6\), \(X_{31} = 4\), \(X_{32} = 0\), \(X_{33} = 6\) and the optimal solution is 28

### 3.4 SOURCE PROGRAM ALGORITHM FOR ZERO SUFFIX METHOD IN C# NET PROGRM

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace zerosuffixmethod
{
    class Program
    {
        public static int costmatrows;
        public static int costmatcols;
        public static int[,] costmat;
        public static int demands;
```
public static int finalrows;
public static int finalcols;
public static int[,] originalcostmat;
public static int supplys;
public static int cost;
public static double[,] suffix;
static void Main(string[] args)
{
    costmat = new int[50, 50];
    suffix = new double[50,50];
    originalcostmat = new int[50, 50];
    demands = 0;
    supplys = 0;
    costmatrows = 0;
    costmatcols = 0;
    int[] supplymat = new int[30];
    int[] demandmat = new int[30];
    Console.Write("\nEnter the no.of Demand elements");
    demands = Int32.Parse(Console.ReadLine());
    costmatcols = demands-1;
    Console.Write("\nEnter the no.of Supply Elements");
    supplys = Int32.Parse(Console.ReadLine());
    costmatrows = supplys-1;
    Console.WriteLine("\nEnter the Supply values");
    int totalsupply = 0;
    int totaldemand = 0;
int i = 0;
int extra = 0;
int temp = 0;
for (i = 0; i < supplys; i++)
{
    temp = Int32.Parse(Console.ReadLine());
    supplymat[i] = temp;
    totalsupply = totalsupply + temp;
}
Console.WriteLine("\nEnter the Demand values");
temp = 0;
for (i = 0; i < demands; i++)
{
    temp = Int32.Parse(Console.ReadLine());
    demandmat[i] = temp;
    totaldemand = totaldemand + temp;
}
int j = 0;
Console.WriteLine("\nEnter the Cost matrix elements");
for (i = 0; i < supplys; i++)
{
    for (j = 0; j < demands; j++)
    {
        costmat[i, j] = Int32.Parse(Console.ReadLine());
    }
}
//if supply less than the demand
if (totalsupply < totaldemand)
{
    extra = totaldemand - totalsupply;
    supplymat[supplys] = extra;
    costmatrows = supplys;
    for (i = 0; i < supplys + 1; i++)
        Console.WriteLine(" 
 " + supplymat[i]);
    for (j = 0; j < demands; j++)
        costmat[supplys, j] = 0;
    Console.WriteLine("\n The Costmatrix is:");
    for (i = 0; i < supplys + 1; i++)
    {
        for (j = 0; j < demands; j++)
        {
            Console.Write("  " + costmat[i, j]);
        }
        Console.Write("\n");
    }
}

if (totaldemand < totalsupply)
{
    extra = totalsupply - totaldemand;
    costmatcols = demands;
    demandmat[demands] = extra;
    for (i = 0; i < demands + 1; i++)
        Console.WriteLine("\n ");
}
Console.WriteLine("\n " + demandmat[i]);
for (i = 0; i < supplys; i++)
{
    costmat[i, demands] = 0;
}
Console.WriteLine("\n The Costmatrix is:");
for (i = 0; i < supplys ; i++)
{
    for (j = 0; j < demands+1; j++)
    {
        Console.Write("  " + costmat[i, j]);
    }
    Console.Write("\n");
}
finalrows = costmatrows;
finalcols = costmatcols;
for (i = 0; i <= costmatrows; i++)
{
    for (j = 0; j <= costmatcols; j++)
    {
        originalcostmat[i, j] = costmat[i, j];
    }
}
int k = 0;
int rowminimum = 0;
// Checking row minimum
for (i = 0; i <= costmatrows; i++)
{
    for (j = 0; j <= costmatcols; j++)
    {
        rowminimum = rowmin(i);
        if (rowminimum != 0)
        {
            for (k = 0; k <= costmatcols; k++)
            {
                costmat[i, k] = costmat[i, k] - rowminimum;
            }
            j = costmatcols + 1;
        }
    }
}

// Cost matrix after row minimized
Console.WriteLine("\nCost Matrix after the row Minimized");
for (i = 0; i <= costmatrows; i++)
{
    for (j = 0; j <= costmatcols; j++)
    {
        Console.Write(" "+costmat[i, j]);
    }
    Console.Write("\n");
}
/Checking for Column Minimum
int colminimum = 0;
for (j = 0; j <= costmatcols; j++)
{
    for (i = 0; i <= costmatrows; i++)
    {
        colminimum = colmin(j);
        if (colminimum != 0)
        {
            for (k = 0; k <= costmatrows; k++)
            {
                costmat[k, j] = costmat[k, j] - colminimum;
            }
            i = costmatrows + 1;
        }
    }
}

//Cost matrix after the column minimized
Console.WriteLine("\nCost matrix after the column minimized");
for (i = 0; i <= costmatrows; i++)
{
    for (j = 0; j <= costmatcols; j++)
    {
        Console.Write(" "+costmat[i, j]);
    }
    Console.Write("\n");
Finding the adjustant of zeros..
for (i = 0; i <= costmatrows; i++)
{
    for (j = 0; j <= costmatcols; j++)
    {
        if (costmat[i, j] == 0)
        {
            adjustant(i, j);
        }
    }
}

Console.Write("\n \n Suffix Values\n");
//Displaying the suffix matrix;
for (i = 0; i <= costmatrows; i++)
{
    for (j = 0; j <= costmatcols; j++)
    {
        Console.Write(" "+ suffix[i, j]);
    }
    Console.Write("\n");
}
while (finalrows >= 0)
{
    int suffixmaxrow = 0;
    int suffixmaxcol = 0;
int extra1;
int i1 = 0;
int j1 = 0;
suffixmax(ref suffixmaxrow, ref suffixmaxcol);
if (supplymat[suffixmaxrow] <= demandmat[suffixmaxcol])
{
    extra1 = demandmat[suffixmaxcol] - supplymat[suffixmaxrow];
demandmat[suffixmaxcol] = extra1;
supplys = supplys - 1;
for (i = 0; i < supplys; i++)
{
    supplymat[i] = supplymat[i + 1];
}
for (i = 0; i < 10; i++)
{
    for (j = 0; j < 10; j++)
    {
        suffix[i, j] = 0.0;
    }
}
cost = cost + (supplymat[suffixmaxrow] * originalcostmat[suffixmaxrow, suffixmaxcol]);
if (suffixmaxrow == 0)
{
    costmatrows = costmatrows - 1;
    finalrows = finalrows - 1;
for (i1 = 0; i1 <= costmatrows; i1++)
{
    for (j1 = 0; j1 <= costmatcols; j1++)
    {
        costmat[i1, j1] = costmat[i1 + 1, j1];
    }
}

else if (suffixmaxrow == costmatrows)
{
    costmatrows = costmatrows - 1;
    finalrows = finalrows - 1;
}

else if (suffixmaxrow != 0 && suffixmaxrow != costmatrows)
{
    costmatrows = costmatrows - 1;
    finalrows = finalrows - 1;
    int add = 0;
    for (i1 = 0; i1 <= costmatrows; i1++)
    {
        if (i1 >= suffixmaxrow)
        {
            add = add + 1;
        }
    }
    for (j1 = 0; j1 <= costmatcols; j1++)
{ 
    costmat[i1, j1] = costmat[i1 + add, j1]; 
}

if (supplymat[suffixmaxrow] >= demandmat[suffixmaxcol])
{
    extra1 = supplymat[suffixmaxrow] - demandmat[suffixmaxcol];
    supplymat[suffixmaxrow] = extra1;
    cost = cost + (demandmat[suffixmaxrow] * originalcostmat[suffixmaxrow, suffixmaxcol]);
    demands = demands - 1;
    for (i = 0; i < demands; i++)
    {
        demandmat[i] = demandmat[i + 1];
    }
    if (suffixmaxcol == 0)
    {
        costmatcols = costmatcols - 1;
        finalcols = finalcols - 1;
        for (i1 = 0; i1 <= costmatrows; i1++)
        {
        
    }
for (j1 = 0; j1 <= costmatcols; j1++)
{
    costmat[i1, j1] = costmat[i1, j1 + 1];
}

else if (suffixmaxcol == costmatcols)
{
    costmatcols = costmatcols - 1;
    finalcols = finalcols - 1;
}
else if (suffixmaxcol != 0 && suffixmaxcol != costmatcols)
{
    costmatcols = costmatcols - 1;
    finalcols = finalcols - 1;
    int add = 0;
    for (i1 = 0; i1 <= costmatrows; i1++)
    {
        for (j1 = 0; j1 <= costmatcols; j1++)
        {
            if (j1 >= j)
            {
                add = add + 1;
            }
            costmat[i1, j1] = costmat[i1 + add, j1];
        }
    }
Console.WriteLine("\nCost Matrix ");
for (i = 0; i <= costmatrows; i++)
{
    for (j = 0; j <= costmatcols; j++)
    {
        Console.Write(" "+costmat[i,j]);
    }
    Console.Write("\n");
}
//Displaying the suffix matrix;
Console.WriteLine("\n");
for (i = 0; i <= costmatrows; i++)
{
    for (j = 0; j <= costmatcols; j++)
    {
        Console.Write(" "+suffix[i,j]);
    }
    Console.WriteLine("\n");
}
rowminimum = 0;
for (i = 0; i <= costmatrows; i++)
{
    for (j = 0; j <= costmatcols; j++)

{
rowminimum = rowmin(i);
if (rowminimum != 0)
{
for (k = 0; k <= costmatcols; k++)
{
costmat[i, k] = costmat[i, k] - rowminimum;
}
j = costmatcols + 1;
}
}

colminimum = 0;

for (j = 0; j <= costmatcols; j++)
{
for (i = 0; i <= costmatrows; i++)
{
colminimum = colmin(j);
if (colminimum != 0)
{
for (k = 0; k <= costmatrows; k++)
{
costmat[k, j] = costmat[k, j] - colminimum;
}
i = costmatrows + 1;
}
for (i = 0; i <= costmatrows; i++)
{
    for (j = 0; j <= costmatcols; j++)
    {
        if (costmat[i, j] == 0)
        {
            adjustant(i, j);
        }
    }
}

Console.WriteLine("Cost of Transportation is:" + cost);
Console.ReadLine();

public static void suffixmax(ref int row, ref int col)
{
    int i = 0;
    int j = 0;
    double tempmax = 0;
    for (i = 0; i <= finalrows; i++)
    {
        for (j = 0; j <= finalcols; j++)
if (tempmax < suffix[i, j])
{
    tempmax = suffix[i, j];
    row = i;
    col = j;
}

public static int rowmin(int row)
{

    int min = 3000;
    int j = 0;

    for (j = 0; j <= costmatcols; j++)
    {
        if (min > costmat[row, j])
        {
            min = costmat[row, j];
        }
    }

    return min;
}
public static int colmin(int col)
{
    int min = 3000;
    int i = 0;

    for (i = 0; i <= costmatrows; i++)
    {
        if (min > costmat[i, col])
        {
            min = costmat[i, col];
        }
    }
    return min;
}

public static void adjustant(int row, int col)
{
    int rowpos = row;
    int colpos = col;
    int temp = 0;
    double elements = 0;

    if (row != costmatrows)
    {
        rowpos = row + 1;
        temp = temp + costmat[rowpos, colpos];
        elements = elements + 1;
    }
if (row != 0) {
    rowpos = row - 1;
    temp = temp + costmat[rowpos, colpos];
    elements = elements + 1;
}

if (col != costmatcols) {
    colpos = col + 1;
    temp = temp + costmat[row, colpos];
    elements = elements + 1;
}

if (col != 0) {
    colpos = col - 1;
    temp = temp + costmat[row, colpos];
    elements = elements + 1;
}

double average = temp / elements;
suffix[row, col] = average;
3.4 CONCLUSION

Thus the zero suffix method provides an optimal value of the objective function for the transportation problem. This method carries systematic procedure, and is very easy to understand. This method can be extended to assignment and traveling salesman problems to get optimal solution. The proposed method is an important tool for the decision makers when they are handling various types of logistic problems.