A study of Transportation Problem for an Essential Item of Southern Part of North Eastern Region of India as an OR Model and Use of Object Oriented Programming

Nabendu Sen¹, Tanmoy Som², Banashri Sinha¹

¹Department of Mathematics, Assam University, Silchar; ²Department of Applied Mathematics, BHU, Varanasi, India

Summary
In this paper we formulate an OR model from the collected data concerning with the transportation of an essential item, rice, from different suppliers of Silchar to different destinations in Mizoram. In this study an attempt has been made to analyze the optimal solution with basic feasible solutions obtained using different methods [10]. Also programs have been developed using object oriented programming, C++.

Key words: O.R model, feasible solution, VAM, object oriented programming.

1. Introduction
Mizoram, a part of southern region of North East India, not being well connected from the other parts of the nation as well as the North East region, depends on the market of the adjacent district Silchar of Assam for its essential goods like rice, flour, salt etc. Different suppliers of Silchar regularly supply rice to the different markets of Mizoram. [6-9]. As such the related data has been collected from the concerned suppliers for the purpose of the mathematical formulation.

2. Tables, Figures and Equations
2.1 Tables and Figures
Table 1 shows the distance of different destinations in Mizoram from Silchar district.

<table>
<thead>
<tr>
<th>Place</th>
<th>Distance (Km)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kolashib</td>
<td>90</td>
</tr>
<tr>
<td>Serchip</td>
<td>300</td>
</tr>
<tr>
<td>Aizwal</td>
<td>180</td>
</tr>
<tr>
<td>Saiha</td>
<td>450</td>
</tr>
</tbody>
</table>

We use the following code for the destination Kolashib, Serchip, Aizwal, Saiha.
X* for Kolashib, Y* for Serchip, Z* for Aizwal and U* for Saiha.
Transportation cost per quintal of rice effective from 2007 from different suppliers to the different destinations (as mentioned above) is displayed in the table 2.

<table>
<thead>
<tr>
<th>Supplier</th>
<th>Quantity Available (Quintal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A*</td>
<td>8000</td>
</tr>
<tr>
<td>B*</td>
<td>9200</td>
</tr>
<tr>
<td>C*</td>
<td>6250</td>
</tr>
<tr>
<td>D*</td>
<td>4900</td>
</tr>
<tr>
<td>E*</td>
<td>6100</td>
</tr>
</tbody>
</table>

The next table 3 shows the quantity available with these suppliers for a particular year.

2.2 Table 2

<table>
<thead>
<tr>
<th>Supplier</th>
<th>X*</th>
<th>Y*</th>
<th>Z*</th>
<th>U*</th>
</tr>
</thead>
<tbody>
<tr>
<td>A*</td>
<td>6</td>
<td>12</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>B*</td>
<td>8</td>
<td>10</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>C*</td>
<td>6</td>
<td>11</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>D*</td>
<td>6</td>
<td>11</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>E*</td>
<td>7</td>
<td>13</td>
<td>8</td>
<td>5</td>
</tr>
</tbody>
</table>

N.B. i) A*, B*, C*, D*, E* are the code names of the suppliers. ii) The transportation cost of different suppliers to the same destination varies to some extent due to their own policies.

The next table 4 shows the total demand of the destination X*, Y*, Z*, U* from these suppliers during the year.
3. Methods of Obtaining Initial Basic Feasible Solutions:
We apply here the following methods to get the initial basic feasible solutions:
(a) Northwest Corner Method (see [10]).
(b) Vogel Approximation Method (see [10]).
(c) Least Cost Method (see [10]).
(d) Row Minima Method (see [10]).
(e) Column Minima Method (see [10]).

4. Formulation of Model:
In this problem we make a transportation schedule for rice, as being the essential commodity (main food of the people) for the state of Mizoram. Combining the data of the tables 2, 3 and 4, we get the following transportation model to determine an optimal schedule so as to minimize the transportation cost for rice to different markets of Mizoram.

### Table 4

<table>
<thead>
<tr>
<th>Destination</th>
<th>Demand (Quintals)</th>
</tr>
</thead>
<tbody>
<tr>
<td>X*</td>
<td>5000</td>
</tr>
<tr>
<td>Y*</td>
<td>2000</td>
</tr>
<tr>
<td>Z*</td>
<td>10000</td>
</tr>
<tr>
<td>U*</td>
<td>6000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Source</th>
<th>Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>A*</td>
<td>60</td>
</tr>
<tr>
<td>B*</td>
<td>58</td>
</tr>
<tr>
<td>C*</td>
<td>62</td>
</tr>
<tr>
<td>D*</td>
<td>65</td>
</tr>
<tr>
<td>E*</td>
<td>70</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Destination</th>
<th>X*</th>
<th>Y*</th>
<th>Z*</th>
<th>U*</th>
<th>V*</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>A*</td>
<td>60</td>
<td>120</td>
<td>75</td>
<td>180</td>
<td>0</td>
<td>8000</td>
</tr>
<tr>
<td>B*</td>
<td>58</td>
<td>100</td>
<td>60</td>
<td>165</td>
<td>0</td>
<td>9200</td>
</tr>
<tr>
<td>C*</td>
<td>62</td>
<td>110</td>
<td>65</td>
<td>170</td>
<td>0</td>
<td>6250</td>
</tr>
<tr>
<td>D*</td>
<td>65</td>
<td>115</td>
<td>80</td>
<td>175</td>
<td>0</td>
<td>4900</td>
</tr>
<tr>
<td>E*</td>
<td>70</td>
<td>135</td>
<td>85</td>
<td>195</td>
<td>0</td>
<td>6100</td>
</tr>
</tbody>
</table>

Demand | 5000 | 2000 | 10000 | 6000 | 11450 |

Now apply the three different methods for initial basic feasible solution.

**Initial Basic Feasible Solution by Northwest Corner Method:**

After applying this method which leads to the following final table as:

From NWCM Method, we find number of occupied cell is (5+5-1=9) which is exactly same as m+n-1. Therefore we get the initial feasible solution as
\[ x_{11} = 5000, \ x_{12} = 2000, \ x_{13} = 1000, \ x_{23} = 9000, \ x_{34} = 200, \ x_{35} = 5800, \ x_{55} = 450, \ x_{55} = 4900, \ x_{55} = 6100. \]

Total T.C is Rs. 21, 74,000/
INITIAL BASIC FEASIBLE SOLUTION BY VOGEL APPROXIMATION METHOD:

After applying this method which leads to the following final table as:

<table>
<thead>
<tr>
<th>Source/Destination</th>
<th>X*</th>
<th>Y*</th>
<th>Z*</th>
<th>U*</th>
<th>V*</th>
</tr>
</thead>
<tbody>
<tr>
<td>A*</td>
<td>5000</td>
<td>120</td>
<td>75</td>
<td>180</td>
<td>0</td>
</tr>
<tr>
<td>B*</td>
<td>58</td>
<td>100</td>
<td>60</td>
<td>165</td>
<td>0</td>
</tr>
<tr>
<td>C*</td>
<td>62</td>
<td>110</td>
<td>65</td>
<td>170</td>
<td>0</td>
</tr>
<tr>
<td>D*</td>
<td>65</td>
<td>80</td>
<td>175</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E*</td>
<td>15</td>
<td>85</td>
<td>195</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

From Vogel Approximation Method, we find number of occupied cell is (5+5-1=9) which is exactly same as \( m+n-1 \). Therefore we get the initial feasible solution as:

\[
\begin{align*}
    x_{11} &= 5000, & x_{13} &= 750, & x_{15} &= 2250, \\
    x_{23} &= 1200, & x_{24} &= 6000, & x_{33} &= 5800, \\
    x_{35} &= 450, & x_{45} &= 4900, & x_{55} &= 6100.
\end{align*}
\]

Total T.C is Rs 12,73,000/

INITIAL BASIC FEASIBLE SOLUTION BY LEAST COST METHOD:

After applying this method which leads to the following final table as:

<table>
<thead>
<tr>
<th>Source/Destination</th>
<th>X*</th>
<th>Y*</th>
<th>Z*</th>
<th>U*</th>
<th>V*</th>
</tr>
</thead>
<tbody>
<tr>
<td>A*</td>
<td>60</td>
<td>120</td>
<td>75</td>
<td>180</td>
<td>0</td>
</tr>
<tr>
<td>B*</td>
<td>58</td>
<td>100</td>
<td>60</td>
<td>165</td>
<td>0</td>
</tr>
<tr>
<td>C*</td>
<td>62</td>
<td>110</td>
<td>65</td>
<td>170</td>
<td>0</td>
</tr>
<tr>
<td>D*</td>
<td>65</td>
<td>80</td>
<td>175</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E*</td>
<td>15</td>
<td>85</td>
<td>195</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

From Least cost Method, we find number of occupied cell is (5+5-1=9) which is exactly same as \( m+n-1 \). Therefore we get the initial feasible solution as:

\[
\begin{align*}
    x_{11} &= 5000, & x_{13} &= 750, & x_{15} &= 2250, \\
    x_{23} &= 1200, & x_{24} &= 6250, & x_{42} &= 1900, \\
    x_{43} &= 3000. & x_{52} &= 100, & x_{54} &= 6000
\end{align*}
\]

Total T.C is Rs 24,04,500/

INITIAL BASIC FEASIBLE SOLUTION BY ROW MINIMA METHOD:

After applying this method which leads to the following final table as:

<table>
<thead>
<tr>
<th>Source/Destination</th>
<th>X*</th>
<th>Y*</th>
<th>Z*</th>
<th>U*</th>
<th>V*</th>
</tr>
</thead>
<tbody>
<tr>
<td>A*</td>
<td>60</td>
<td>120</td>
<td>75</td>
<td>180</td>
<td>0</td>
</tr>
<tr>
<td>B*</td>
<td>58</td>
<td>100</td>
<td>60</td>
<td>165</td>
<td>0</td>
</tr>
<tr>
<td>C*</td>
<td>62</td>
<td>110</td>
<td>65</td>
<td>170</td>
<td>0</td>
</tr>
<tr>
<td>D*</td>
<td>65</td>
<td>80</td>
<td>175</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E*</td>
<td>15</td>
<td>85</td>
<td>195</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

From Row minima methods, we find number of occupied cell is (5+5-1=9) which is exactly same as \( m+n-1 \). Therefore we get the initial feasible solution as:

\[
\begin{align*}
    x_{15} &= 5000, & x_{21} &= 5000, & x_{23} &= 750, \\
    x_{25} &= 3450, & x_{33} &= 6250, & x_{42} &= 1900, \\
    x_{43} &= 3000. & x_{52} &= 100, & x_{54} &= 6000
\end{align*}
\]

Total T.C is Rs 23,83,250/

INITIAL BASIC FEASIBLE SOLUTION BY COLUMN MINIMA METHOD:

After applying this method which leads to the following final table as:

<table>
<thead>
<tr>
<th>Source/Destination</th>
<th>X*</th>
<th>Y*</th>
<th>Z*</th>
<th>U*</th>
<th>V*</th>
</tr>
</thead>
<tbody>
<tr>
<td>A*</td>
<td>60</td>
<td>120</td>
<td>75</td>
<td>180</td>
<td>0</td>
</tr>
<tr>
<td>B*</td>
<td>58</td>
<td>100</td>
<td>60</td>
<td>165</td>
<td>0</td>
</tr>
<tr>
<td>C*</td>
<td>62</td>
<td>110</td>
<td>65</td>
<td>170</td>
<td>0</td>
</tr>
<tr>
<td>D*</td>
<td>65</td>
<td>80</td>
<td>175</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E*</td>
<td>15</td>
<td>85</td>
<td>195</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

From Column minima methods, we find number of occupied cell is (5+5-1=9) which is exactly same as \( m+n-1 \). Therefore we get the initial feasible solution as:

\[
\begin{align*}
    x_{15} &= 5000, & x_{21} &= 5000, & x_{23} &= 750, \\
    x_{25} &= 3450, & x_{33} &= 6250, & x_{42} &= 1900, \\
    x_{43} &= 3000. & x_{52} &= 100, & x_{54} &= 6000
\end{align*}
\]

Total T.C is Rs 23,83,250/
From Row minima methods, we find number of occupied cell is (5+5=10) which is exactly same as m+n-1. Therefore we get the initial feasible solution as 
\[ x_{13} = 1550, \quad x_{14} = 1100 \quad x_{15} = 750, \]
\[ x_{21} = 5000, \quad x_{23} = 4200 \quad x_{32} = 2000, \]
\[ x_{33} = 4250 \quad x_{44} = 9900, \quad x_{55} = 6000 \]
Total T.C is Rs22,10,000/

5. OPTIMALITY:
Taking initial basic feasible solution due to Vogel’s approximation method ,we now proceed for optimality using MODI method. Here we determine a set of \( u_i \) and \( v_j \) starting with \( u_i=0 \) and using the relation \( c_{ij} = u_i + v_j \) for occupied basic cells as shown below.
\[ c_{1i} = u_1 + v_1 = 60 = 0 + v_1 = v_1 = 60, \quad c_{13} = u_1 + v_3 = 75 = 0 + v_3 = v_3 = 75, \]
\[ c_{22} = u_2 + v_2 = 100 = u_2 + v_2 = v_2 = 75, \quad c_{23} = u_2 + v_3 = 60 = u_2 + 75 = u_3 = v_3 = 75, \]
\[ c_{34} = u_3 + v_4 = 165 = 75 = 165 - 75 + v_4 = v_4 = 180, \quad c_{33} = u_3 + v_3 = 65 = u_3 + 75 = u_3 = v_3 = -10, \]
\[ c_{35} = u_3 + v_5 = 10 = 10 + v_5 = v_5 = 10, \quad c_{45} = u_4 + v_5 = 0 = u_4 + 10 = u_4 = 10 \]
We now find net evaluations for unoccupied cells by using the relation \( d_{ij} = z_{ij} - c_{ij} \)
\[ d_{12} = z_{12} - c_{12} = u_1 + v_2 - 120 = 5, d_{14} = z_{14} - c_{14} = u_1 + v_4 = 180 = 0, \]
\[ d_{15} = z_{15} - c_{15} = u_1 + v_5 = 10 = 180 - 5, d_{21} = z_{21} - c_{21} = u_2 + v_1 = 58 = -13, \]
\[ d_{23} = z_{23} - c_{23} = u_2 + v_3 = 75 = 58 - 13, d_{24} = z_{24} - c_{24} = u_2 + v_4 = -5 = 75 - 70, \]
\[ d_{25} = z_{25} - c_{25} = u_2 + v_5 = 62 = 62 - 62, d_{32} = z_{32} - c_{32} = u_3 + v_2 = 0 = 75 - 75, \]
\[ d_{33} = z_{33} - c_{33} = u_3 + v_3 = 60 = 60, d_{34} = z_{34} - c_{34} = u_3 + v_4 = 0 = 60 - 60, \]
\[ d_{35} = z_{35} - c_{35} = u_3 + v_5 = 0 = 0, d_{42} = z_{42} - c_{42} = u_4 + v_2 = 0 = 115 - 115, \]
\[ d_{43} = z_{43} - c_{43} = u_4 + v_3 = 80 = 80, d_{44} = z_{44} - c_{44} = u_4 + v_4 = 0 = 175 - 175, \]
\[ d_{45} = z_{45} - c_{45} = u_4 + v_5 = 70 = 70, d_{53} = z_{53} - c_{53} = u_5 + v_3 = 85 = 85, d_{54} = z_{54} - c_{54} = u_5 + v_4 = 195 = 195 \]

Since \( d_{14} \) is most positive, therefore cell (1, 5) enters the basis. We allocate an unknown quantity \( \theta \) to this cell and identify a closed loop involving basic cells around this entering cell. Now \( \theta = \text{min} \{450, 3000\} = 450 \), so we drop cell (3, 5).
Solving this we get the final optimal table after one iteration as given below:

**Final optimal table:**

<table>
<thead>
<tr>
<th></th>
<th>X²</th>
<th>Y²</th>
<th>Z²</th>
<th>U²</th>
<th>V²</th>
<th>u₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>A²</td>
<td>5000</td>
<td>(5)</td>
<td>2350</td>
<td>180</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>60</td>
<td>20</td>
<td>75</td>
<td>180</td>
<td>(6)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B²</td>
<td>(15)</td>
<td>2000</td>
<td>1200</td>
<td>6500</td>
<td>(5)</td>
<td>(5)</td>
</tr>
<tr>
<td>58</td>
<td>165</td>
<td>165</td>
<td>0</td>
<td>(15)</td>
<td>(5)</td>
<td>(15)</td>
</tr>
<tr>
<td>C²</td>
<td>(12)</td>
<td>(5)</td>
<td>2500</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
</tr>
<tr>
<td>62</td>
<td>110</td>
<td>60</td>
<td>170</td>
<td>195</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D²</td>
<td>(20)</td>
<td>(20)</td>
<td>(20)</td>
<td>(20)</td>
<td>(20)</td>
<td>(20)</td>
</tr>
<tr>
<td>65</td>
<td>115</td>
<td>80</td>
<td>175</td>
<td>195</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E²</td>
<td>70</td>
<td>135</td>
<td>85</td>
<td>195</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>v₁</td>
<td>60</td>
<td>115</td>
<td>75</td>
<td>180</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>

Here all \( d_{ij} \leq 0 \), so an optimal solution has reached as given below:
\[ x_{11} = 5000, \quad x_{13} = 2550, \quad x_{15} = 450, \]
\[ x_{22} = 2000, \quad x_{23} = 1200, \quad x_{24} = 6000, \]
\[ x_{33} = 6250, \quad x_{45} = 4900, \quad x_{55} = 6100. \]
Thus the optimal transportation cost is \( Z = \text{Rs. 12,46,000.00} \)/

6. **Pseudo Code for different methods for initial basic feasible solution:**

6.1 **North West Corner Method**

- define row_max = 5;
- define col_max=5;
- //create supply_array and require_array
- float supply_array[row_max];
- float require_array[col_max];
- //creating the cost matrix and unit matrix
- float cost_matrix[row_max][col_max];
- float unit_matrix[row_max][col_max];
- //initialize cost matrix
- for i:0 to row_max-1
  - cin>>cost_matrix[i][j];
- end loop
end loop
// initialize unit_matrix
for i:0 to row_max-1
for j:0 to col_max-1
unit_matrix[i][j]=0;
end loop
end loop
float cost_minimal = 0.0;
float *supply_ptr;
float *require_ptr;
supply_ptr = &supply_array[0];
require_ptr = &require_array[0];
// initialize supply_array
for i:0 to row_max-1
require_ptr=supply_ptr;
end loop
//initialize require_array
for i:0 to col_max-1;
require_ptr=require_ptr+1;
end loop

// displaying the unit matrix
for i:0 to row_max-1
for j:0 to col_max-1
cout<<unit_matrix[i][j];
end loop
displaying the minimal cost
cout<<"the minimal cost obtained is :"<<cost_minimal;
end

6. Vogel Approximation Method
#define TRUE 1
#define FALSE 0
#define INFINITY 1111
#define N 3
#define M 4
void input(void);
void display(void);
void displayfinal(void);
void diffmin(void);
void table(void);
int max(int *,int *,int);
int min(int,int);
int mini(int *,int *,int);
int condition(void);
int arr[N][M];
int arccopy[N][M];
int value[N][M];
int u[N];
int v[M];
int rowdifmin[N];
int coldifmin[M];
table(void)
int carr[N][M];
int rowmin[N];
int colmin[M];

int condition(void)
{    max(decide,&decidemaxpos,M+N);
    if(decidemaxpos<0 && decidemaxpos<N)
    {        rowdifminmaxpos;
    }
    else if(decidemaxpos=N && decidemaxpos<M+N)
    {        coldifminmaxpos;
    }
    return(TRUE);
    return(FALSE);
}
main point of execution starts from here
{    int i,j;
    table();
    for(i=0;i<3;i++)
    {        for(j=0;j<4;j++)
        {            if(value[i][j]!=0)
                cout<<"U[i+1]+V[j+1]= arr[i][j])";
        }
    }
    getch();}
    void table(void)
    {        int rowdifminmaxpos;
        int coldifminmaxpos;
        int decidemaxpos;
        int temp;
        int temparr[M];
        int i;
        clrsr();
        input();
        diffmin();
        display();
        while(condition())
        {            max(decide,&decidemaxpos,M+N);
                if(decidemaxpos=0 && decidemaxpos<N)
                {                    rowdifminmaxpos=decidemaxpos;
                    for(i=0;i<M;i++)
                            temparr[i]=arr[decidemaxpos][i];
                            mini(temparr,&coldifminmaxpos,M);
                }
                else if(decidemaxpos=N && decidemaxpos<M+N)
                {                    coldifminmaxpos=decidemaxpos-N;
                    for(i=0;i<N;i++)
                            temparr[i]=arr[decidemaxpos][i];
                            temparr[i]=INFINITY;
                            mini(temparr,&rowdifminmaxpos,M);
                }
temp=min(u[rowdiffminmaxpos],v[coldiffminmaxpos]);
value[rowdiffminmaxpos][coldiffminmaxpos]=temp;
if(temp==u[rowdiffminmaxpos])
    for(i=0;i<N;i++)
        arr[i][coldiffminmaxpos]=INFINITY;
    u[rowdiffminmaxpos]=temp;
    v[coldiffminmaxpos]=temp;
else if(temp==v[coldiffminmaxpos])
    for(i=0;i<N;i++)
        arr[i][coldiffminmaxpos]=INFINITY;
    u[rowdiffminmaxpos]=temp;
    v[coldiffminmaxpos]=temp;
}
diffmin();
getch();
display();
getch();
displayfinal();
getch();
void input(void){
    int i,j;
    for(i=0;i<N;i++)
        for(j=0;j<M;j++)
            arr[i][j]=arrcopy[i][j];
    for(i=0;i<N;i++)
        u[i]=19;
    for(j=0;j<M;j++)
        v[j]=37;
    cout<<endl;
    for(i=0;i<N;i++)
        temp[i]=v[i];
    for(i=0;i<M+N;i++)
        temp[i]=u[i];
    for(i=0;i<N;i++)
        for(j=0;j<M;j++)
            arr[i][j]=arrcopy[i][j];
    cout<<endl;
    for(i=0;i<N;i++)
        for(j=0;j<M;j++)
            arr[i][j]=arrcopy[i][j];
    for(i=0;i<N;i++)
        for(j=0;j<M;j++)
            arr[i][j]=arrcopy[i][j];
    for(i=0;i<N;i++)
        for(j=0;j<M;j++)
            arr[i][j]=arrcopy[i][j];
    for(i=0;i<N;i++)
        for(j=0;j<M;j++)
            arr[i][j]=arrcopy[i][j];
    if(temp[i]!=0)
        for(i=0;i<N;i++)
            for(j=0;j<M+N;i++)
                if(value[i][j]!=0)
                    cout<<arr[i][j]<<"\"<<value[i][j];
            cout<<endl;
}
void diffmin(void){
    int i,j;
    for(i=0;i<N;i++)
        for(j=0;j<M;j++)
            if(value[i][j]!=0)
                cout<<arr[i][j]<<"\"<<value[i][j];
            cout<<endl;
    if(flag==0)
        flag=1;
    else
        return(FALSE);
}

int min(int a,int b){
    if(a>b)
        return(b);
    else
        return(a);
}

int mini(int a,int *aminpos,int n){
    int i;
    amin=a[0];
    *aminpos=0;
    for(i=0;i<n;i++)
        if(amin<a[i])
            {amin=a[i]; *aminpos=i; }
    return(amin);
}

int max(int *a,int *amaxpos,int n){
    int i;
    amax=a[0];
    *amaxpos=0;
    for(i=0;i<n;i++)
        if(amax>a[i])
            {amax=a[i]; *amaxpos=i; }
    return(amax);
}

int main(void){
    int i,j;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    for(i=0;i<N;i++)
        for(j=0;j<M;j++)
            arr[i][j]=arrcopy[i][j];
    for(i=0;i<N;i++)
        for(j=0;j<M;j++)
            arr[i][j]=arrcopy[i][j];
    for(i=0;i<N;i++)
        for(j=0;j<M;j++)
            arr[i][j]=arrcopy[i][j];
    for(i=0;i<N;i++)
        for(j=0;j<M;j++)
            arr[i][j]=arrcopy[i][j];
    for(i=0;i<N;i++)
        for(j=0;j<M;j++)
            arr[i][j]=arrcopy[i][j];
    if(temp[i]!=0)
        for(i=0;i<N;i++)
            for(j=0;j<M+N;i++)
                if(value[i][j]!=0)
                    cout<<arr[i][j]<<"\"<<value[i][j];
            cout<<endl;
}
void displayfinal(void){
    int i,j;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    for(i=0;i<N;i++)
        for(j=0;j<M;j++)
            if(value[i][j]!=0)
                cout<<arr[i][j]<<"\"<<value[i][j];
            cout<<endl;
    if(flag==0)
        flag=1;
    else
        return(FALSE);
}

int mini(int a,int *aminpos,int n){
    int i;
    amin=a[0];
    *aminpos=0;
    for(i=0;i<n;i++)
        if(amin<a[i])
            {amin=a[i]; *aminpos=i; }
    return(amin);
}

int max(int *a,int *amaxpos,int n){
    int i;
    amax=a[0];
    *amaxpos=0;
    for(i=0;i<n;i++)
        if(amax>a[i])
            {amax=a[i]; *amaxpos=i; }
    return(amax);
}

int main(void){
    int i,j;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    for(i=0;i<N;i++)
        for(j=0;j<M;j++)
            arr[i][j]=arrcopy[i][j];
    for(i=0;i<N;i++)
        for(j=0;j<M;j++)
            arr[i][j]=arrcopy[i][j];
    for(i=0;i<N;i++)
        for(j=0;j<M;j++)
            arr[i][j]=arrcopy[i][j];
    for(i=0;i<N;i++)
        for(j=0;j<M;j++)
            arr[i][j]=arrcopy[i][j];
    for(i=0;i<N;i++)
        for(j=0;j<M;j++)
            arr[i][j]=arrcopy[i][j];
    if(temp[i]!=0)
        for(i=0;i<N;i++)
            for(j=0;j<M+N;i++)
                if(value[i][j]!=0)
                    cout<<arr[i][j]<<"\"<<value[i][j];
            cout<<endl;
}
void displayfinal(void){
    int i,j;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    for(i=0;i<N;i++)
        for(j=0;j<M;j++)
            if(value[i][j]!=0)
                cout<<arr[i][j]<<"\"<<value[i][j];
            cout<<endl;
}
```c
{     min1=arr[i][0];     arrmin1pos=0;
    for(j=0;j<M;j++)
    {        if(arr[i][j]<min1)
        {
            min1=arr[i][j];            arrmin1pos=j;
        }
    }
    if(arrmin1pos==1)
    {        min2=arr[i][0];            arrmin2pos=0;
    }
    else
    {        min2=arr[i][1];            arrmin2pos=1;
    }
    for(j=0;j<M;j++)
    {        if(arr[i][j]<min2 && j!=arrmin1pos)
        {
            min2=arr[i][j];                arrmin2pos=j;
        }
    }
    rowdiffmin[i]=min2-min1;
    decide[i]=rowdiffmin[i];
}
for(i=0;i<M;i++)
{     min1=arr[0][i];     arrmin1pos=0;
    for(j=0;j<N;j++)
    {        if(arr[j][i]<min1)
        {
            min1=arr[j][i];                arrmin1pos=j;
        }
    }
    if(arrmin1pos==1)
    {        min2=arr[0][i];            arrmin2pos=0;
    }
    else
    {        min2=arr[1][i];            arrmin2pos=1;
    }
    for(j=0;j<N;j++)
    {        if(arr[j][i]<min2 && j!=arrmin1pos)
        {
            min2=arr[j][i];                arrmin2pos=j;
        }
    }
    coldiffmin[i]=min2-min1;
    decide[i+N]=coldiffmin[i];
}
void display(void)
{     int i,j;     cout<<endl;     for(i=0;i<N;i++)
    {        for(j=0;j<M;j++)
        {            cout<< arr[i][j];
        }
        cout<< u[i]<<""<< rowdiffmin[i];        printf("\n");
    }
    cout<<endl;
    for(i=0;i<M;i++)
    {        cout<< v[i];
    }
    cout<<endl;
    for(i=0;i<M;i++)
    {        cout<< coldiffmin[i];
    }
}
6.3 Least cost method –
struct matrix
{
    int r;
    int c;
};
define row_max R
define col_max C;
matrix find_minloc(float*);
/* create unit_matrix & initialize it to zero */
float unit_matrix[R][C];
for i=0 to R-1;
    float j=0 to C-1
    unit_matrix[i][j]=0;
end loop;
end loop;
/* create supply_array and demand_array */
float supply_array[R];
float require_array[C];
float *cost_matrix_ptr;
const_matrix_ptr = & cost_matrix[0][0];
while(count<R-C+4)
{    float minr_array[C]=0;
    float minc_array[R]=0;
    find_min_cost_matrix(cost_matrix[i][j]);
    struct matrix min_loc=find_min_loc(cost_matrix[i][j]);
    int a = i;
    int b = min_loc;
    int c=j;
    int x = 0;
    j=a;
    if(require_array[min_loc.c]>supply_array[min_loc.r])
    {
        unit_matrix[min_loc.c][min_loc.c]= supply_matrix[min_loc.r];
        require_array[min_loc.c]=require_array[min_loc.c]-
        unit_matrix[min_loc.r][min_loc.c];
        supply_array[min_loc.c]=supply_array[min_loc.c]-
        unit_matrix[min_loc.r][min_loc.c];
        delete_row_costmatrix(cost_matrix[min_loc.r][min_loc.c]);
        // the row is deleted
        if(min_loc.r==0)
            i=i+1;
        //construct new cost matrix(cost_matrix[i][j])
        count++;
        float cost = cost + 
        unit_matrix[min_loc.r][min_loc.c]*cost_matrix[min_loc.r][min_l
        oc.c];
        construct_new_cost_matrix(cost_matrix[i][j]);
        delete_row_costmatrix(cost_matrix[min_loc.r][min_loc.c]);
    }
    if (require_array[min_loc.c]<=
        supply_array[min_loc.r])
    {        unit_matrix[min_loc.r][min_loc.c]=require_array[min_loc.c]-
        unit_matrix[min_loc.r][min_loc.c];
        supply_array[min_loc.c]=supply_array[min_loc.c]-
        unit_matrix[min_loc.r][min_loc.c];
        delete_row_costmatrix(cost_matrix[min_loc.r][min_loc.c]);
        // the row is deleted
        if(min_loc.r==0)
            i=i+1;
        //construct new cost matrix(cost_matrix[i][j])
        count++;
        float cost = cost + 
        unit_matrix[min_loc.r][min_loc.c]*cost_matrix[min_loc.r][min_l
        oc.c];
        construct_new_cost_matrix(cost_matrix[i][j]);
    }
    // display the final cost
    cout<<"the final cost is ":<<cost;
    //display the final unit matrix
    int l,m;
}```
for l:0 to R-1
for m:0 to C-1
    cout<<unit_matrix[l][m];
End loop;
End loop;

6.4 Row Minima Method
define row_max R;
define col_max C;
/* create initial matrix
float cost_matrix[R][C];
for i:0 to R-1
    for y:0 to C-1
        cin>>cost_matrix[i][j]
    end loop
end loop
i=0;
j=0;
/* Create unit matrix and initialize it to zero
float unit_matrix[R][C];
for i:0 to R-1
    for j:0 to C-1
        unit_matrix[i][j]=0;
    end loop
end loop

/* create supply array and demand array
float supply_matrix[R];
float require_matrix[C];
float*cost_matrix_ptr;
cost_matrix_ptr = &cost_matrix[0][0];
int count=0;
while(count<R-C+4)
{
    float minr_array[C]=0;
    float minc_array[R]=0;
    create_minr_array(cost_matrix[i][j]);
    find_minr_array(cost_matrix[i][j]);
    int min_loc = find_min_loc(cost_matrix[i][j]);
    int a = i;
    int b = min_loc;
    int c = j;
    int r = 0;
    if (require_array[min_loc].c>supply_array[min_loc].r)
    {
        unit_matrix[a][b] = supply_array[min_loc].r;
        require_array[min_loc].r = require_array[min_loc].r - unit_matrix[a][b];
        supply_array[min_loc].r = supply_array[min_loc].r - unit_matrix[a][b];
        j=j+1;
        count++;
        cost+=unit_matrix[a][b]*cost_matrix[a][b];
        continue;
    }
    if (require_array[min_loc]<supply_array[y])
    {
        int x = min_loc;
        int y = 0;
        unit_matrix[a][b] = supply_array[y];
        require_array[min_loc]=require_array[min_loc]-unit_matrix[a][b];
        supply_array[y]=supply_array[y]-unit_matrix[a][b];
        i=i+1;
        y=y+1;
        cost+=unit_matrix[a][b]*cost_matrix[a][b];
        continue;
    }
    if (supply_array[0]==require_array[min_loc]&==0)
    {
        find 2_min_array(cost_matrix[i][j]);
        int loc2= find loc2min_array(cost_matrix[i][j]);
        unit_matrix[a][loc2]=0;
        i=i+1;
        j=1;
        cost=cost+unit_matrix[a][b]*cost_matrix[a][b];
        x=x+1;
        continue;
    }
} /* display unit_matrix
int l,m;
for l:0 to R-1
for m:0 to C-1
    cout<<unit_matrix[l][m];
/* display the final min_cost
cout<<"the final minimal cost is" <<cost;

6.5 Column minima method -
Pseudo-code -
define row_max R;
define col_max C;
/* create initial cost matrix
float cost_matrix[R][C];
for i:0 to R-1
    for j:0 to C-1
        cin>>cost_matrix[i][j];
end loop
end loop
i=0,j=0;
/* create unit matrix & initialize it to zero
float unit_matrix[R][C];
for i:0 to R-1
    for j:0 to C-1
        unit_matrix[i][j]=0;
    end loop
end loop

/* create supply array and demand array
float supply_matrix[R];
float require_matrix[C];
float *cost_matrix_ptr;
cost_matrix_ptr = &cost_matrix[0][0];
int count=0;
x=0;
while (count<R-C+4)
{
    float minr_array[C]=0;
    float minc_array[R]=0;
    create_minr_array(cost_matrix[i][j]);
    find_minr_array(cost_matrix[i][j]);
    int min_loc = find_min_loc(cost_matrix[i][j]);
    int a = min_loc;
    int b = j;
    int c = j;
    int r = 0;
    if (require_array[y]>supply_array[min_loc].c)
    {
        unit_matrix[a][b] = supply_array[min_loc].c;
        require_array[y] = require_array[y] - unit_matrix[a][b];
        supply_array[min_loc] = supply_array[min_loc] - unit_matrix[a][b];
        int loc2 = find 2_min_loc(cost_matrix[i][j]);
        unit_matrix[a][loc2]=0;
        i=i+1;
        j=1;
        cost=cost+unit_matrix[a][b]*cost_matrix[a][b];
        x=x+1;
        continue;
    }
} /* create supply array & demand array
float supply_array[R];
float demand_array[C];
float*cost_matrix_ptr;
cost_matrix_ptr = &cost_matrix[0][0];
int count=0;
x=0;
while (count<R-C+4)
{
    float minr_array[C]=0;
    float minc_array[R]=0;
    create_minr_array(cost_matrix[i][j]);
    find_minr_array(cost_matrix[i][j]);
    int min_loc = find min_loc(cost_matrix[i][j]);
    int a = min_loc;
    int b = j;
    int c = j;
    int r = 0;
    if (require_array[y]>supply_array[min_loc])
    {
        unit_matrix[a][b] = supply_array[min_loc];
        require_array[y] = require_array[y] - unit_matrix[a][b];
        supply_array[min_loc] = supply_array[min_loc] - unit_matrix[a][b];
        j=j+1;
        count++;
        cost = cost + unit_matrix[a][b]*cost_matrix[a][b];
        continue;
    }
if (require_array[y]<supply_array[min_loc])
{
    unit_matrix[a][b] = require_array[j];
    require_array[j] = require_array[j] – unit_matrix[a][b];
    supply_array[min_loc] = supply_array[min_loc] – unit_matrix[a][b];
    cost = cost + unit_matrix[a][b]*cost_matrix[a][b];
    j=j+1;
    continue;
}

if ( supply_array[min_loc]==require_array[0])
{
    find loc2=2min array(cost_matrix[i][j]);
    int loc2 = find loc2min array(cost_matrix[i][j]);
    unit_matrix[loc2][b]= 0;
    j=j+1;
    i=i;
    x=x+1;
    cost = cost + unit_matrix[a][b]*cost_matrix[a][b];
}

/* display unit_matrix
int 1,m;
for l:0 to R-1
for m:0 to C-1
cout<<unit_matrix;
*/
/* display the final min_cost
cout<<"the final minimal cost is"<<cost;

7. Conclusion.
If this above optimal schedule is adopted by the suppliers of rice to Mizoram it would not only involve minimization of the transportation cost but it would also minimize the consumption of fuel in transporting the goods by the different carriers on the other hand. The optimal solution obtained in this present investigation shows much more closeness with initial basic feasible solution obtained by Vogel approximation methods. The comparison of optimal solution have been made with other methods of finding initial solutions and observe that Vogel’s method give the better initial feasible solutions which are closer to optimal solution. The object oriented programs using c++ have been developed and the compared with computed results for initial basic feasible solutions. The comparison shows that the computed results tally with the results obtained c++ programming. Pseudo code for said programs is given for better understanding.

8. Acknowledgments
Authors are thankful to the suppliers for providing necessary data in this regards. Authors are also thankful to Mr. Panthadeep Bhattacharjee, Assam University, silchar for providing technical support during this work.

References

Dr.Nabendu Sen is working as an Assistant Professor, Department of Mathematics, Assam University, Silchar (India). He has received Ph.D (Mathematics) in 2008 from Assam University, Silchar. His area of Research includes Optimization and Mathematical Modeling. He has published several papers journals of good repute. He has reviewed many research papers in reputed Journals.

Ms. Banashri Sinha is currently pursuing her research work under the supervision of Dr. Nabendu Sen in Assam University, silchar (India) on Optimization. She has completed her M.Sc (Mathematics) from Guwahati University, Assam (India) in 2008.

Dr.Tanmooy Som is serving as a Professor in the Department of Applied Mathematics, Institute of Technology, Banaras Hindu University, Varanasi, India. He was awarded Ph.D. in Mathematics by Banaras Hindu University, Varanasi in 1986. He has published more than 55 research papers in the areas of ‘Functional Analysis, Optimization Modeling, Fuzzy Set Theory and Image Processing in the Journals of National and International repute. He is also a reviewer of many National and International Journals and member of many academic bodies. He has delivered several invited talks at National as well as International level Seminars & Conferences. He has produced a good number of M. Phil. and Ph.D. s.