ISSN:1991-8178

Australian Journal of Basic and Applied Sciences

Journal home page: www.ajbasweb.com

Minimization of Transportation cost for an Essential Item of Southern Part India as an OR Model and Use of OOPs

[1]M.Kavitha and [2]Dr.V.Vinoba

[1]*Research Scholar,K.N.G.Arts College/Bharathidasan University,Trichy,Tamil Nadu,India*
[2]*Assistant Professor of Mathematics, K.N.G.Arts C(W),Thanjavur,-7*

ARTICLE INFO

Article history:
Received 06march 2015
Accepted 06march 2015

Keywords:
O.R model, basic feasible solution, Vogel's approximation method, oops.

ABSTRACT

In this paper we formulate an OR model from the collected data concerning with the transportation of essential, from different suppliers of Cuddalore to different destinations in Puducherry. In this study an attempt has been made to analyze the optimal solution with basic feasible solutions obtained using different methods.Then programs have been developed using OOPs.

To Cite This Article: M.Kavitha and Dr.V.Vinoba., Minimization of Transportation cost for an Essential Item of Southern Part India as an OR Model and Use of OOPs. ***Aust. J. Basic & Appl. Sci.,*** *9(15): 60-69, 2015*

INTRODUCTION

Puducherry, a part of south India, depends on the market of the adjacent district Cuddalore for its essential goods like rice, flour, salt etc. Different suppliers of Cuddalore regularly supply rice to the different markets of Puducherry. 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 Pudhucherry from Cuddalore district.

Table1

Place	Distance (Km)
Mahe	595
Pudhucherry	21
Yanam	843
Karaikal	109

We use the following code for the destination Mahe, Pudhucherry, Yanam, Karaikal.

X* for Mahe, Y* for Pudhucherry, Z* for Yanam and U* for Karaikal.

Transportation cost per quintal of rice from 2014 from different suppliers to the different destinations (as mentioned above) is displayed in the table 2.

Table 2: X* Y* Z* U*

A*	60	120	75	180
B*	58	100	60	165
C*	62	110	65	170
D*	65	115	80	175
E*	70	135	85	195

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

Table 3

Supplier	Quantity Available (Quintal)
A*	8000
B*	9200
C*	6250
D*	4900
E*	6100

Corresponding Author: M.Kavitha, Research Scholar, K.N.G.Arts College/Bharathidasan University,Trichy,Tamil Nadu, India.
E-mail: mkavitha.saran@gmail.com

Australian Journal of Basic and Applied Sciences, 9(15) Special 2015, Pages: 60-69

The next table 4 shows the total demand of the destination X*, Y*, Z*, U* from these suppliers during the year

Table 4:

Destination	Demand (Quintals)
X*	5000
Y*	2000
Z*	10000
U*	6000

3. Methods of Obtaining Initial Basic Feasible Solutions:

The following methods to get the initial basic feasible solutions:

(a) Northwest Corner Method
(b) Vogel Approximation Method
(c) Least Cost Method
(d) Row Minima Method
(e) Column Minima Method

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 Tamil Nadu. 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 Pudhucherry.

	X*	Y*	Z*	U*	Availability
A*	60	120	75	180	8000
B*	58	100	60	165	9200
C*	62	110	65	170	6250
D*	65	115	80	175	4900
E*	70	135	85	195	6100
Demand	5000	2000	10000	6000	

Here $\Sigma a_i = 34450$, $\Sigma b_j = 23000$. Since $\Sigma a_i \neq \Sigma b_j$, we introduce a dummy destination V* with requirement of 11450 units and zero (0) transportation cost, shown in the next table in the form of balanced transportation problem.

	X*	Y*	Z*	U*	V*	
A*	60	120	75	180	0	8000
B*	58	100	60	165	0	9200
C*	62	110	65	170	0	6250
D*	65	115	80	175	0	4900
E*	70	135	85	195	0	6100
Demand	5000	2000	10000	6000		11450

The three different methods for initial basic feasible solution.

Ibfs By North-West Corner Method:

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

Source /Destination	X*	Y*	Z*	U*	V*
A*	5000	2000	1000	180	0
	60	120	75		
B*	58	100	60	9000	200
				165	0
C*	62	110	65	5800	450
				170	0
D*	65	115	80	175	4900
					0
E*	70	135	85	195	6100
					0

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

x11 =5000, x12 =2000, x13 =1000,
x23 =9000, x24 =200, x34 =5800,
x35 =450, x45 =4900, x55 =6100.
Total T.C is Rs. 21, 74,000/

Ibfs By Vogel Approximation Method:

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

Source /Destination	X*	Y*	Z*	U*	V*
A*	5000		3000	180	0
	60	120	75		
B*	58	2000	1200	6000	0
		100	60	165	
C*	62	110	5800		450
			65	170	0
D*	65	115	80	175	4900
					0
E*	70	135	85	195	6100
					0

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

x11 =5000, x13 =3000, x22 =2000,
x23 =1200, x24 =6000, x33 =5800,
x35 =450, x45 =4900, x55 =6100.
Total T.C is Rs 12,73,000/

Ibfs By Least Cost Method:

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

Source /Destination	X*	Y*	Z*	U*	V*
A*	5000		75	180	2250
	60	120			0
B*	58		60	165	9200
		100			0
C*	62	110	6250		0
			65	170	
D*	65	1900	3000	175	0
		115	80		
E*	70	100	85	6000	0
		135		195	

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

x11 =5000, x13 =750, x15 =2250,
x25 =9200, x33 =6250, x42 =1900,
x43 =3000. X52 =100, x54 =6000
Total T.C is Rs 24,04,500/

Ibfs By Row Minima Method:

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

Source /Destination	X*	Y*	Z*	U*	V*
A*			75	180	3000
	60	120			0
B*	5000		750	165	3450
	58	100	60		0
C*	62	110	6250		0
			65	170	
D*	65	1900	3000	175	0
		115	80		
E*	70	100	85	6000	0
		135		195	

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

x15 =5000, x21 =5000 x23 =750,
x25 =3450, x33 =6250. X42 =1900,
x43 =3000 ,X52 =100, x54 =6000
Total T.C is Rs 23,83,250/

Ibfs By Column Minima Method:

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

Source /Destination	X*	Y*	Z*	U*	V*
A*			1550	1100	5350
	60	120	75	180	0
B*	5000		4200	165	0
	58	100	60		
C*	62	2000	4250		0

		110	65	170	
D*	65	115	80	4900 175	0
E*	70	100 135	85	195	6100 0

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

x13 =1550, x14 =1100 x15 =750,
x21 =5000, x23 =4200. X32 =2000,
x33 =4250 ,X44 =4900, x55 =6000
Total T.C is Rs22,10,000/

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 ui and vj starting with u1=0 and using the relation cij = ui + vj for occupied basic cells as shown below.

c11= u1 + v1 $\Rightarrow$ 60 = 0 + v1 $\Rightarrow$ v1 = 60, c13 = u1 + v3 $\Rightarrow$ 75 = 0 + v3 $\Rightarrow$ v3 = 75,

c22 = u2 + v2 $\Rightarrow$ 100 = u2 + v2 $\Rightarrow$ v2 = 75, c23 = u2 + v3 $\Rightarrow$60 = u2 + 75 $\Rightarrow$u2 = -15,

c24 = u2 + v4 $\Rightarrow$165 = -15 + v4 $\Rightarrow$v4 =180, c33 = u3 + v3 $\Rightarrow$ 65 = u3 + 75 $\Rightarrow$ u3 = -10,

c35 = u3 + v5$\Rightarrow$ 0 = -10 + v5 $\Rightarrow$ v5 = 10, c45 = u4 + v5$\Rightarrow$ 0 = u4 +10 $\Rightarrow$u4 = -10 and c55 = u5 + v5 $\Rightarrow$0 = u5 +10 $\Rightarrow$ u5 = -10 Find the net evaluation for unoccupied cells by using the relation dij = zij - cij

d12 =z12 – c12 = u1 +v2 – 120 = -5,d14 =z14 – c14 = u1 +v4 – 180 = 0,

d15 =z15 – c15 = u1 +v5 – 0 =10, d21 =z21 – c21 = u2 +v1 – 58 = -13,

d25 =z25 – c25 = u2 +v5 – 0 = -5, d31 =z31 – c31 =u3 +v1 – 62 = -12,

d32 =z32 – c32 = u3 +v2 – 110 = -5, d34 =z34 – c34 = u3 +v4 – 170 = 0,

d41 =z41 – c41 = u4 +v1 – 65 = -15,d42 =z42 – c42 = u4 +v2 – 115 = -10,

d43 =z43 – c43 = u4 +v3 – 80 = -15,d44 =z44 – c44 = u4 +v4 – 175 = -5,

d51 =z51 – c51 = u5 +v1 – 70 = -20,d52 =z52 – c52 = u5 +v2 – 135 = -30,

d53 =z53 – c53 =u5 +v3 – 85 = -20,d54 =z54 – c54 =u5 +v4 – 195 = -25

Then the initial iteration is given

Initial Iteration:

Source /Destination	X*	Y*	Z*	U*	V*	u_i
A*	5000 60	(-5) 20	3000- θ 75	0 180	+ θ (10) 0	0
B*	(-13) 58	2000 100	1200 60	6000 165	(-5) 0	-15
C*	(-12) 62	(-5) 110	5800+ θ ⇢ 65	0 170⇢	(450- θ) ⇢ 0	-10
D*	(-15) 65	(-10) 115	(-15) 80	(-5) 175	0	-10
E*	(-20) 70	(-30) 135	(-20) 85	(-25) 195	6100 0	-10
Y_j	60	115	75	180	10	

Since d15 is most positive, therefore cell (1, 5) enters the basis. We allocate an unknown quantity θ to this cell and identify a closed loop involving basic cells around this entering cell. Now θ = min {450, 3000} = 450, so we drop cell (3, 5). Final optimal table after one iteration as given below:

Final optimal table:

Source /Destination	X*	Y*	Z*	U*	V*	u_i
A*	5000 60	(-5) 20	2500 75	0 180	450 0	0
B*	(-13) 58	2000 100	1200 60	6000 165	(-5) 0	-15
C*	(-12) 62	(-5) 110	6250 65	(0) 170	(0) 0	-10
D*	(-15)	(-10)	(-15)	(-5)	4900	-10

	65	115	80	175	0	
E*	(-20) 70	(-30) 135	(-20) 85	(-25) 195	6100 0	-10
Y_j	60	115	75	180	10	

Here all dij ≤ 0, so an optimal solution has reached as given below:

x11 = 5000, x13 = 2550, x15 = 450,
x22 = 2000, x23 =1200, x24 = 6000,
x33 = 6250, x45 = 4900, x55 = 6100.
The optimal transportation cost is
Z = 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;
float supply_array[row_max];
float require_array[col_max];
float cost_matrix[row_max][col_max];
float unit_matrix[row_max][col_max];
** initialize cost_matrix
for i:0 to row_max-1
for j:0 to col_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
cin>>supply_array;
end loop
//initialize require_array
for i:0 to col_max-1;
cin>>require_array;
float *matrix_ptr;
matrix_ptr=&cost_matrix[0][0];
int r = 0,c=0,x=0,y=0;
while(x<=row_max-1 && y=col_max-1)
{
if(*require_ptr>*supply_ptr)
{
unit_matrix[x][y]=supply_array[x];
require_array[y] = require_array[y]-
unit_matrix[x][y];
supply_array[x] = supply_array[x]-
unit_matrix[x][y];
cost_minimal =
cost_minimal+unit_matrix[x][y];
x=x+1;
supply_ptr=supply_ptr+1;
matrix_ptr=matrix_ptr+col_max;
continue;
}
if(*require_ptr<*supply_ptr)
{
unit_matrix[x][y]=require_array[y];
require_array[y] = require_array[y]-
unit_matrix[x][y];
supply_array[x] = supply_array[x]-
unit_matrix[x][y];
cost_minimal =
cost_minimal+unit_matrix[x][y]*cost_matrix[x][y];
y=y+1;
require_ptr=require_ptr+1;
matrix_ptr=matrix_ptr+1;
continue;
}
if ( *require_ptr==*supply_ptr)
{
unit_matrix[x][y]=require_array[y];
require_array[y] = require_array[y]-
unit_matrix[x][y];
supply_array[x] = supply_array[x]-
unit_matrix[x][y];
cost_minimal=cost_minimal+unit_matrix[x][y]*
cost_matrix[x][y];
y=y+1;
x=x+1;
require_ptr=require_ptr+1;
supply_ptr=supply_ptr+1;
matrix_ptr=matrix_ptr+col_max;
continue;
}
}
// displaying the unit matrix
for i:0 to row_max-1
for j:0 ti col_max-1
cout<<unit_matrix[i][j];
end loop
end loop
//displaying the minimal cost
end
```

6. 2 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 arrcopy[N][M];
int value[N][M];
int u[N];
int v[M];
int rowdiffmin[N];
int coldiffmin[M];
int decide[M+N];
int x[N],y[M]; /* x is u y is v */
{ int i,j;
table(); x[0]=0;
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 rowdiffminmaxpos;
int coldiffminmaxpos;
int decidemaxpos;
int temp;
int temparr[M];
int i;
clrscr();
input();
diffmin();
display();
while(condition())
{ max(decide,&decidemaxpos,M+N);
if(decidemaxpos>=0 && decidemaxpos<N)
{ rowdiffminmaxpos=decidemaxpos;
for(i=0;i<M;i++)
temparr[i]=arr[decidemaxpos][i];
mini(temparr,&coldiffminmaxpos,M);
}
else if(decidemaxpos>=N && decidemaxpos<M+N)
{ coldiffminmaxpos=decidemaxpos-N;
for(i=0;i<N;i++)
temparr[i]=arr[decidemaxpos][i];
temparr[i]=INFINITY;
mini(temparr,&rowdiffminmaxpos,M);
}
temp=min(u[rowdiffminmaxpos],v[coldiffminmaxpos]);

if(temp==u[rowdiffminmaxpos])
{ for(i=0;i<M;i++)
arr[rowdiffminmaxpos][i]=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]=-1;
arr[0][0]=arrcopy[0][0]=5;
arr[0][1]=arrcopy[0][1]=3;
arr[0][2]=arrcopy[0][2]=6;
arr[0][3]=arrcopy[0][3]=2;
arr[1][0]=arrcopy[1][0]=4;
arr[1][1]=arrcopy[1][1]=7;
arr[1][2]=arrcopy[1][2]=9;
arr[1][3]=arrcopy[1][3]=1;
arr[2][0]=arrcopy[2][0]=3;
arr[2][1]=arrcopy[2][1]=4;
arr[2][2]=arrcopy[2][2]=7;
arr[2][3]=arrcopy[2][3]=5;
/* Supply */
/* Demand */ v[0]=16; v[1]=18; v[2]=31; v[3]=25; /00000000000000000000\ 5 3 6 2 19 (1) 4 7 9 1 37 (3) 3 4 7 5 34 (1) 16 18 31 25 (1) (1) (1) (1) ---------- 5 3 6 1111 19 (2) 4 7 9 1111 12 (3) 3 4 7 1111 34 (1) 16 18 31 0 (1) (1) (1) (0) ---------- 5 3 6 1111 19 (2) 1111 1111 1111 1111 0 (0) 3 4 7 1111 34 (1) 4 18 31 0 (2) (1) (1) (0) ---------- 5 1111 6 1111 1 (1) 1111 1111 1111 1111 0 (0) 3 1111 7 1111 34 (4) 4 0 31 0 (2) (0) (1) (0) ---------- 1111 1111 6 1111 1 (1105) 1111 1111 1111 1111 0 (0) 1111 1111 7 1111 30 (1104) 0 0 31 0 (0) (0) (1) (0) ---------- 1111 1111 1111 1111 0 (0) 1111 1111 1111 1111 0 (0) 1111 1111 7 1111 30 (1104) 0 0 30 0 (0) (0) (1104) (0) ---------- 1111 1111 1111 1111 0 (0) 1111 1111 1111 1111 0 (0) 1111 1111 1111 1111 0 (0) 0 0 0 0 (0) (0) (0) (0) ---------- ---------- 5 3|18 6|1 2 4|12 7 9 1|25 3|4 4 7|30 5 1111 1111 7 1111 30 (1104)
X[1][2] = 18
X[1][3] = 1
X[2][1] = 12
X[2][4] = 25
X[3][1] = 4
X[3][3]=30 /00000000000000000000/}
void displayfinal(void)
{ int 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++)
if(value[i][j]==0)
cout<< arr[i][j];
else
cout<< arr[i][j]<<"|"<< value[i][j];
cout<<endl;
}
cout<<endl;
for(i=0;i<N;i++)
for(j=0;j<M;j++)
if(value[i][j]!=0)
cout<<X[i+1][j+1]= value[i][j];}
int condition(void)
{ int i; int flag; int temp[M+N]; flag=1;
for(i=0;i<N;i++)
temp[i]=u[i];
for(;i<M+N;i++)
temp[i]=v[i];
for(i=0;i<M+N;i++)
{
if(temp[i]!=0)
flag=0;
}
if(flag==0)
return(TRUE);
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; int amin;
amin=a[0]; *aminpos=0;
for(i=0;i<n;i++)
{ if(a[i]<amin)
{ amin=a[i]; *aminpos=i; }
}
return(amin);}
int max(int *a,int *amaxpos,int n)
{ int i; int amax; amax=a[0]; *amaxpos=0;
for(i=0;i<n;i++)
{ if(a[i]>amax)
{ amax=a[i]; *amaxpos=i; }
}
return(amax);}
void diffmin(void)
{ int min1,min2; int arr min1pos,arr min2pos; int
i,j;
for(i=0;i<N;i++)
{ min1=arr[i][0]; arr min1pos=0;
for(j=0;j<M;j++)
{ if(arr[i][j]<min1)
{ min1=arr[i][j]; arr min1pos=j; }
} if(arrmin1pos==1)
{ min2=arr[i][0]; arr min2pos=0; }
else
{ min2=arr[i][1]; arrmin2pos=1; }
for(j=0;j<M;j++)
{ if(arr[i][j]<min2 && j!=arr min1pos)
{ min2=arr[i][j]; arr min2pos=j; } }
rowdiffmin[i]=min2-min1;
decide[i]=rowdiffmin[i];
}
for(i=0;i<M;i++)
{ min1=arr[0][i]; arr min1pos=0;
for(j=0;j<N;j++)
{ if(arr[j][i]<min1)
{ min1=arr[j][i]; arr min1pos=j;
}
}
if(arr min1pos==1)
{ min2=arr[0][i]; arr min2pos=0;
}
else
{ min2=arr[1][i]; arr min2pos=1;
}
for(j=0;j<N;j++)
{
if(arr[j][i]<min2 && j!=arr min1pos)
{ min2=arr[j][i]; arr min2pos=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];
}
out<<endl;
for(i=0;i<M;i++)
cout<< v[i];
cout<<endl;
for(i=0;i<M;i++)
cout<< coldiffmin[i];
} end;
```

6.3 Least cost method –

```
struct mat
{
int r;
int c;
};
define row_max R
define col_max C;
matrix find_minloc(float*);
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;
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;
cost_matrix_ptr = & cost_matrix[0][0];
while(count<R-C+4)
{
float minr_array[C]=0;
float minc_array[R]=0;
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.r]-unit_matrix[min_loc.r][min_loc.c];
delete_row_cost matrix(cost_matrix[min_loc.r][min_loc.c]);
if(min_loc.r==0)
i=i+1;
count++;
float cost = cost + unit_matrix[min_loc.r][min_loc.c]*cost_matrix[min_loc.r][min_loc.c];
delete_row_cost_matrix(cost_matrix[min_loc.r][min_loc.c]);
construct new cost_matrix (cost_matrix[i[j]);
}
if (require_array[min_loc.c]<=supply_array[min_loc.r])
{
unit_matrix[min_loc.r][min_loc.c]- require_array[min_loc.c];
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.r]-unit_matrix[min_loc.r][min_loc.c];
count++;
float cost = cost + unit_matrix[min_loc.r][min_loc.c]*cost_matrix[min_loc.r][min_loc.c];
if(min_loc.c==0)
j=j+1;
delete_cost_matrix(cost_matrix[min_loc.r][min_loc.c]);
construct_new_cost_matrix (cost_matrix[i][j]);
}
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;
float unit_matrix[R][C];
for i:0 to R-1
for j:0 to C-1
unit_martix[i][j]=0;
end loop
end loop
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)
{
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=I;
count++;
cost=cost+unit_matrix[a][b]*cost_matrix[a][b];
continue;
}
If(require_array[mini_loc]<supply_array[y])
{
unit_matrix[i][b]=require_array[mini_loc];
require_array[min_loc]=require_array[min_loc]-
unit_matrix[i][b];
supply_array[y] = supply_array[y]-
unit_matrix[a][b];
cost=cost+unit_matrix[a][b]*cost_matrix[a][b];
y=y+1;
continue;
}
if
(supply_array[0]==require_array[min_loc]&==0)
{
find 2_min_array(cost_matrix[i][j]);
int loc2= find loc2mini_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];
cout<<" final minimal cost is" <<cost;
```

6.5 Column minima method:

```
Pseudo-code –
define row_max R;
define col_max C;
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;
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 & demand_array
float supply_array[R];
float demand_array[C];
float *cost_matrix_ptr;
cost_matrix_ptr = &cost_matrix[0][0];
int count=0;
int x=0;
while (count<R-C+4)
{
float minr_array[C] = 0;
float minc_array[R]=0;
create_minc_array(cost_matrix[i][j]);
find_minr_array(cost_matrix[i][j]);
int min_loc = find min_loc (cost_matrix[i][j]);
a= min_loc;
b=j;
int c = j;
int x=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 loc2min_array(cost_matrix[i][j]);
int loc2 = find loc2min_array(cost_matrix[i][j]);
unit_matrix[loc2][b]= 0;
j=j+1;
i=j;
x=x+1;
cost = cost +unit
_matrix[a][b]*cost_matrix[a][b];
}
/* display unit_matrix
int l,m;
for l:0 to R-1
for m:0 to C-1
cout<<unit_matrix;
/* display the final min_cost
cout<<" final minimal cost is"<<cost;
```

Conclusion:

If this above optimal schedule is adopted by the suppliers of rice to Tamil Nadu 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 is 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 oops 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

REFERENCES

Hamdy, A., Taha. Operation Research: An Introduction, Eighth Edition, *ISBN*-13: 978-0132555937

Rama Murthy, P., Operation Research, Second Edition, ISBN (13): 978-81-224-2944-2.

Hamdy, A, Taha. TORA Optimizing System Software.

Intrator, J., 1989. A note on the Optimal Solution of a Transportation Problems. Asia –Pacific Journal of OR , 6(2): 207-208.

Patel, S.M., 1992. Resolution of Closed Loop in Transportation Problem. International Journal of Management Systems, 81(1): 35-46.

Rao, S.S., 1987. Optimization Theory and Applications.

Sen, N. and T. Som, 2008a. Mathematical Modeling of Transportation Related Problem of Southern IndiaS and Its Optimal Value. AUJSc, 31(1): 22-27.

Sen, N. and T. Som, 2008b. Mathematical Modeling of Transportation Related Fare Minimization Problem of South-India and an Approach to Its Optimal Value. IJTM, 32(3): 201-208.