Volume 6, No. 1, Jan-Feb 2015
ISSN No. 0976-5697
International Journal of Advanced Research in Computer Science
RESEARCH PAPER
Available Online at www.ijarcs.info

Programming Development on Transportation Problem using c++ and Stepping Stone for Evaluation

Adeoye Akeem .O.*
Department of mathematics/statistics
Federal Polytechnic Offa Nigeria

Babalola John.B
Department of mathematics/statistics
Federal Polytechnic Offa Nigeria

Igbinehi,E.M
Department of mathematics/statistics
Federal Polytechnic Offa Nigeria

Emiola Olawale.K.S
Department of mathematics/statistics
Federal Polytechnic Offa Nigeria

Abstract: The aim of this research is to develop a programm for three methods of solving transportation problems. C++ was used to write the programm on initial feasible solution and optimal solution using North West Corner Rule, Least Cost Rule and Vogel Approximation Method. Stepping stone is used to obtain the optimal solution. From the result of the analysis, we discovered that the Programming on Vogel Approximation Method gives the same result at optimal solution and at low number of iterations. Hence we conclude that programming on Vogel Approximation Method is the best method of finding initial feasible solution, optimal solution and for distribution of good.

Keywords: Optimal, Initial, Feasible, Cost and Solution.

I. INTRODUCTION

Transportation is the movement of people and commodities from one place to another. The progress of any recent organization depends on the effective utilization of transportation system. Moreover, moving from one place to another or distribution of goods and services always take place through transportation system. The transportation model depends on individual objectives and taste in terms of cost, safety, speed and comfortability [1],[4],[6].

Production is the creation of goods and services to satisfy human wants. According to the economic Philosopher Adams Smith (1976), production is not complete until the goods produced reach the final consumer. After production, it is mandate for manufacturers to discharge or distribute the products to the markets or various depots and this can be achieved through effective transportation system [2],[3],[5]. Transportation plays a dominant role in the world economy because the cost of transportation goes a long way in influencing the cost of finished products. That is the lower the transportation cost the cheaper the cost of products. This makes business organization, individuals and government to look for effective ways of solving transportation problems. From the discussion above one can easily come to the conclusion that the progress of any company is directly proportional to the efficiency of the transportation method of the company.

Transportation problem can be described as a way of distributing finished goods from different sources to numerous destinations at a minimum cost or rate. Suppose there are m warehouses, where commodities are stocked an n markets (Locations) where they are needed and supply available in the warehouses be S_1,S_2,S_3 …S_m while the demand at the market be d_1,d_2,d_3…..d_n. Let the unit cost of shipping from warehouse i to market j be ₦Cij. The transpiration problem wants to find an optimal shipping schedule which minimizes the cost of transportation from the warehouse to the markets. The various ways by which a particular commodity can be distributed from source to destination is illustrated by the diagram below.

II. LINEAR PROGRAMMING FORMULATION

In order to formulate the transportation problem as a linear programing problem, we define X_{ij} as the quantity shipped from warehouse i to market j, i = 1,2,….m and j = 1, 2, …n. The number of decision variables is given by the product of m and n

The supply constraint guarantee that the total amounts shipped from any warehouse does not exceed its capacity. The demand constraint guarantee that the total shipped to market including the non-negative constraints, the total number of constraint is (m + n). The market demands of the warehouse is equal to the total demand at the market.

$$\sum_{i=1} S_i = \sum_{j=1} d_j$$

This implies that every available product at the warehouse will be shipped to meet the minimum demand at the markets. In this case, all the supply and the demand constraint would become strict equalities and we shall have a standard transportation problem given by

$$\text{Min } Z = \sum_{i=1}^{m} \sum C_{ij}X_{ij}$$

Subject to

$$\sum_{j=1}^{n} X_{ij} = S_i \quad i = 1, 2, \ldots\ldots\ldots\ldots m \text{ (supply)}$$

$$\sum_{i=1}^{m} X_{ij} = d_i \quad j = 1, 2, \ldots\ldots\ldots\ldots n \text{ (demand)}$$

$X_{ij} \geq 0$ for i and j

The above transportation problem can be expanded as follows.

Min Z = $C_{11}X_{11}+C_{12}X_{12}+….C_{1n}X_{1n}+C_{21}X_{21}+….C_{2n}X_{2n}$ + … $C_{mi}X_{mi}$ + …$C_{mn}X_{mn}$

Subject to

$$X_{11} + X_{12} + …+ X_{1n} = S_1$$
$$X_{21} + X_{22} + …+ X_{2n} = S_2$$
$$… \quad … \quad …$$
$$X_{m1} + X_{m2} + …+ X_{mn} = S_m$$
$$X_{11} + X_{21} + …+ X_{m1} = d_1$$
$$X_{12} + X_{22} + …+ X_{m2} = d_2$$
$$/ \quad /$$
$$/ \quad /$$
$$X_{1n} + X_{2n} + …+ X_{mn} = d_n$$

$S1 \geq 0$, $dj \geq 0$ for i = 1,2, …m, j = 1,2,…n

III. METHOD OF SOLVING TRANSPORTATION PROBLEM

The following are three methods employed in establishing an initial feasible solution, they are

a. North West Corner Rule Method
b. Least Cost Rule Method
c. Vogel Approximation Method

IV. METHOD FOR ESTIMATING OPTIMAL SOLUTION

After an Initial Feasible Solution to the distribution problem has been obtained through North West Corner Rule, Least Cost Rule and Vogel Approximation Method, alternative solution must be evaluated. There are two straight forward method for calculating the effect of alternative allocation. In this research a program develops will be used to find the initial feasible solution and optimal solution using stepping stone method.

A. Stepping Stone Method:

Procedure for stepping stone method

a. Choose the empty cell unused square to be evaluated.
b. Beginning with the selected empty cell, trace a closed path (Moving horizontally and vertically) from this empty cell via stone squares (used square) back to the original cell. Only one closed path exists for each empty cell in a given solution. Although the path may skip over non-empty (stone) or empty cells and may cross over itself. Corners of the closed path may occur only at the stone squares and the unused square (empty cell) being evaluated.
c. Assign plus (+) and Minus (-) signs alternatively at each corner square of the closed path, beginning with a plus sign at the empty cell. Assign these signs by starting in either a clockwise or anti-clockwise direction. The positive and negative signs represent the addition or subtraction of 1 unit to a cell.
d. Determine the net change in the costs as a result of the changes made in tracing the path. Summing the unit cost in each cell with a plus sign will give the addition to the cost. The decrease in cost is obtained by summing the unit cost in each cell with a negative sign.
e. If all the improvement indices are greater than or equal to zero stop. The solution obtained is optimal. Otherwise go to step 6.
f. Develop a new solution and go to step 1 to develop a new solution we shift a smaller quantity of the stone that has negative figure to the most negative cell of the improvement index.
g. Repeat the above steps until the solution is optimal. That is, the improvement index is greater than or equal to zero.

V. DESCRIPTION OF PROGRAM DEVELOPED

C++ is the programming language used to developed a programming for solving transportation problem using North West, Least Cost and Vogel Approximation Method procedures for solving initial feasible solution and stepping stone procedure for finding the optimal solution.

C++ is one of computer programming languages which is a High Level Language (HLL). It is an Object Oriented Programming (OOP) Language. It is also a scientifically oriented and it is used in handling mathematical problems. Some of the characteristics that make C++ different from other common languages are.

a. It bridges the gap between conventional High Level Language (HLL) and Machine Language (ML).
b. It is more efficient than other High Level Language (HLL)
c. It is more portable and convectional in nature
d. It has more concise source code
e. It isolates its machine dependent feature to its library formation

VI. PROGRAM SOURCE CODE ON TRANSPORTATION

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>
#include <dos.h>
#include <time.h>
#define MAXC 500
#define MAXR 500
#define MT 250000
/* GLOBAL VARIABLE DECLARATION */
int NOSUP, NODEM, CP, CT, INIT, IT,IPR, N9, MENUNEXT,cc;
int MOVEX[MT], MOVEY[MT],OPT,MINI,MINJ,NOWI,NOWJ,INDIC;
float C[MAXR][MAXC], FLOW[MAXR][MAXC],SUPPLY[MAXR], S1[MAXR];
float DEMAND[MAXC], D1[MAXC], U[MAXR], V[MAXC], EC[MAXR+1][MAXC+1];
float R[MAXR][MAXC];
float TCOST, CMIN,XMIN,CT2;
int P[500][2*MAXC+5];//
int ITERATE;
int ID,IDES,IF1,IOPTIMAL,ISOU,IX,IY,LT,NR;
   double CC,QT,TT,XINFCC;
char SW, ch;
char *plant[] = {"S1","S2","S3","S4","S5","S6","S7","S8","S9","S10","S11","S12","S13","S14","S15","S16","S17","S18","S19","S20",
```

```
"S21","S22","S23","S24","S25","S26","S27","S28","S29","
S30","S31","S32","S33","S34","S35","S36","S37","S38","S
39","S40"};
char                    *depot[]                    =
{"D1","D2","D3","D4","D5","D6","D7","D8","D9","D10","
D11","D12","D13","D14","D15","D16","D17","D18","D19
","D20",
"D21","D22","D23","D24","D25","D26","D27","D28","D2
9","D30","D31","D32","D33","D34","D35","D36","D37","
D38","D39","D40"};
FILE *fp;
int IERROR;
char ERR_MSG[100];

void INPUT_MODE()
{
   int i, j, response;
      printf("
**********************************************
**** \n");
      printf("                    COMPUTARIZATION OF
TRANSPORTATION MODEL\n");
      printf("                    THIS PROGRAM CAN
ANALYSIS \n");
      printf("              INITIAL FEASIBLE SOLUTION
ALLOCATION \n");
      printf("                BY THREE METHOD\n");
      printf("
**********************************************
*** \n");
      printf("              1.      NORTH-WEST CORNER
RULE \n");
      printf("           2.     LEAST COST RULE \n");
      printf("            3.      VOGEL APPROXIMATION
\n");
   printf("\n\n\n");
   printf("HOW MANY SOURCES? ");
   scanf("%d",&NOSUP);
  // printf("LABEL SOURCES AND PRESS ENTER ");
   printf("HOW MANY DESTINATIONS? ");
   scanf("%d",&NODEM);
   MENUNEXT = 1;
   //printf("LABEL DESTINATIONS AND PRESS ENTER
");
   //for (i = 1;i<NODEM;i++) scanf("%s", depot[i]);
   do
   {
      printf("SELECT 1 OR 2 OR 3 \n");
      scanf("%d",&INIT);
      if ((INIT < 1 ) || (INIT > 3))
      printf("YOU    HAVE    ENTERED    A    WRONG
CHARACTER, TRY AGAIN! \n");
}
while((INIT < 1 ) || (INIT > 3));
printf("\n\n");
printf("
*****************************************\n");
printf("                HOW MUCH COST IS REQUIRED
\n");
printf("                 BETWEEN EACH SOURCE AND
DESTIANTION  \n");
printf("
*****************************************\n");
for(i=1;i<= NOSUP; i++)
{
   do
   {
      printf("\n\n");
      printf(" from %s \n", plant[i - 1]);
      for(j=1;j<= NODEM;j++)
      {
         printf("      to %s = ", depot[j-1]);
         scanf("%f", &C[i][j]);
         printf("\n");
      }
      printf("\n");
      printf("    IS THE DATA CORRECT ? <Y/N>\n");
      getchar();
      response = getchar();
   }
   while(response =='N' || response =='n');
}
printf("\n\n");
printf("
*****************************************\n");
printf("                    WHAT AMOUNT IS POSSIBLE
\n");
printf("                FOR EACH SOURCE \n");
printf("
*****************************************\n");
do
{
   for(i=1;i<= NOSUP;i++)
   {
   printf(" %s = ",plant[i-1]);
   scanf("%f", &SUPPLY[i]);
   printf("\n");
}
printf("    IS THE DATA CORRECT ? <Y/N>\n");
getchar();
response = getchar();
}
while (response == 'N' || response == 'n');
printf("\n\n");
printf("
*****************************************\n");
printf("            WHAT AMOUNT IS REQUIRED     \n");
printf("              FOR EACH DESTINATION  \n");
printf("
*****************************************\n");
do
{
   for(j=1;j<= NODEM;j++)
   {
      printf("    %s = ",depot[j-1]);
      scanf("%f", &DEMAND[j]);
      printf("\n");
   }
   printf("    IS THE DATA CORRECT? <Y/N> \n");
   getchar();
   response = getchar();
}
while (response == 'N' || response == 'n');
}
 void Optimal();
  void Seek_Path(int I,int J);
```

```
  void Increase(int I,int J);
  void TCost();
/*BALANCE8*/
void BALANCE()
{
   float ss,ds,diff;
   int i,j;
   ss = 0.0;
   ds=0.0;
   for(i=1;i<=NOSUP;i++) ss = ss + SUPPLY[i];
   for(j=1;j<=NODEM;j++) ds+=DEMAND[j];
   N9 = NODEM;
   diff=ss-ds;
   if(diff == 0.0) return;
   if(diff < 0.0)
   {
      printf("The total supply must be greater than the total
demand.\n");
      printf(" So this problem is infeasible.\n");
      printf("Check the data.\n");
      printf(ERR_MSG," The total supply must be greater
than the total demand.");
      IERROR = 3;
      return;
   }
   NODEM+=1;
   for(i=1;i<=NOSUP;i++) C[i][NODEM]=1E+20;
   DEMAND[NODEM]=diff;
   if(IPR> 1)
   {
      if (diff == 0)
      {
         printf("\n The original problem is balanced."); return;
      }
      if (diff > 0)
      {
         printf("\n The unbalanced problem is now
balanced.");
         return;
      }
   }
      return;
   }
   /*NORTH*/
   void NWCR()
   {
      int i,j;
      for (i=1;i<=NOSUP;i++)
            for(j=1;j<=NODEM;j++)
            FLOW[i][j] = 0;
            for (i=1;i<=NOSUP;i++) S1[i] = SUPPLY[i];
            for(j=1;j<=NODEM;j++) D1[j] = DEMAND[j];
            for (i=1;i<=NOSUP;i++)
            {
               if (fabs(S1[i]) < 0.00001) continue;
               for (j=1;j<=NODEM;j++)
               {
                  if (fabs(D1[j]) < 0.00001 ) continue;
                  if ((fabs(S1[i] - D1[j]) < 0.00001))
                  {
                     FLOW[i][j] = S1[i];
                      S1[i] = 0;
                      D1[j] = 0;
                  }
                  else if (S1[i] < D1[j])
                  {
                     FLOW[i][j] = S1[i];
                     D1[j] = D1[j] - S1[i];
                     S1[i] = 0;
                  }
                  else
                  {
                     FLOW[i][j] = D1[j];
                     S1[i] = S1[i] - D1[j];
                     D1[j] = 0;
                  }
               }
            }
                  }
   void LCR()
    {
      int i,j,icom,MINI,MINJ;
      float CMIN;
      for (i=1;i<= NOSUP;i++)
            for(j=1;j<=NODEM;j++)
            FLOW[i][j] = 0;
      for (i=1;i<= NOSUP;i++) S1[i] = SUPPLY[i];
            for(j=1;j<=NODEM;j++) D1[j] = DEMAND[j];
            do
            {
               icom = 0;
               CMIN = 1E+20;
                MINI=0;
                 MINJ=0;
               for (i=1;i<=NOSUP;i++)
               if (S1[i] > 0.00001)
               {
                  icom = 1;
                  for(j=1;j<=NODEM;j++)
                  if(D1[j]>0.00001)
                  {
                     icom=1;
                     if(CMIN > C[i][j])
                     {
                        CMIN = (C[i][j]);
                        MINI = i;
                        MINJ=j;
                     }
                  }
               }

    if (icom==1)
    {
       if(fabs(S1[MINI] -D1[MINJ])<0.00001)
       {
          FLOW[MINI][MINJ] = S1[MINI]; S1[MINI] = 0;
D1[MINJ] = 0;
       }
    if((fabs(S1[MINI] - D1[MINJ])>=0.00001) && (S1[MINI]
< D1[MINJ]))
       {
          FLOW[MINI][MINJ] = S1[MINI];D1[MINJ]-=
S1[MINI];S1[MINI]=0;
     }
    if((fabs(S1[MINI] -D1[MINJ])>=0.00001) && (S1[MINI]
> D1[MINJ]))
```

```
    {
       FLOW[MINI][MINJ]     =     D1[MINJ];S1[MINI]-=
D1[MINJ];D1[MINJ]=0;
      }
  }
 }
while (icom==1);
if (icom ==0) return;
}
/*VOGEL*/
void VOGEL()
{
  int LB,L;
   int i,A,j,k,icom,MINI,MINJ,imin1,jmin1,itt;
   float CMIN1,CMIN2,DIFF,DIFFT;
  LB = NOSUP + NODEM;
  LB = NOSUP + NODEM;
  LB = LB - 1;
   for(i = 1; i <= NOSUP; i++)
   for (j = 1; j <= NODEM; j++)
   FLOW[i][j] = 0;
     A = 1;
     //LB = 0;
   for(i=1;i<=NOSUP;i++) S1[i] = SUPPLY[i];
   for (j=1;j<=NODEM;j++) D1[j] = DEMAND[j];
 //do {
   for(L = 0; L <= LB; L++){
        icom = 0; DIFF =-1E+20;MINI = 0; MINJ= 0;

  for(i=1;i<=NOSUP;i++)
    {
      if (S1[i] > 0.00001)
      {
         icom = 1; CMIN1 = 1E+20;imin1 = 0;jmin1 = 0;
CMIN2 = 1E+20; itt= 0;
             for(j=1;j<=NODEM;j++){
                if(D1[j] > 0.00001)
                {
                   itt++;
                  if(CMIN1 > C[i][j])
                  {
                     CMIN1 = C[i][j];imin1 = i;jmin1=j;
                }
              }
          }
                  if(itt > 1){
                     for(j = 1;j <= NODEM; j++){
                        if(D1[j] > 0.00001)
                        if((j != jmin1) && (CMIN2 > C[i][j]))
CMIN2 = C[i][j];
                      }
                  }
                  DIFFT = CMIN2 - CMIN1;
                  if(itt == 1) DIFFT = 0;
                  if(DIFF < DIFFT)
                  {
                     DIFF    =    DIFFT;MINI    =    imin1;MINJ
=jmin1;
                  }
               }
            }
    for(j=1;j<=NODEM;j++)
    {
        if (D1[j] > 0.00001)
        {
           icom = 1; CMIN1 = 1E+20;imin1 = 0;jmin1 = 0;
CMIN2 = 1E+20; itt= 0;
              for(i=1;i<=NOSUP;i++){
                 if(S1[i] > 0.00001)
                 {
                    itt++;
                   if(CMIN1 > C[i][j])
                   {
                      CMIN1 = C[i][j];imin1 = i;jmin1=j;
                  }
                }
            }
                   if(itt > 1){
                      for(i = 1;i <= NOSUP; i++){
                         if(S1[i] > 0.00001)
                         if((i != imin1) && (CMIN2 > C[i][j]))
CMIN2 = C[i][j];
                      }
                   }

                   DIFFT = CMIN2 - CMIN1;
                   if(itt == 1) DIFFT = 0;
                   if(DIFF < DIFFT)
                   {
                      DIFF    =    DIFFT;MINI    =    imin1;MINJ
=jmin1;
                   }
                }
             }
if (icom==1)
  {
     if(fabs(S1[MINI] - D1[MINJ])<0.00001)
     {
        FLOW[MINI][MINJ] = S1[MINI]; S1[MINI] = 0;
D1[MINJ] = 0;
        A++;
     }
 if((fabs(S1[MINI] - D1[MINJ])>=0.00001) && (S1[MINI]
< D1[MINJ]))
     {
        FLOW[MINI][MINJ]     =     S1[MINI];D1[MINJ]-=
S1[MINI];S1[MINI]=0;
        A++;

}
 if((fabs(S1[MINI]  -  D1[MINJ])  >=  0.00001)  &&
(S1[MINI] > D1[MINJ]))
     {
        FLOW[MINI][MINJ]     =     D1[MINJ];S1[MINI]-=
D1[MINJ];D1[MINJ]=0;
        A++;

      }
   }
 }//while(A <= LB);
            // if(icom == 1) goto LB;
               if(icom == 0) return;
               }

              /*INISOL*/
              void INISOL()
```

```
{
   int cnt;
   int BLINK,linet,ptr,p;
   int valid,char_count;
   int i,j;
   for(i=1;i<=NOSUP;i++)
   for(j=1;j<=NODEM;j++) EC[i][j] = 0;
   switch(INIT)
   {
      case 1: NWCR(); break;
      case 2: LCR(); break;
      case 3: VOGEL(); break;
   }
   TCOST = 0.0;
   for(i=1;i<=NOSUP;i++)
   for(j=1;j<= NODEM;j++)
   {
      if(FLOW[i][j] == 0) continue;
      EC[i][j] = 1; CT++;
     // TCOST = TCOST + CT++;
      TCOST = TCOST + (C[i][j] * FLOW[i][j]);
   }
    if(CT < CP) printf("\n The initial solution is
not feasiable.");
    else printf("\nThe initial solution is
feasiable.");
    }
void Optimal() {
//Labels: e10, e70, e140, e150
   int I,J;
   ITERATE = 1;
e10: XINFCC=0.0;
   for (I=1; I<=NOSUP; I++)
     for (J=1; J<=NODEM; J++) {
       if (FLOW[I][J] != 0.0) goto e70;
       Seek_Path(I,J);
       Increase(I,J);

e70:;  }
          if (XINFCC>=0.0) {
     IOPTIMAL=1;
     goto e150;
   }
   for (I=1; I<=LT; I++) {
     IX=P[3][I]; IY=P[4][I];
     if (I % 2 == 0) {
       FLOW[IX][IY] -= TT;
       goto e140;
     }
     FLOW[IX][IY] += TT;
e140:;}
e150:if (IOPTIMAL==0) {
   ITERATE = ITERATE + 1;
   goto e10;}
}
void Seek_Path(int I, int J) {
//Labels: e70, e160, e260
    int I1,I2;
    for (I1=1; I1<=NOSUP; I1++)
      for (I2=1; I2<=NODEM; I2++)
        R[I1][I2]=FLOW[I1][I2];
    for (I1=1; I1<=NOSUP; I1++) R[I1][0]=0.0;
    for (I2=1; I2<=NODEM; I2++) R[0][I2]=0.0;
    R[I][J]=1.0;
e70: for (I2=1; I2<=NODEM; I2++) {
      if (R[0][I2]==1.0) goto e160;
      NR=0;
      for (I1=1; I1<=NOSUP; I1++)
        if (R[I1][I2] != 0.0) NR++;
            if (NR!=1) goto e160;
      for (I1=1; I1<=NOSUP; I1++) R[I1][I2]=0.0;
      R[0][I2]=1.0; IF1=1;
e160:;}
    for (I1=1; I1<=NOSUP; I1++) {
      if (R[I1][0]==1.0) goto e260;
      NR=0;
      for (I2=1; I2<=NODEM; I2++)
        if (R[I1][I2] != 0.0) NR++;
      if (NR!=1) goto e260;
      for (I2=1; I2<=NODEM; I2++) R[I1][I2]=0.0;
      R[I1][0]=1.0; IF1=1;
e260:;}
    if (IF1==1) {
      IF1=0; goto e70;
    }
}

void Increase(int I, int J) {
//Labels: e20,e70,e130,e170,e180,e230
    int I1,I2;
    P[1][1]=I;  P[2][1]=J;  IX=I;  IY=J;  ID=1;  CC=0.0;
QT=999999.0;
e20: ID++; IF1=0;
           for (I1=1; I1<=NOSUP; I1++) {
      if (R[I1][IY]==0.0 || I1==IX) goto e70;
      P[1][ID]=I1; P[2][ID]=IY; IX=I1; CC -= C[IX][IY];
      IF1=1; I1=NOSUP;
      if (FLOW[IX][IY] < QT && FLOW[IX][IY] > 0.0)
QT=FLOW[IX][IY];
e70:;}
    if (IF1==0) goto e170;
    ID++; IF1=0;
           for (I2=1; I2<=NODEM; I2++) {
      if (R[IX][I2]==0.0 || I2==IY) goto e130;
      P[1][ID]=IX; P[2][ID]=I2; IY=I2; CC += C[IX][IY];
      IF1=1; I2=NODEM;
e130:;}
    if (IF1==0) goto e170;
    if (IX!=I || IY!=J) goto e20;
    goto e180;
e170:printf(" DEGENERATE SOLUTION !\n");
    return;
e180:if (CC>0.0 || CC>XINFCC) goto e230;
    TT=QT; XINFCC=CC; ID--; LT=ID;
    for (I1=1; I1<=ID; I1++) {
      P[3][I1]=P[1][I1]; P[4][I1]=P[2][I1];
    }
e230:;}
  void TCost() {
   int I,J;
   CT2=0.0;
   for (I=1; I<=NOSUP; I++)
     for (J=1; J<=NODEM; J++) {
       CT2 = CT2 + (FLOW[I][J] * C[I][J]);
       if (FLOW[I][J]==0.0) goto e10;
```

```
    printf("                FROM SOURCE%d TO DESTINATION%d: %8.2f\n", I, J, FLOW[I][J]);
e10:;}
  printf("\n TOTAL TRANSPORT COST: %10.1f\n\n", CT2);
}

        /*main*/

int main()

{
  int i,j,menu10;
  char AB;
  int DATAOPT;
  menu10:
  TCOST = 0;
  //do
  //{

    INPUT_MODE();
    CP = NOSUP + NODEM - 1; CT =0;
    BALANCE();
    INISOL();
    printf("\n\n");

    if(INIT == 1) printf("  INITIAL FEASIABLE SOLUTION OF NWCR\n\n");
    if(INIT == 2) printf("  INITIAL FEASIABLE SOLUTION OF LCR\n\n");
    if(INIT == 3) printf("  INITIAL FEASIABLE SOLUTION OF VOGEL\n\n");
      for(i=1;i<=NOSUP;i++){
            for(j=1;j<= NODEM;j++)
            {

              printf("%f",FLOW[i][j]);
              printf("      ");
            }
            printf("\n");
          }
           printf("Total cost = %f\n\n",(TCOST));

           if(INIT==1) printf("OPTIMAL SOLUTION OF NORTH WEST CORNER RULE\n\n");
           if(INIT==2) printf("OPTIMAL SOLUTION OF LEAST COST RULE\n\n");
           if(INIT==3) printf("OPTIMAL SOLUTION OF VOGEL APPROXIMATION\n\n");
          Optimal();
          TCost();
          printf("\n\n NUMBER OF ITERATION = %d\n\n", ITERATE);
          printf("PRESS 1 TO GO TO MENU? ");
          scanf("%d",&DATAOPT);
          if(DATAOPT == 1) goto menu10;
   //          }while((DATAOPT == 'M') || (DATAOPT == 'm'));
 system("PAUSE");
 return 0;
}
```

VII. APPLICATION

Table 1:- Shows the sources and supply capacity of Seven Up Bottling Company

Sources	Supply Capacity
Plant 1	3,200
Plant 2	3,080
Plant 3	2,720
Total	**9000**

Table.2 Shows the destinations and their requirement of Seven Up Bottling Company PLC Ilorin.

Destination	Demand
Jebba	1,320
Kabba	1,1720
Ogbomosho	2,520
Ekiti	1,240
Oshogbo	2,200
Total	**9000**

Table 3 Shows the unit cost of transportation in Naira (₦) from sources to destinations.

Sources	Jebba	Kabba	Ogbomosho	Ekiti	Oshogo
Plant 1	150	165	150	145	100
Plant 2	148	162	172	127	160
Plant 3	220	228	170	160	155

Plant 1 =	Warehouse 1	=	W_1
Plant 2 =	Warehouse 2	=	W_2
Plant 3 =	Warehouse 3	=	W_3
Jebba =	Market 1	=	M_1
Kabba =	Market 2	=	M_2
Ogbomosho=	Market 3	=	M_3
Ekiti =	Market 4	=	M_4
Oshogbo=	Market 5	=	M_5

Table 4 Shows the combined cost matrix, supply capacity and demand

	M_1	M_2	M_3	M_4	M_5	S_i
W_1	150	165	150	145	100	3,200
W_2	148	162	172	127	160	3080
W_3	220	228	170	160	155	2,720
dj	1,320	1,720	2,520	2,200	9000	

VIII. ANALYSIS OF DATA

When the program developed was used to analyse the data, the following result were obtained.

From the analysis of the data set, using the progam developed in this study North West Corner Rule was optimal at sixth iteration, Least Cost Rule at fourth iteration while Vogel Approximation method is optimal at second iteration. This indicate that Vogel Approximation Method is the best

IX. SUMMARY OF FINDING

Summary of the analysis carried out so far are presented in the table 11

	Methods (Programing)	Initial Feasible Solution	Optimal Solution	No of Iteration
1.	North West Corner Rule	₦1,427,360	₦1,288,480	6
2.	Least Cost Rule	₦1,339,080	₦1,288,480	4
3.	Vogel Approximation	₦1,295,080	₦1,288,480	2

X. CONCLUSION

Since program developed on Vogel Approximation has the advantages of obtaining initial feasible solution; easy to compute and easy to edit when error are committed in the process of data entering. Hence programming developed on Vogel Approximation Method using C++ is the best method of getting Approximate solution to transportation problem and best method of distributing the goods and services.

XI. REFERENCES

[1]. Abel Mizahr and Michael Sillivant (1979): Mathematics for Business and Social Sciences Published by John Willey & Sons Inc.

[2]. Aminu Y.A (1998): Operation Research, for Science and Management Studies, Best Way Publisher Ltd. Offa.

[3]. R.J Vanderbei, Linear Programming: Foundations and Extensions, Kluwer Academic Publishers, Boston, 1996.

[4]. S.J. Wright, Primal-Dual Interior-Point Methods, Society for Industrial and Applied Mathematics, Philadelphia, 1997.

[5]. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein 2001 Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill.

[6]. Michael. J. Todd (February 2002). "The many facets of linear programming" Mathematical Programming 91 (3).(3).