

“Coding Genetic Algorithm for Cluster Analysis Through Basic Programming Techniques and the SAS Macro”

Kaveephong Lertwachara, Louisiana Tech University,
Lertwachara@hotmail.com

ABSTRACT

This paper describes an alternative approach in coding a genetic algorithm. The genetic algorithm is implemented for a cluster analysis using basic programming techniques and the SAS macro. The main code performs a single iteration of cluster analysis. Simple statements such as IF, THEN, ELSE are used at this stage. Then, the SAS macro is used for create global loops and specify some variables.

1. INTRODUCTION

Genetic Algorithm (GA) is a heuristic optimization technique that imitates genetic production using genetic operators to repeatedly manipulate members in the population, generation after generation, attempting to eventually reach an optimum. GAs are nonlinear in nature; therefore, they do not require certain assumptions such as normality and homogeneity of variance, and they are flexible to a variety of forms of objective functions. Moreover, Chiou and Lan (2001) also add that the type of variables and the number of variables used in the analysis do not severely affect the accuracy of the GAs but do affect the computation storage and time of GAs. GAs have been used in a variety of applications such as analyzing credit card fraud, forecasting machine tool loading, capital markets analysis, crop forecasting, product marketing, and property tax analysis. Lertwachara and Cochran (2003) use a genetic algorithm cluster stocks within the Real Estate Investment Trusts and report that the genetic algorithm provides valuable results comparing to analysts' recommendation.

In cluster analysis a large number of objects are classified into a smaller number of meaningful groups based on pre-defined criteria. Therefore, a massive amount of information may be summarized so that it is easily understood and effectively employed. Everitt (2001) states that cluster analysis is a collection of techniques that discover groups in data. These techniques are similar to discrimination methods except that clustering techniques do not generate any discriminating rule and the number of groups is not known prior to the clustering process. Although the techniques originated from biology, practitioners and researchers apply cluster analysis to various applications across disciplines including market segmentation, modeling economic prospects, price discrimination, information retrieval, and disease diagnosis. In market segmentation, for example, marketers can apply an appropriate marketing strategy to each market segment once the market is clustered based on various attributes such as age, gender, disposable income, and geography. The marketer can economically conduct experiments on how the

market responds to a new product by taking a random sample from each market cluster in the entire market. Since the size of the sample is smaller than the size of the entire market, the marketer benefits from cluster analysis because of reduced costs and time associated with the market testing.

This paper describes an alternative approach in coding a genetic algorithm. The genetic algorithm is implemented for a cluster analysis using basic programming techniques and the SAS macro. The main code performs a single iteration of cluster analysis. Simple statements such as IF, THEN, ELSE are used at this stage. Then, the SAS macro is used for create global loops and specify some variables. The remaining of the paper is organized as follows. The next section describes how the initial round of cluster analysis is coded using some simple statements such as IF, THEN, ELSE. Section 3 illustrates how genetic algorithm's operators can be done. Then, section 4 describes how SAS codes are assembled together using SAS Macros.

2. CODING SAS FOR THE INITIAL ITERATION OF CLUSTER ANALYSIS

The GA in this paper imitates clustering process of the k-means algorithm (KM). KM's objective function is to minimize within-group variance, which equates to $\sum \sum \sum (X_{ijk} - M_{jk})^2$ ---- where X_{ijk} is the j^{th} variable of the i^{th} observation in the k^{th} cluster and M_{jk} is the j^{th} variable of the seed for the k^{th} cluster. KM begins with a pre-determined starting centroid, or seed for each cluster. The pre-determined seeds are normally randomly selected. The following code with data step generates initial seeds with values between 0-10.

```

**Randomize Seeds;
DATA Seeds;
  ARRAY M[&CHR, &CL, &VAR];
  NUM=_N_;
  IF NUM > 1 THEN DELETE;
  DO ic=1 to &CHR;
    DO ig=1 to &CL;
      DO ix=1 to &VAR;
        M[ic,ig,ix]=10*RANUNI(0);
      END;
    END;
  END;
  KEEP M1--M&N_M;
RUN;

```

Observations are then grouped on the basis of their distances from the seeds. Each observation is placed into the cluster with the nearest centroid and the centroids are recalculated immediately after all observations are placed into clusters. The procedure can be accomplished through the following code.

```

%MACRO A_Clus;
DATA NextGen;
  SET FirstGen;
  ARRAY M[&CHR, &CL, &VAR];

```

```

    ARRAY X[&VAR];
    ARRAY R[&CHR,&CL];
    ARRAY CLUS[&CHR];
    ARRAY SmallR[&CHR];
    ARRAY TotR[&CHR];
    ARRAY INV[&CHR];
    ARRAY P_INV[&CHR];
    ARRAY TP_INV[&CHR];

    do ic=1 to &CHR; ***** ig is chromosome number;
        do iq=1 to &CL; ***** iq is cluster number;
            R[ic,iq]=0;
        end;
    end;

**Calculating Distance between observation and centroids;
    do ic=1 to &CHR; ***** ig is chromosome number;
        do iq=1 to &CL; ***** iq is cluster number;
            do ix=1 to &VAR; ***** iq is cluster number;
                R[ic,iq]=(x[ix]-m[ic,iq,ix])**2)+R[ic,iq];
            end;
        end;
    end;

**Find the centroid with the nearest distance to observation;
    do ic=1 to &CHR; ***** iq is chromosome number;
        CLUS[ic]=1;
        SmallR[ic]=R[ic,1];
    end;
    do ic=1 to &CHR; ***** ig is chromosome number;
        do iq=2 to &CL; ***** iq is cluster number;
            IF SmallR[ic]>R[ic,iq] THEN CLUS[ic]=iq;
            IF SmallR[ic]>R[ic,iq] THEN SmallR[ic]=R[ic,iq];
        end;
    end;

**Calculate evaluating value;
    do ic=1 to &CHR; ***** ic is chromosome number;
        TotR[ic]+SmallR[ic];
    end;

RUN;
%MEND A_Clus;

```

3. CODING SAS FOR BASIC GENETIC ALGORITHM OPERATORS

As GAs emulate genetic production, members in each generation are usually called chromosomes and represent a feasible solution to the problem. Each chromosome consists of basic elements that are referred to as genes. Each gene represents a possible solution (cluster centroids). As described by Goldberg (1989) and many others, the processes of GAs are as follows. The initial generation is usually randomly selected. Consequently, members of the initial population are randomly selected to be parents of the next generation with probability of selection based on the member's success in the

first generation. A member with a higher evaluating value has a higher chance in the selection. The evaluating value is the inverse of the within-group variance. The following code implements the parents-selection process.

```
%MACRO Worst_C;
DATA W_Chroms (DROP=X1--X&VAR);
  SET NextGen;
  ARRAY M[&CHR,&CL,&VAR];
  ARRAY X[&VAR];
  ARRAY R[&CHR,&CL];
  ARRAY CLUS[&CHR];
  ARRAY SmallR[&CHR];
  ARRAY TotR[&CHR];
  ARRAY INV[&CHR];
  ARRAY P_INV[&CHR];
  ARRAY TP_INV[&CHR];
  ARRAY Counter[&CHR];
  IF NUM=107 THEN
    DO;
      DO ic=1 to &CHR;
        counter[ic] = 1;
      END;

      DO ia=1 To &CHR;
        DO ib=1 To &CHR;
          IF ia NE ib Then
            Do;
IF abs(TotR[ia]-TotR[ib]) < &Accu THEN counter[ia] = counter[ia]+1;
            END;
          END;
        End;

        count=1;
        Do ic=1 to &CHR-1;
          IF Counter[ic]>count THEN count=Counter[ic];
        END;
        DO ic=1 to &CHR;
          INV[ic]=1/(TotR[ic]);
        END;

T_INV=INV1+INV2+INV3+INV4+INV5+INV6+INV7+INV8+INV9+INV10;

        DO ic=1 to &CHR;
          P_INV[ic]=100*INV[ic]/T_INV;
        END;

        TP_INV1=P_INV1/100;

        DO ic=2 to &CHR;
          TP_INV[ic]=TP_INV[ic-1]+P_INV[ic]/100;
        END;

        BEST1=1;

        DO ic=1 to &CHR;
```

```

        IF P_INV[BEST1]<P_INV[ic] THEN BEST1=ic;
    END;

    IF BEST1 = 1 THEN BEST2=2; ELSE BEST2=1;

    DO ic=2 to &CHR;
    IF ic NE Best1 Then
    IF P_INV[Best2]<P_INV[ic] THEN Best2=(ic);
    END;

    Worst1=1;
    DO ic=2 to &CHR;
    IF P_INV[Worst1]>P_INV[ic] THEN Worst1=ic;
    END;

    IF Worst1 = 1 THEN Worst2=2; ELSE WORST2=1;
    DO ic=2 to &CHR;
    IF ic NE Worst1 Then
    IF P_INV[Worst2]>P_INV[ic] THEN Worst2=(ic);
    END;

    END;
ELSE DELETE;
RUN;
%MEND Worst_C;

```

Then the selected chromosomes pass through one or more processes of crossover, mutation, and inversion.

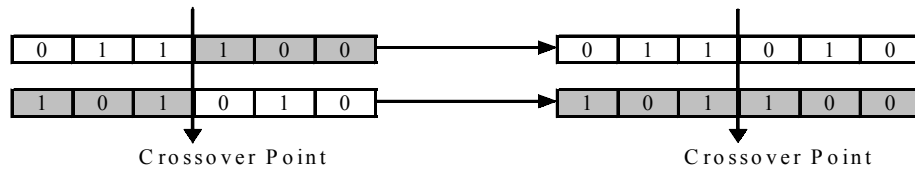


Figure 1: Crossover



Figure 2: Mutation

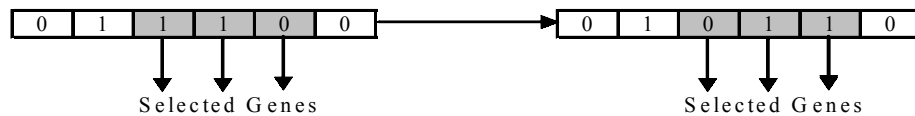


Figure 3: Inversion

Crossover is simply a process of swapping parts of the two selected chromosomes. Figure 1 illustrates a simple example of a crossover. A crossover point is randomly selected. Then, all genes behind the crossover point of the two selected chromosomes are swapped. The following code performs the crossover operation.

```
%MACRO CrossO;
DATA W_C_CO;
    SET W_Chroms;
    ARRAY M[&CHR,&CL,&VAR];
    ARRAY TP_INV[&CHR];
    ARRAY INV[&CHR];
    ARRAY INVV[&CHR];
    ARRAY TP2_INV[&CHR];
    ARRAY NewC1[&CL,&VAR];
    ARRAY NewC2[&CL,&VAR];
IF count < &Quit THEN
DO;
*****Randomly Select 2 Chromosomes;
    RN1=RANUNI(0); *****Random a number between 0 and 1;
    DO ic=1 to &CHR;*****Find out the random number fall into
which chromosomes;
        IF TP_INV[&CHR-ic+1] > RN1 THEN RC1=(&CHR-ic+1);
    END;
    DO ic=1 to &CHR; *****Take out the value of the first selected
chromosome;
        IF ic = RC1 THEN INVV[ic]=0;
        ELSE INVV[ic]=INV[ic];
    END;
    *****ReWeight the Prob;
    T2_INV=INVV1+INVV2+INVV3+INVV4+INVV5+INVV6+INVV7+INVV8+INVV9+INVV
10;
    TP2_INV1=INVV1/T2_INV;
    DO ic=2 to &CHR;
        TP2_INV[ic]=TP2_INV[ic-1]+INVV[ic]/T2_INV;
    END;
    *****Select the second chromosome;
    RN2=RANUNI(0);
    DO ic=1 to &CHR;
        IF TP2_INV[&CHR-ic+1] > RN2 THEN RC2=(&CHR-ic+1);
    END;
    *****Randomize CrossOver Point;
    X_Randm=1+(((&VAR)-1)*RANUNI(0));
    X_Rand=ROUND(X_Randm,1);
    CL_Randm=1+(((&CL)-1)*RANUNI(0));
    CL_Rand=ROUND(CL_Randm,1);
    *****Start Crossover;
    IF CL_Rand > 1 THEN
    DO;
        DO iq=1 to (CL_Rand-1);
            DO ix=1 to &VAR;
                NewC1[iq,ix]=M[RC1,iq,ix];
                NewC2[iq,ix]=M[RC2,iq,ix];
            END;
        END;
    END;
END;
```

```

END;

DO ix=1 to X_Rand;
    NewC1[CL_Rand,ix]=M[RC1,CL_Rand,ix];
    NewC2[CL_Rand,ix]=M[RC2,CL_Rand,ix];
END;

IF X_Rand < (&VAR+1) THEN
DO;
    DO ix=(X_Rand+1) to &VAR;
        NewC1[CL_Rand,ix]=M[RC2,CL_Rand,ix];
        NewC2[CL_Rand,ix]=M[RC1,CL_Rand,ix];
    END;
END;

IF CL_Rand < (&CL+1) THEN
DO;
    DO iq=(CL_Rand+1) to &CL;
        DO ix=1 to &VAR;
            NewC1[iq,ix]=M[RC2,iq,ix];
            NewC2[iq,ix]=M[RC1,iq,ix];
        END;
    END;
END;

*****Substitute Two Worst by Two New Chromosomes;
DO iq=1 to &CL;
    DO ix=1 to &VAR;
        M[WORST1,iq,ix]=NewC1[iq,ix];
        M[WORST2,iq,ix]=NewC2[iq,ix];
    END;
END;

END;
RUN;
%MEND CrossO;

```

Mutation process deviates randomly selected genes. Figure 2 shows how genes are mutated. First, target genes are randomly selected. Then values of the selected genes are changed. Inversion flips the series of genes. The following code executes the mutation operator.

```

%MACRO Mutat;
DATA CO_MU;
    SET W_C_CO;
    ARRAY M[&CHR,&CL,&VAR];
    ARRAY MuC[&NumMu];
    ARRAY MuQ[&NumMu];
    ARRAY MuX[&NumMu];
IF count < &Quit THEN
DO;
*****Start Mutation;
    Do i=1 to &NumMu;
        MuC[i]=1+(((&CHR)-1)*RANUNI(0));
        MuQ[i]=1+(((&CL)-1)*RANUNI(0));
    End;

```

```

        MuX[i]=1+(((&VAR)-1)*RANUNI(0));
        MC=ROUND (MuC[i],1);
        MQ=ROUND (MuQ[i],1);
        MX=ROUND (MuX[i],1);
        M[MC,MQ,MX]=10-M[MC,MQ,MX];
    END;
END;
RUN;
%MEND Mutat;

```

Figure 3 demonstrates a basic inversion operation. First, the GA randomly selects a series of genes. Then the series of selected genes will be inversed. The following code performs the basic inversion.

```

%MACRO Invers;
DATA Inversion;
    SET CO_MU;
    ARRAY M[&N_M];
    ARRAY SM[&N_M];
    IF count < &Quit THEN
    DO;
        *****Randomly Select Two Points;
        INVP1=1+(&N_M-1)*RANUNI(0);
        INVP2=1+(&N_M-1)*RANUNI(0);
        INVP1=ROUND (INVP1,1);
        INVP2=ROUND (INVP2,1);
        *****Figure out Which is Start and Which is End Point;
        DIFF=INVP1-INVP2;
        IF DIFF>0 THEN
            DO;    INVP3=INVP1;
                    INVP1=INVP2;
                    INVP2=INVP3;
                END;
        NumInv=abs (INVP2-INVP1);
        MP_Inv=NumInv/2;
        MP_Inv=Round (Mp_Inv, 1);

        *****Swap Them;
        Do i=1 to MP_Inv-1;
            SM[INVP1+i-1]=M[INVP1+i-1];
            M[INVP1+i-1]=M[INVP2-i+1];
            M[INVP2-i+1]=SM[INVP1+i-1];
        END;
    END;
RUN;
%MEND Invers;

```

The new chromosomes are substituted for chromosomes with low evaluating values from the previous generation. Thus, the new generation consists both of chromosomes with high evaluating values and new chromosomes. This procedure is accomplished by the following code.

```

%MACRO Replace;
DATA Seeds;
    SET Inversion;
    KEEP M1--M&N_M BEST1;
RUN;
%MEND Replace;

%MACRO GA_Seeds;
DATA SeedGA (KEEP=X1--X&NumMu);
    SET SEEDS;
    ARRAY M[&CHR,&CL,&VAR];
    ARRAY X[&CL,&VAR];
    DO iq=1 to &CL;
        DO ix=1 to &VAR;
            X[iq,ix]=M[Best1,iq,ix];
        END;
    END;
END;

```

4. ASSEMBLING CODES TOGETHER

The clustering procedure is continues (using the new centroids) until at least one of the following conditions criterion is satisfied: (1) at least nine out of ten chromosomes achieve the same evaluating value, or (2) the clustering process has run 50 iterations. The code described in previous sections embeds a measurement of the first condition and the code needs to be re-run. The SAS Macros implement the re-execution of the code. The following code evokes the Macros and allows the loop.

```

%MACRO MyGA;
    %DO i=1 %to &Iteration;
        %Upd_Seeds
        %A_Clus
        %Worst_C
        %CrossO
        %Mutat
        %Invers
        %Replace;
    %END;
    %Ite
    %GA_Seeds
%MEND MyGA;
%MyGA

```

It is important to mention that the following settings must be placed at the beginning of the program. In this case, we are solving a clustering problem with 5 clusters (CL = 5) and 5 variables (VAR = 5) using 10 chromosomes (CHR = 10). The GA can stop after 50 iterations (Iteration = 50) or at least nine out of ten chromosomes possess the same evaluating value (Quit = 9). Since the evaluating value can take on any real value on the number line, the chance that any two evaluating values will be exactly the same is very tiny. So, we give it some slacks to the definition of “the same”. If any two evaluating values are within 100 units of each other, we acknowledge that the two values are the same.

```

Title 'Genetic Algorithm';
%LET CHR = 10;
%LET CL = 5;
%LET VAR = 5;
%LET N_M = %eval(&CL*&VAR*&CHR);
%LET NumMu = %eval(&N_M/&CHR);
%LET Iteration = 50;
%LET Accu = 100;
%LET Quit = 9;

```

5. CONCLUDING REMARKS

GA, a heuristic optimization technique, has been employed in a number of applications including cluster analysis. In cluster analysis, GA attempts to discover information in dataset through some genetic operations: crossover, mutation, and inversion. This paper illustrates a way we code GA in SAS using some basic programming techniques with a little help from SAS Macros to put a loop on the procedure and to specify some global parameters. A complete SAS code is provided in Appendix.

Author Information

Kaveephong (Mike) Lertwachara
 Telephone: (318) 513-9341
 E-mail: Lertwachara@hotmail.com

REFERENCES

- Chiou, Y., and L. W. Lan. 2001. "Theory and Methodology: Genetic Clustering Algorithms," *European Journal of Operational Research*, vol 135: 413-427.
- Everitt, B. S. 2001. *Cluster Analysis*. 4th ed. London: Arnold Publishers.
- Goldberg, D. E. 1989. *Genetic Algorithms in Search*. New York: Addison-Wesley.
- Holland, J. H. 1992. "Genetic Algorithms," *Scientific American* vol. 267(12): 66-72.
- Lertwachara, K., and J. J. Cochran. 2003. "Selecting Stocks Using a Genetic Algorithm: A Case of Real Estate Investment Trusts," *Proceeding of the Annual Meeting of the Decision Sciences Institute 2003 (forthcoming)*.
- SAS Institute Inc. 1990. *SAS Language and Procedures Guide, Version 6, 1st ed.* Cary, NC: SAS Institute Inc.
- SAS Institute Inc. 1997. *SAS Macro Language: Reference, 1st ed.* Cary, NC: SAS Institute Inc.

APPENDIX

Complete SAS Code

```
Title 'Genetic Algorithm';
%LET CHR = 10;
%LET cl = 5;
%LET VAR = 5;
%LET N_M = %eval(&CL*&VAR*&CHR);
%LET NumMu = %eval(&N_M/&CHR);
%LET Iteration = 50;
%LET Accu = 100;
%LET Quit = 9;

**Randomize Seeds;

    ARRAY M[&CHR,&CL,&VAR];
    NUM=_N_;
    IF NUM > 1 THEN DELETE;
    DO ic=1 to &CHR;
        DO ig=1 to &CL;
            DO ix=1 to &VAR;
                M[ic,ig,ix]=10*RANUNI(0);
            END;
        END;
    END;
    KEEP M1--M&N_M;

%MACRO Upd_Seeds;
DATA FirstGen;
    IF _n_=1 THEN SET Seeds;
    SET SimDat;
    NUM=_n_;
    DROP n;
RUN;
%MEND Upd_Seeds;

%MACRO A_Clus;
DATA NextGen;
    SET FirstGen;
    ARRAY M[&CHR,&CL,&VAR];
    ARRAY X[&VAR];
    ARRAY R[&CHR,&CL];
    ARRAY CLUS[&CHR];
    ARRAY SmallR[&CHR];
    ARRAY TotR[&CHR];
    ARRAY INV[&CHR];
    ARRAY P_INV[&CHR];
    ARRAY TP_INV[&CHR];

    do ic=1 to &CHR; ***** ig is chromosome number;
        do iq=1 to &CL; ***** iq is cluster number;
            R[ic,iq]=0;
        end;
    end;
```

```

end;

**Calculating Distance between observation and centroids;
do ic=1 to &CHR; ***** iq is chromosome number;
  do iq=1 to &CL; ***** iq is cluster number;
    do ix=1 to &VAR; ***** iq is cluster number;
      R[ic,iq]=(x[ix]-m[ic,iq,ix])**2)+R[ic,iq];
    end;
  end;
end;

**Find the centroid with the nearest distance to observation;
do ic=1 to &CHR; ***** iq is chromosome number;
  CLUS[ic]=1;
  SmallR[ic]=R[ic,1];
end;
do ic=1 to &CHR; ***** iq is chromosome number;
  do iq=2 to &CL; ***** iq is cluster number;
    IF SmallR[ic]>R[ic,iq] THEN CLUS[ic]=iq;
    IF SmallR[ic]>R[ic,iq] THEN SmallR[ic]=R[ic,iq];
  end;
end;

**Calculate evaluating value;
do ic=1 to &CHR; ***** ic is chromosome number;
  TotR[ic]+SmallR[ic];
end;

RUN;
%MEND A_Clus;

%MACRO Worst_C;
DATA W_Chroms (DROP=X1--X&VAR);
  SET NextGen;
  ARRAY M[&CHR,&CL,&VAR];
  ARRAY X[&VAR];
  ARRAY R[&CHR,&CL];
  ARRAY CLUS[&CHR];
  ARRAY SmallR[&CHR];
  ARRAY TotR[&CHR];
  ARRAY INV[&CHR];
  ARRAY P_INV[&CHR];
  ARRAY TP_INV[&CHR];
  ARRAY Counter[&CHR];
  IF NUM=107 THEN
    DO;
      DO ic=1 to &CHR;
        counter[ic] = 1;
      END;

      DO ia=1 To &CHR;
        DO ib=1 To &CHR;
          IF ia NE ib Then
            DO;
IF abs(TotR[ia]-TotR[ib]) < &Accu THEN counter[ia] = counter[ia]+1;
              END;
            END;
          END;
        End;
      End;
    DO;

```

```

count=1;
Do ic=1 to &CHR-1;
    IF Counter[ic]>count THEN count=Counter[ic];
END;
DO ic=1 to &CHR;
    INV[ic]=1/(TotR[ic]);
END;
T_INV=INV1+INV2+INV3+INV4+INV5+INV6+INV7+INV8+INV9+INV10;
DO ic=1 to &CHR;
    P_INV[ic]=100*INV[ic]/T_INV;
END;

TP_INV1=P_INV1/100;

DO ic=2 to &CHR;
    TP_INV[ic]=TP_INV[ic-1]+P_INV[ic]/100;
END;

BEST1=1;

DO ic=1 to &CHR;
    IF P_INV[BEST1]<P_INV[ic] THEN BEST1=ic;
END;

IF BEST1 = 1 THEN BEST2=2; ELSE BEST2=1;

DO ic=2 to &CHR;
    IF ic NE Best1 Then
    IF P_INV[Best2]<P_INV[ic] THEN Best2=(ic);
    END;

Worst1=1;
DO ic=2 to &CHR;
    IF P_INV[Worst1]>P_INV[ic] THEN Worst1=ic;
    END;

IF Worst1 = 1 THEN Worst2=2; ELSE WORST2=1;
DO ic=2 to &CHR;
    IF ic NE Worst1 Then
    IF P_INV[Worst2]>P_INV[ic] THEN Worst2=(ic);
    END;
END;
ELSE DELETE;
RUN;
%MEND Worst_C;

%MACRO Cross0;
DATA W_C_CO;
    SET W_Chroms;
    ARRAY M[&CHR,&CL,&VAR];
    ARRAY TP_INV[&CHR];
    ARRAY INV[&CHR];
    ARRAY INVV[&CHR];
    ARRAY TP2_INV[&CHR];
    ARRAY NewC1[&CL,&VAR];
    ARRAY NewC2[&CL,&VAR];

```

```

IF count < &Quit THEN
DO;
*****Randomly Select 2 Chromosomes;
    RN1=RANUNI(0); *****Random a number between 0 and 1;
    DO ic=1 to &CHR;*****Find out the random number fall into
which chromosomes;
        IF TP_INV[&CHR-ic+1] > RN1 THEN RC1=(&CHR-ic+1);
    END;
    DO ic=1 to &CHR; *****Take out the value of the first selected
chromosome;
        IF ic = RC1 THEN INVV[ic]=0;
        ELSE INVV[ic]=INV[ic];
    END;
    *****ReWeight the Prob;
    T2_INV=INVV1+INVV2+INVV3+INVV4+INVV5+INVV6+INVV7+INVV8+INVV9+INVV
10;
    TP2_INV1=INVV1/T2_INV;
    DO ic=2 to &CHR;
        TP2_INV[ic]=TP2_INV[ic-1]+INVV[ic]/T2_INV;
    END;
    *****Select the second chromosome;
    RN2=RANUNI(0);
    DO ic=1 to &CHR;
        IF TP2_INV[&CHR-ic+1] > RN2 THEN RC2=(&CHR-ic+1);
    END;
    *****Randomize CrossOver Point;
    X_Randm=1+((&VAR)-1)*RANUNI(0);
    X_Rand=ROUND(X_Randm,1);
    CL_Randm=1+((&CL)-1)*RANUNI(0);
    CL_Rand=ROUND(CL_Randm,1);
    *****Start Crossover;
    IF CL_Rand > 1 THEN
    DO;
        DO iq=1 to (CL_Rand-1);
            DO ix=1 to &VAR;
                NewC1[iq,ix]=M[RC1,iq,ix];
                NewC2[iq,ix]=M[RC2,iq,ix];
            END;
        END;
    END;

    DO ix=1 to X_Rand;
        NewC1[CL_Rand,ix]=M[RC1,CL_Rand,ix];
        NewC2[CL_Rand,ix]=M[RC2,CL_Rand,ix];
    END;

    IF X_Rand < (&VAR+1) THEN
    DO;
        DO ix=(X_Rand+1) to &VAR;
            NewC1[CL_Rand,ix]=M[RC2,CL_Rand,ix];
            NewC2[CL_Rand,ix]=M[RC1,CL_Rand,ix];
        END;
    END;

    IF CL_Rand < (&CL+1) THEN
    DO;
        DO iq=(CL_Rand+1) to &CL;

```

```

        DO ix=1 to &VAR;
            NewC1[iq,ix]=M[RC2,iq,ix];
            NewC2[iq,ix]=M[RC1,iq,ix];
        END;
    END;
END;
*****Substitute Two Worst by Two New Chromosomes;
DO iq=1 to &CL;
    DO ix=1 to &VAR;
        M[WORST1,iq,ix]=NewC1[iq,ix];
        M[WORST2,iq,ix]=NewC2[iq,ix];
    END;
END;
END;
RUN;
%MEND CrossO;

%MACRO Mutat;
DATA CO_MU;
    SET W_C_CO;
    ARRAY M[&CHR,&CL,&VAR];
    ARRAY MuC[&NumMu];
    ARRAY MuQ[&NumMu];
    ARRAY MuX[&NumMu];
IF count < &Quit THEN
DO;
*****Start Mutation;
    Do i=1 to &NumMu;
        MuC[i]=1+(((&CHR)-1)*RANUNI(0));
        MuQ[i]=1+(((&CL)-1)*RANUNI(0));
        MuX[i]=1+(((&VAR)-1)*RANUNI(0));
        MC=ROUND (MuC[i],1);
        MQ=ROUND (MuQ[i],1);
        MX=ROUND (MuX[i],1);
        M[MC,MQ,MX]=10-M[MC,MQ,MX];
    END;
END;
RUN;
%MEND Mutat;

%MACRO Invers;
DATA Inversion;
    SET CO_MU;
    ARRAY M[&N_M];
    ARRAY SM[&N_M];
IF count < &Quit THEN
DO;
*****Randomly Select Two Points;
        INVP1=1+(&N_M-1)*RANUNI(0);
        INVP2=1+(&N_M-1)*RANUNI(0);
        INVP1=ROUND (INVP1,1);
        INVP2=ROUND (INVP2,1);
*****Figure out Start and End Point;
        DIFF=INVP1-INVP2;
        IF DIFF>0 THEN
            DO; INVP3=INVP1;
                INVP1=INVP2;

```

```

            INVP2=INVP3;
        END;
        NumInv=abs (INVP2-INVP1);
        MP_Inv=NumInv/2;
        MP_Inv=Round (Mp_Inv, 1);

        *****Swap Them;
        Do i=1 to MP_Inv-1;
            SM[INVP1+i-1]=M[INVP1+i-1];
            M[INVP1+i-1]=M[INVP2-i+1];
            M[INVP2-i+1]=SM[INVP1+i-1];
        END;
    END;
    RUN;
%MEND Invers;

%MACRO Replace;
DATA Seeds;
    SET Inversion;
    KEEP M1--M&N_M BEST1;
RUN;
%MEND Replace;

%MACRO GA_Seeds;
DATA SeedGA (KEEP=X1--X&NumMu);
    SET SEEDS;
    ARRAY M[&CHR, &CL, &VAR];
    ARRAY X[&CL, &VAR];
    DO iq=1 to &CL;
        DO ix=1 to &VAR;
            X[iq,ix]=M[Best1,iq,ix];
        END;
    END;
%MEND GA_Seeds;

%MACRO Ite;
PROC PRINT DATA=Inversion;
    TITLE "This is the &iteration iteration";
    VAR BEST1 BEST2 WORST1 WORST2;
RUN;
%MEND Ite;

%MACRO MyGA;
    %DO i=1 %to &Iteration;
        %Upd_Seeds
        %A_Clus
        %Worst_C
        %CrossO
        %Mutat
        %Invers
        %Replace;
    %END;
    %Ite
    %GA_Seeds
%MEND MyGA;
%MyGA
RUN;

```