Appendix B

ZBF Program

Program Description

Input:

- The generating unit system data which consists of the heat input-output curves, ramp rate limits, unit operating limits, fuel cost of each unit, initial power outputs at time t=0, forecast load demands, and transmission loss B-matrix coefficients.

Output:

- The result of the program consists of the power output of each generating units and their highest and lowest possible power output, the power loss, the generator fuel cost and the CPU time of each time period.

```
/*-------------------------------------------------------*/
/* Program : Zoom Brute Force for Constrained ED problem
/* Compile : gcc zbf.c -o ZBF -lm
/* Execute : ZBF
/*-------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "os.h"
#include "coops.c"
#define MaxUnit  10
#define MaxMember 5
#define MaxHour 500
#define INF pow(10,14)

FILE    *OUTP;
FILE    *Inp;
typedef unsigned short  Word;
char    MyFolder[10],Command[20],datafile[20],rsfile[40];
Word    piece[MaxUnit],Gen[MaxUnit],NUnit,Step,MaxStep,Member,LstIter;
int     Dimension[MaxUnit],P_flag[MaxUnit];
/* Cost Coefficient */
        double Fuel_Cost[MaxUnit],F[MaxUnit];
        double  P_int[MaxUnit][MaxMember];
        double  Ah[MaxUnit][MaxMember],Bh[MaxUnit][MaxMember],Ch[MaxUnit][MaxMember];
        double  aC[MaxUnit][MaxMember],bC[MaxUnit][MaxMember],cC[MaxUnit][MaxMember];
        double  BFCost,SumBestCost,Cost,BestCost;

/* B Matrix */
        double B00;
        double B0[MaxUnit];
        double B[MaxUnit][MaxUnit];
/* Based Power */
        double P_Base;

/* Power output variables */
        Word   ref;
        unsigned long combination;
        Word    i,j;
        double  P_pu[MaxUnit],P_Best[MaxUnit],P_ini[MaxUnit];
        double  Pmin[MaxUnit],Pmax[MaxUnit],UR[MaxUnit],DR[MaxUnit];
        double  Prmin[MaxUnit],Prmax[MaxUnit];
        double  Upperbound[MaxUnit],Lowerbound[MaxUnit];
        double  Pd[MaxHour];
        double  SumPrmin,SumPrmax,Demand_Actual;
        double  PLoss,SumPow,PowPlant,PLoss_Best,SumPow_Best,PowPlant_Best;
        int     Range;
        Word    *pt;
        double  *P,*P_BF;

/* Parameter -*/
        int     NumStepSize,Iter;
        double  delta_1,K_delta,StepSize;
/* Time */
        struct   timeval   start_time,stop_time,diff_time,SumTime;
        struct   timezone  timz;


void SetBound(void)
{  Word i;
   /*------------- Set Bound -------------*/
        for (i=0; i < NUnit; i++)
        {
                Lowerbound[i] = P_Best[i] - (StepSize*Dimension[i]);
                Upperbound[i] = P_Best[i] + (StepSize*Dimension[i]);
                if(Lowerbound[i]<=Prmin[i])Lowerbound[i] = Prmin[i];
                if(Upperbound[i]>=Prmax[i])Upperbound[i] = Prmax[i];
}}

void TitleReport(void)
{     Word i;
      fprintf(OUTP,"\nData: %s  Result: %s\n",datafile,rsfile);
      fprintf(OUTP,"\nStep_ini %lf K_delta %lf LstIter %d",delta_1,K_delta,LstIter);
      fprintf(OUTP,"\nStep\tPowDemand\tPowPlant\tP_Loss\tSumPow\t");
      for (i=0; i < NUnit; i++) fprintf(OUTP,"Prmax[%d]\t",i);
```

```
        for (i=0; i < NUnit; i++) fprintf(OUTP,"Prmin[%d]\t",i);
        for (i=0; i < NUnit; i++) fprintf(OUTP,"P[%d]\t",i);
        fprintf(OUTP,"ZBFCost");
        fprintf(OUTP,"\tTime(Sec)\tMicroSec\n");
}

void Report(void)
{  Word i;
   fprintf(OUTP,"\n%d",Step);
   fprintf(OUTP,"\t%lf\t%lf\t%lf\t%lf",Pd[Step],PowPlant_Best,PLoss_Best,SumPow_Best);
   for (i=0; i<NUnit; i++)fprintf(OUTP,"\t%lf",Prmax[i]);
   for (i=0; i<NUnit; i++)fprintf(OUTP,"\t%lf",Prmin[i]);
   for (i=0; i<NUnit; i++)fprintf(OUTP,"\t%lf",P_Best[i]);
   fprintf(OUTP,"\t%.4f",BestCost);
   fprintf(OUTP,"%.6d\t%.6d",diff_time.tv_sec,diff_time.tv_usec);
   fflush(OUTP);
}
void ShowScreen(void)
{  Word i;
   write_xy(1,1,"");
   printf("\nYou're running ZBF in case of %s",datafile);
   printf("\nPowerDemand = %10.4f Step %2d\n",Pd[Step],Step);
   for (i=0; i<NUnit; i++)
   {  if(i%8 == 0)printf("\n\n");
      printf("%3.4f  ",P_Best[i]);
   }
        printf("\nPLoss = %4.4f",PLoss_Best);
        printf("\nSumPow = %lf",SumPow_Best);
        printf("\tBestCost = %.4f",BestCost);
        printf("\nSumCost = %.4f",SumBestCost);
        printf("\nCPU time = %6i seconds %6I micro_seconds",diff_time.tv_sec,
         diff_time.tv_usec);
        printf("\nSumCPU time = %6i seconds %6i micro_seconds\n",SumTime.tv_sec,
         SumTime.tv_usec);
}

void SetGen(Word ref)
{  Word i;
        Gen[0] = ref;
   for(i=1;i<=NUnit-1;i++)
   {  Gen[i] = Gen[i-1]+1;
              if(Gen[i]==NUnit)Gen[i] = 0;
   }
}

double BF(Word *pt)
{
        P_flag[*pt]=0;
        if(pt!=&Gen[0])
        {       P[*pt] = Lowerbound[*pt];
                do{
                        BFCost=BF(--pt);
                        ++pt;
                        P[*pt]+=StepSize;
                        if(P[*pt]>=Upperbound[*pt]){P_flag[*pt]++; P[*pt] = Upperbound[*pt];}
                } while(P_flag[*pt]<2);
        }
        else
        {
                PLoss = B_Matrix1(*pt,Demand_Actual,P,&PowPlant,&SumPow);
                if (fabs(SumPow-Demand_Actual)<0.00001)
                {       Cost = CostCal(P);
                        /*---------- cost calculation -----------------*/
                        if(Cost<BFCost)
                        {       BFCost = Cost;
                                for (i=0; i<NUnit; i++)P_BF[i] = P[i];
        }}}
        return BFCost;
}

int main ()
{       /* Reserve memory for arrays of P and P_BF */
        if((P=(double *)malloc(sizeof(double)*MaxUnit))==NULL)
                {printf("\nMalloc P Error");getchar();exit(-1);}
        if((P_BF=(double *)malloc(sizeof(double)*MaxUnit))==NULL)
                {printf("\nMalloc P_BF Error");getchar();exit(-1);}
```

```
/* Input Filename with */
printf("\nData Filename (.txt)?"); scanf("%s",datafile);

/* Parameter Selection */
printf("\nInitial Step Size ?");scanf("%lf",&delta_1);
printf("\nK_delta (5)?");scanf("%lf",&K_delta);
printf("\nNumbers of Iteration ?");scanf("%d",&LstIter);
printf("\nInit Step %lf, K_delta %lf Iterations %d",delta_1,K_delta,LstIter);
printf("\nZBF run %s",datafile);


if((Inp     = fopen(datafile,"r"))==NULL){
     printf("cannot open datafile %s input\n",datafile);
     exit(0);
     }

ReadData();
BuildCoef();
CalDim();

/* Open Result File */
/* Make Folder Report */
     sprintf(MyFolder,"Result%i",NUnit);
     sprintf(Command,"mkdir %s",MyFolder);
     system(Command);
     sprintf(rsfile,"Result%d/ZBF%s",NUnit,datafile);
     if((OUTP = fopen(rsfile,"w"))==NULL){
            printf("cannot open file output\n");
            exit(0);
            }
TitleReport();
/*--------- end of openning the Result file ----------*/


/*------ Set P_Best, Reset the TotalBestCost and CPU Time -----*/
for (i=0; i < NUnit; i++)P_Best[i] = Pmin[i];
ClrScr();
SumBestCost = 0;
SumTime.tv_sec = 0; SumTime.tv_usec = 0;


for(Step = 0; Step < MaxStep; Step++)
{    Demand_Actual = Pd[Step];
     Start_Time();
     /*-------- clear bestcost and set ramp limit --------*/
     BestCost = INF;BFCost = INF;
     if(Step==0)for(i=0; i<NUnit; i++){Prmin[i] = Pmin[i];Prmax[i]=Pmax[i];}
     else SetRampLimit(P_Best,Prmax,Prmin,&SumPrmax,&SumPrmin);

     /*------------- Adjust Step Size -------------*/
     for(Iter = 0;Iter < LstIter;Iter++)
     {
        if(Iter==0)
        {    StepSize = delta_1;
             for(i=0; i<NUnit; i++){Lowerbound[i] = Prmin[i];Upperbound[i] = Prmax[i];}
        }
        else
        {    StepSize /= K_delta;
             SetBound();
        }

         ref = 0;
        for(ref = 0; ref<NUnit; ref++)
        {    SetGen(ref); pt = &Gen[NUnit-1];
             BFCost = BF(pt);
             if(BFCost<BestCost)
             {       BestCost = BFCost;
                     for(i=0; i<NUnit; i++)P_Best[i]=P_BF[i];
                     PLoss_Best = B_Matrix1(ref,Demand_Actual,P_Best,&PowPlant,&SumPow);
                     PowPlant_Best = PowPlant;
                     SumPow_Best = SumPow;
             }
        }
     }

     Stop_Time();
     ReadTime(start_time,stop_time,&diff_time);
```

```
        Sum_Time();
        Report();
        ShowScreen();
        SumBestCost += BestCost;
        printf("\nBestCost = %lf",BestCost);
    }
    printf("\nSumBestCost = %lf",SumBestCost);
    fprintf(OUTP,"\nSumBestCost = %lf",SumBestCost);
    fprintf(OUTP,"\nSumBestCost = %lf",SumBestCost/4);
    fprintf(OUTP,"\nSumTime %.6d\t%.6d",SumTime.tv_sec,SumTime.tv_usec);
    fclose(OUTP);
    return 0;
}
```