CPE 354 : Optimization Design and Evolutionary Computing

Topic : Diet and Nutrition Plan

by

Varattaya Rojanarachneekorn 57070503431

Ekkapon Lumsuay 57070503442

Kornklow Khantilapapan 57070503484

Send to

Aj. Nareumon Wattanapongsakorn

Objective

Our project is 'Diet and nutrition planning'. Which will help the user manage what they should consume for their meal in each day. The Diet and Nutrition Planning can solve by genetic algorithm and consider to this objectives :

- To help user find 6 food meals in 2 days (3 meals per day)
- With maximum nutrient but calories and cost in each day must not exceed than user input.

Model formulation with notation description

- **Objective Function**

MaxNutrient = carbohydrate+fat+protein

$\text{Max } N = \sum^{m}_{j=1} c_{ij}x_i + p_{ij}x_i + f_{ij}x_i$

Where;

N = nutrient should consume in 2 days

j = 1 to 6, number of constraints (number of meals in 2 day)

m = 32, number of food commodities

c_{ij} = content of the i^{th} carbohydrate in the j^{th} food

p_{ij} = content of the i^{th} protein in the j^{th} food

f_{ij} = content of the i^{th} fats in the j^{th} food

x_i = mass of the selected food commodities expressed in grams

- **Constrain**

$\text{Min } K = \sum^{m}_{j=1} kcal_j$

Where;

K = kcal constraint

$kcal_j$ = energy quantity per grams for the j^{th}

$\text{Min } C = \sum^{m}_{j=1} cost_j$

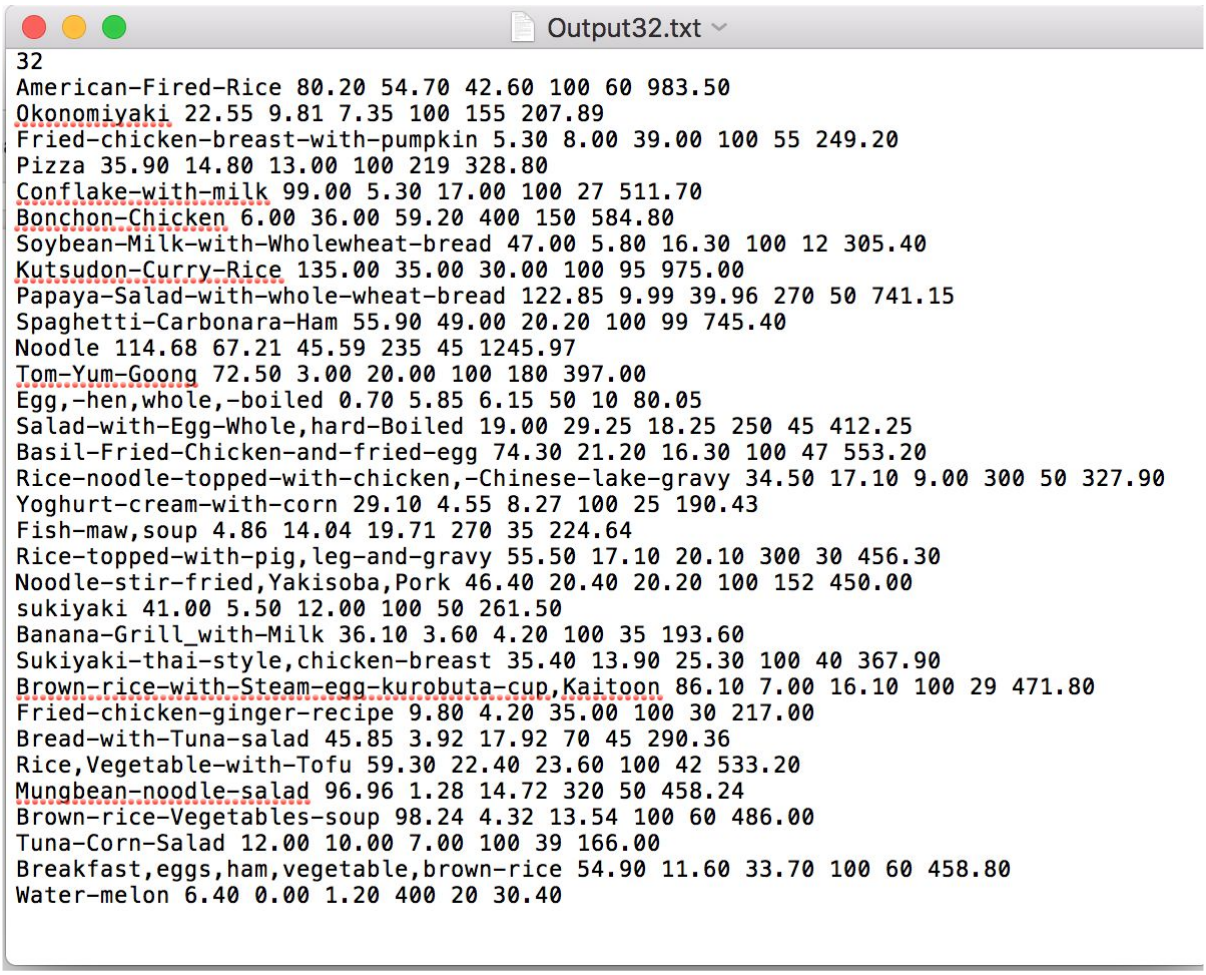
Where;

C = budget constraint

c_j = cost per gram of j^{th} food commodity

Input data

Our input file may look like figure below. The input file contain 32 menus data. In one line consist of food name, carbohydrate/gram, fat/gram, protein/gram, dish size, food cost and kcal of the food.



Problem Size

From input data file that we use there are 32 meals to select, but we need 6 meals for 2 days (3 meals per day) with repetition.

So we have:

(n+k-1)!/(n-1)!k!

Where; n = total menu

k = number of menu picked.

= (32+6-1)!/(32-1)!6!

= 2324784 ways to choose meal.

Algorithm and parameter setting

- **Algorithm**

We use Genetic Algorithm to solve our problem. First we generate randomly population in each generation then check which population are allowed to pass in their gene depend on range of fitness function and constraints.

Then, two individuals are chosen from the population to be parents for generate many different offspring by using crossover method.

from picture we can we that the point at which the chromosome is broken depends on the randomly selected crossover point.

Next is mutation, mutation works by choosing an element of the string at random and replacing it with another symbol (bit filpped) to avoid bad case.

After that repeat from the second step(pick parents) until we reach a termination condition.

Population Size

Population size is 100.

Parents Selection Technique

Our method that use for select parents is Elitism. After we generate randomly 100 populations for the first generation, the program will sort populations by fitness value in descending order. Then we will keep 10 individuals (how many individuals will keep is up to user's death rate input) that have best fitness for next generation. The first parents is randomly selected from 10 best fitness individuals and the second parents we randomly selected from current population by not the same individual in first parent which mean we give a change for low fitness individual to mate. The first and second parent then send to crossover function.

Crossover Type

We use single point crossover. In crossover function, it starts by random crossover point. All data beyond the crossover point in either chromosome string is swapped between the two parent chromosomes. The resulting chromosomes are the children. The child that meet constraints will transfer to next generation. Otherwise, we will reproduce children until it meet the constraints and reach the number of death individuals (if 90 individuals death, we need to produce 90 children)

Mutation Type

Occur by probability between 0-100percent up to user. When mutation is occur to that generation, the mutation operator takes the chosen individual and random inverts 1 bits in their chromosome (i.e. if the genome bit is 1, it is changed to 0 and vice versa).

- **Some essential function in the programs**

- void readFile(); // read input from text file
- void get_user_input(); // get input from user
- struct Chrom decode(chrom_t popcurrent); // decode from bit string to base 10
- double calFitness(chrom_t popcurrent); // calculate each individual fitness value
- int calCost(chrom_t popcurrent); // calculate each individual total cost
- double calCal(chrom_t popcurrent); // calculate each individual total calory
- void swopData(chrom_t * popData); // arrange population by fitness value in decending order
- void crossOver(); // for crossover, create new individuals
- void mutationData(chrom_t * popData) // select and mutate individual
- void deleteData(int numDelete, chrom_t * popData); // delete unessential data
- void resetpopTest(chrom_t popData); // initialize temporary variable
- void finalShow(chrom_t popData); // print the final result

- **Data Structure**

```
// creating the chromosome structure
typedef struct Chrom
{
        short int bit[30];          // keep bit string (chromosome)
        int bit6[6];                // keep the decimal decode from bit string
        double fitness;             // fitness value
        int allCost;                // total cost for 6 meals
        double allCal;              // total calory for 6 meals
}chrom_t;
```

```
// structure to keep input data
typedef struct Data
 {
        char name[64];          // food name
        double car;             // carbohydrate
        double fat;             // fat
        double pro;             // protein
        int dish_size;          // dize size in gram
        int cost;               // cost in bath unit
        double kcal;            // calory per dish
  }data_t;
```

- **Global variables in the programs**

```
- #define POP_SIZE 100                  // population size

- char fileName[]="Output32.txt";       // input file name
- data_t foods[32];                     // keep input data
- int inputCal;                         // keep user's calory input
- int inputMoney;                       // keep user's budget input
- int numGen;                           // number of iteration
- int perMutation;                      // mutate rate
- int numDel;                           // number of death rate
- chrom_t popCurrent[POP_SIZE];         // keep current population (before mate)
- chrom_t popNext[POP_SIZE];            // keep population for next generation
- chrom_t popTemp[POP_SIZE];            // tempory variable to keep valid children
                                           before pass it to next generation
- chrom_t popTest[2];                   // keep seleced parents
```

- **C Code**

```c
#include<stdio.h>   //to use the printf function
#include<conio.h>   //to use the getche function
#include<stdlib.h>  //to use the rand function
#include<time.h>
#include<math.h>

typedef struct Chrom            // creating the chromosome structure
    {
  short int bit[30];
  int bit6[6];
  double fitness; //all nutrient
  int allCost;
  double allCal;
  }chrom_t;

typedef struct Data
  {
    char name[64];
    double car; //carbohydrate
    double fat; //fat
    double pro; //protein
    int dish_size;  //dize size in gram
    int cost; //cost in bath unit
    double kcal; // callory per dish
  }data_t;
```

We create two structures to keep data. the first is ‘Chrom’, it keep data of 6 chromosomes those are binary 30 bits, 6 meal in 2 days, fitness value(all nutrient in 2 days), and all cost and calories in 2 days. The second structure ‘data_t’ keep data of input file name ‘Output32.txt’, the information that program read from that file will keep in this structure.

Output32.txt

```
32
American-Fired-Rice 80.20 54.70 42.60 100 60 983.50
Okonomiyaki 22.55 9.81 7.35 100 155 207.89
Fried-chicken-breast-with-pumpkin 5.30 8.00 39.00 100 55 249.20
Pizza 35.90 14.80 13.00 100 219 328.80
Conflake-with-milk 99.00 5.30 17.00 100 27 511.70
Bonchon-Chicken 6.00 36.00 59.20 400 150 584.80
Soybean-Milk-with-Wholewheat-bread 47.00 5.80 16.30 100 12 305.40
Kutsudon-Curry-Rice 135.00 35.00 30.00 100 95 975.00
Papaya-Salad-with-whole-wheat-bread 122.85 9.99 39.96 270 50 741.15
Spaghetti-Carbonara-Ham 55.90 49.00 20.20 100 99 745.40
Noodle 114.68 67.21 45.59 235 45 1245.97
Tom-Yum-Goong 72.50 3.00 20.00 100 180 397.00
Egg,-hen,whole,-boiled 0.70 5.85 6.15 50 10 80.05
Salad-with-Egg-Whole,hard-Boiled 19.00 29.25 18.25 250 45 412.25
Basil-Fried-Chicken-and-fried-egg 74.30 21.20 16.30 100 47 553.20
Rice-noodle-topped-with-chicken,-Chinese-lake-gravy 34.50 17.10 9.00 300 50 327.90
Yoghurt-cream-with-corn 29.10 4.55 8.27 100 25 190.43
Fish-maw,soup 4.86 14.04 19.71 270 35 224.64
Rice-topped-with-pig,leg-and-gravy 55.50 17.10 20.10 300 30 456.30
Noodle-stir-fried,Yakisoba,Pork 46.40 20.40 20.20 100 152 450.00
sukiyaki 41.00 5.50 12.00 100 50 261.50
Banana-Grill_with-Milk 36.10 3.60 4.20 100 35 193.60
Sukiyaki-thai-style,chicken-breast 35.40 13.90 25.30 100 40 367.90
Brown-rice-with-Steam-egg-kurobuta-cup,Kaitoon 86.10 7.00 16.10 100 29 471.80
Fried-chicken-ginger-recipe 9.80 4.20 35.00 100 30 217.00
Bread-with-Tuna-salad 45.85 3.92 17.92 70 45 290.36
Rice,Vegetable-with-Tofu 59.30 22.40 23.60 100 42 533.20
Mungbean-noodle-salad 96.96 1.28 14.72 320 50 458.24
Brown-rice-Vegetables-soup 98.24 4.32 13.54 100 60 486.00
Tuna-Corn-Salad 12.00 10.00 7.00 100 39 166.00
Breakfast,eggs,ham,vegetable,brown-rice 54.90 11.60 33.70 100 60 458.80
Water-melon 6.40 0.00 1.20 400 20 30.40
```

This is example of Output32.txt it keep any menus and nutrition and cost of each menu. From left to right is menu’s name, carbohydrate, fats, protein, and cost

```
26
27  char fileName[]="Output32.txt";
28  data_t foods[32];
29  int popNow=0;
30
31  //can fix
32  #define POP_SIZE 100
33  int inputCal=1500;
34  int inputMoney=600;
35  int numGen=1;
36  int perMutation=50;
37  int numDel=1;
38
39  chrom_t popCurrent[POP_SIZE];
40  chrom_t popNext[POP_SIZE];
41  chrom_t popTemp[POP_SIZE];
42  chrom_t popTest[2];
43
44  /*other function*/
45  void readFile();
46  void get_user_input();
47  struct Chrom decode(chrom_t popcurrent);
48  double calFitness(chrom_t popcurrent); //calAllCal
49  int calCost(chrom_t popcurrent);
50  double calCal(chrom_t popcurrent);
51  void showData(chrom_t * popData);
52  void swopData(chrom_t * popData);
53  void deleteData(int numDelete, chrom_t * popData);
54  void resetpopTest(chrom_t popData);
55  void finalShow(chrom_t popData);
56
```

Declare function and structure. And declare variable line 30-45 in global.

Main Function

```
56
57  void main()
58  {
59      int i,j,k;
60      int count=0;
61      int row=0;
62
63      srand(time(NULL));
64
65  //  Read File
66      readFile();
67  //  Get data from user
68      get_user_input();
69
70  //  M1. Create first generation
71  //  1.1 Random data -> by check with cost and cal
72  for(i=0;i<POP_SIZE;i++)
73  {
74      while(1)
75      {
76      //random 21 menu in 126 bit
77          for(j=0;j<30;j++)
78          {
79              popCurrent[i].bit[j] = rand()%2;
80              //popCurrent[i].bit21[j] = randomNum(0);
81          }
82
83      //decode 30 to 6 meal
84      popCurrent[i]=decode(popCurrent[i]);
85
86  //  1.2 Calculate fitness & allCost & allCal
87      popCurrent[i].fitness=calFitness(popCurrent[i]);
88      popCurrent[i].allCost=calCost(popCurrent[i]);
89      popCurrent[i].allCal=calCal(popCurrent[i]);
90
91      if(popCurrent[i].allCal<inputCal && popCurrent[i].allCost<inputMoney)
92          {
93          break;
94          }
95
96      }//end while
97  }//end for
98
99      //set pop in Now equal POP_SIZE
100     popNow=POP_SIZE;
101 //  1.3 Sort by fitnesss
102     swopData(&popCurrent);
```

In main function other function are used here. Main function start from read input data file using function readFile().

Then get calories and cost in each day from user to calculate all cost and all calories in 2 days to be constraint to select population by using get_user_input() function.

Create randomly binary number of 2 meals from input file and change binary number to be decimal number by using decodepopCurrent[i]) function.

Calculate nutrient, cost, and calories using calFitness, calCost, calCal. Check that value of calories and cost from user is higher than input file.

If only one of cost or calories from user are lower, set popNow = POP_SIZE then sort data that have fitness value from increase to decrease.

```
//  1.4 copy data to popNext
    for(i=0;i<POP_SIZE;i++)
    {
        popNext[i]=popCurrent[i];
    }


//  1.5 Cut 20 value that least in popCurrent
    deleteData(numDel,&popCurrent);
// LOOP INTERRATION
for(j=0;j<numGen;j++)
{
    printf("\nCOM : INTERATION %d ----------------------------------------\n",j+1);
    printf("COM : popNow is %d\n",popNow);

//  M2. Create NEW generation
//  2.1 Crossover -> Random to pick 2 different parent -> new child by check Cost and cal
    crossOver();  //keep crossdata in popTemp + Calculate new fitness and cost of Mutation + sort
    //copy popTemp to popNext
    count=0;
    for(i=popNow-1;i<POP_SIZE;i++)
    {
        popNext[i]=popTemp[count];
        count++;
    }
    popNow=POP_SIZE;
    swopData(&popNext);
    //Copy popNext to popCurrent
    for(i=0;i<POP_SIZE;i++)
    {
    popCurrent[i]=popNext[i];
    }
    printf("\nCOM : gerate new gen already.");

//  2.2 Mutation -> random popuration to has mutation -> if has, random row and position
    mutationData(&popCurrent);
```

Keep data into popNext. Remove last 40 population from popNext. Crossover bit. Crossover to get offspring and save into popTemp. Then copy population in popTemp to popNext. Check that how many population that we have in popCurrent. Sort data by fitness value. Copy population in popNext into popCurrent.

```
//  2.3 Sort data by fitness  from fit H-L
    swopData(&popCurrent);

    //copy data to popNext before cut
    for(i=0;i<POP_SIZE;i++)
    {
        popNext[i]=popCurrent[i];
    }

//  2.4 Cut some value that least in popCurrent
    if(row!=numGen-1)
    {
    deleteData(numDel,&popCurrent);
    }
    row++;
}//for interration

//show last result
    showData(&popCurrent);
//  SHOW DATA TO USER
    finalShow(popCurrent[0]);/**/
}//main
```

Sort data by fitness value. Copy data from popCurrent to popNext before remove last 40 data. Remove final 40 data from popCurrent. Show data from popCurrent to user. Print the best answer to user.

readFile function

```
/*-------------------------------------------------------------------------------------------------------------*/
/*OTHER FUNCTION*/
void readFile()
{
  FILE * fread=NULL;
  int countMenu1=0;
  printf("COM : Open file name [%s]\n", fileName);
  fread = fopen(fileName,"r");
  int i;

  if(fread==NULL)
    {
      printf("COM : Cannot open file <%s>\n",fileName);
      exit(0);
    }

  //can run
  fscanf(fread,"%d",&countMenu1);
  printf("COM : Total foods have %d menu for choose in this program\n",countMenu1);

  for(i=0;i<countMenu1;i++)
  {
    fscanf(fread,"%s %lf %lf %lf %d %d %lf",foods[i].name, &foods[i].car, &foods[i].fat, &foods[i].pro, &foods[i]
        .dish_size, &foods[i].cost, &foods[i].kcal);
    //printf("%d. %-40s %.2lf %.2lf %.2lf | %d | %d | %.2lf\n",i+1 ,foods[i].name, foods[i].car, foods[i].fat,
        foods[i].pro, foods[i].dish_size, foods[i].cost, foods[i].kcal);
  }
  fclose(fread);

  printf("COM : Read file already.\n");
}
```

This function use for read and keep data inside input file to struct name data_t.

get_user_input function

```
void get_user_input()
{
    int calory = 0;
    int expense = 0;
    int iteration=0;
    int percent=0;
    int del=0;
    char input[64];

    printf("Please input calorie in 2 day : ");
    fgets(input,sizeof(input),stdin);
    sscanf(input,"%d",&calory);

    inputCal = calory;

    printf("Please input cost in 2 day : ");
    fgets(input,sizeof(input),stdin);
    sscanf(input,"%d",&expense);

    inputMoney = expense;

    printf("Please input iteration : ");
    fgets(input,sizeof(input),stdin);
    sscanf(input,"%d",&iteration);
    numGen=iteration;

    printf("Please input percent in mutation [0-100] : ");
    fgets(input,sizeof(input),stdin);
    sscanf(input,"%d",&percent);
    perMutation=100-percent;

    printf("Please input delete [0-%d] : ",POP_SIZE);
    fgets(input,sizeof(input),stdin);
    sscanf(input,"%d",&del);
    numDel=del;

}
```

This function will ask user to input calories and cost in 2 day. Ask for input iteration, percent of mutation, and number that user want to delete from popCurrent.

Chrom decode function

```
228
229
230 struct Chrom decode(chrom_t popcurrent)
231 {
232     int i,k;
233     int j=0;
234     int value = 0;
235     chrom_t ex;
236
237     ex = popcurrent;
238     //loop to keep value in value[i]
239     for(i=0;i<6;i++)
240     {
241         j = j+5;
242         //printf("j = %d\n",j);
243         for(k=0;k<5;k++)
244         {
245             value +=  (popcurrent.bit[j-k-1]*(pow(2,k)));
246         }
247         value+=1;// chang to menu 1 - 32
248         ex.bit6[i] = value;
249         //printf("popcurrent.value[%d] =  %d\n",i,ex.bit6[i]);
250         value = 0;
251         //printf("\n\n");
252
253     }
254     return ex;
255 }
256
```

The decode function will decode binary number to be decimal number from 1-32. Decimal number will tell us that what menu is store in that number, but binary number use for crossover because binary number can create more different offspring than decimal number.

showData function

```
563
564 void showData(chrom_t * popData)
565 {
566   int  i,j,k=0;
567   for(i=0;i<POP_SIZE;i++)
568   {
569   printf("Random Menu %d : ",i+1);
570   for(j=0; j<6; j++)
571     {
572       printf(" %d",popData[i].bit6[j]);
573       k++;
574       if(k==3)
575       {
576         printf(" |");
577         k=0;
578       }
579       if(j==5)
580       {
581         printf("\n");
582       }
583     }
584   printf("Fitness %.2lf Cost %d Cal %.2lf\n",popData[i].fitness, popData[i].allCost, popData[i].allCal);
585   }
586
587 }
```

showData function print fitness value, all cost, all calories in 2 days.

calFitness function

```
256
257 double calFitness(chrom_t popcurrent) //calAllCal
258 {
259     int j=0;
260     double fitnessValue = 0;
261     double fatG =0,carG=0,proG=0;
262
263     for(j=0;j<6;j++)
264     {
265         carG += foods[popcurrent.bit6[j]-1].car;
266         fatG += foods[popcurrent.bit6[j]-1].fat;
267         proG += foods[popcurrent.bit6[j]-1].pro;
268     }
269     fitnessValue = carG + fatG + proG;
270     //printf(" fitValue = %.2lf",fitnessValue);
271      //printf(" totalC = %.2lf",fitC);
272      //printf("\n");
273
274     return fitnessValue;
275 }
```

This function use for calculate nutrient 6 menus by plus carbohydrate, fat, protein together.

calCost and calCal function

```
276
277 int calCost(chrom_t popcurrent)
278 {
279     int j=0;
280     int allCost=0;
281
282     for(j=0;j<6;j++)
283     {
284         allCost += foods[popcurrent.bit6[j]-1].cost;
285         //printf("%d .. %d \n",j+1, foods[popcurrent.bit21[j]-1].cost);
286     }
287 //  printf("allCost = %d\n",allCost);
288     return allCost;
289 }
290
291 double calCal(chrom_t popcurrent)
292 {
293     int j=0;
294     double allCal=0;
295
296     for(j=0;j<6;j++)
297     {
298         allCal += foods[popcurrent.bit6[j]-1].kcal;
299         //printf("%d .. %d \n",j+1, foods[popcurrent.bit6[j]-1].cost);
300     }
301 //  printf("allCal = %.2lf\n",allCal);
302     return allCal;
303 }
```

Calculate all cost and calories in 2 days

swopData function

```
376
377 void swopData(chrom_t * popData)
378 {
379     int i,j,k;
380     chrom_t temp;
381     for(i=0;i<POP_SIZE;i++)
382     {
383         for(j=0;j<POP_SIZE-1;j++)
384         {
385         //  printf("before fitness %d %d",popcurrent[j].fitness,popcurrent[j+1].fitness);
386             if(popData[j].fitness < popData[j+1].fitness)
387             {   //swop temp=popcurrent[j]
388                 for(k=0;k<126;k++)
389                 {
390                 temp.bit[k]=popData[j].bit[k];
391                 }
392                 for(k=0;k<21;k++)
393                 {
394                 temp.bit21[k]=popData[j].bit21[k];
395                 }
396                 temp.fitness=popData[j].fitness;
397                 temp.allCost=popData[j].allCost;
398                 temp.allCal=popData[j].allCal;
399                 for(k=0;k<126;k++) //swop popcurrent[j]=popcurrent[j+1]
400                 {
401                 popData[j].bit[k]=popData[j+1].bit[k];
402                 }
403                 for(k=0;k<21;k++)
404                 {
405                 popData[j].bit21[k]=popData[j+1].bit21[k];
406                 }
407                 popData[j].fitness=popData[j+1].fitness;
408                 popData[j].allCost=popData[j+1].allCost;
409                 popData[j].allCal=popData[j+1].allCal;
410                 for(k=0;k<126;k++)//swop popcurrent[j+1]=temp
411                 {
412                 popData[j+1].bit[k]=temp.bit[k];
413                 }
414                 for(k=0;k<21;k++)
415                 {
416                 popData[j+1].bit21[k]=temp.bit21[k];
417                 }
418                 popData[j+1].fitness=temp.fitness;
419                 popData[j+1].allCost=temp.allCost;
420                 popData[j+1].allCal=temp.allCal;
421             }
422         //  printf("After fitness %d %d\n\n",popcurrent[j].fitness,popcurrent[j+1].fitness);
423         }
424     }
425     printf("COM : Swop data already.\n");
426 }
```

This function use for sort fitness value from high to low value.

deleteData function

```
362
363 void deleteData(int numDelete, chrom_t * popData)
364 {
365     int i=0,k=0;
366     if(numDelete>POP_SIZE)
367     {
368         printf("Cannot delete data more than %d/%d\n",numDelete,popNow);
369     }
370     for(i=POP_SIZE-numDelete;i<POP_SIZE;i++)
371     {
372         for(k=0;k<30;k++)
373         {
374             popData[i].bit[k]=0;
375         }
376         for(k=0;k<6;k++)
377         {
378             popData[i].bit6[k]=0;
379         }
380         popData[i].fitness=0;
381         popData[i].allCost=0;
382         popData[i].allCal=0;
383     }
384     popNow-=numDelete;
385     printf("\nCOM : population after delete  is %d\n", popNow);
386 }
387
```

This function will delete bad data from population by remove follow the value in variable numDelete.

mutationData function

```
485 void mutationData(chrom_t * popData)
486 {
487     int random;
488     int i;
489     int row=0,col=0;
490     int bit6=0;
491
492     random = rand()%100;
493
494     if(random >= perMutation) // popbability 50%
495     {
496         row = rand()%POP_SIZE;
497         col = rand()%30;
498
499         bit6=col/5;
500         if(col%5>0)
501         {
502             bit6++;
503         }
504         printf("\nMutation Random menu[%d] bit is %d/126  in Meal %d/21\n", row+1, col+1, bit6);
505
506
507         if(popData[row].bit[col]==1)
508         {
509             popData[row].bit[col]=0;
510         }
511         else
512         {
513             popData[row].bit[col]=1;
514         }
515
516     //decode
517     popData[row]=decode(popData[row]);
518
519     //new calculate
520     popData[row].fitness=calFitness(popData[row]);
521     popData[row].allCost=calCost(popData[row]);
522     popData[row].allCal=calCal(popData[row]);
523
524     //swopdata
525     swopData(&popCurrent); //change it to pop next
526     //check if cal and cost if it not chang fitness to 0
527     if(popData[row].allCal>inputCal || popData[row].allCost>inputMoney)
528         {
529         popData[row].fitness=0;
530         printf("COM : Data change and the value is not correct in condition.");
531         }
532     }
533
534     else // no mutation
535     {
536         printf("\nCOM : NO Mutation in this generation.");
537     }
538
539     printf("COM : Mutation already.\n");
540 }
```

This function use for mutate gene in chromosome. We use binary number for mutate, first choose chromosome and which bit in that chromosome that we want to flip. If that bit is 1, it going to change to be 0. On the other hand if it is 0 ,it have to change to be 1.

crossOver Function

```
387
388 void crossOver()
389 {
390     int random1,random2;
391     int crosspoint;
392     int i,j;
393     int countTemp=0;
394 //popCurrent popNext popTemp
395     //C. create data to popNext
396     while(1)
397     {
398         //C.1 random differrent parent
399         while(1)
400         {
401             random1=rand()%popNow; //popNow from popCurrent that still alive
402             random2=rand()%POP_SIZE;
403
404             if(random1 != random2)
405             {
406                 break;
407             }
408         }
409         //C.2 random crosspoint
410         crosspoint = ((rand()%30)+1);
411         //C.3 crossover
412         //crossing the bits before the cross point index
413         for(i=0;i<crosspoint;i++)
414         {
415             popTest[0].bit[i]=popNext[random1].bit[i];
416             popTest[1].bit[i]=popNext[random2].bit[i];
417         }
418         //crossing the bits after the cross point index
419         for(i=crosspoint;i<30;i++)
420         {
421             popTest[1].bit[i]=popNext[random1].bit[i];
422             popTest[0].bit[i]=popNext[random2].bit[i];
423         }
424         //C.4 decode
425         popTest[0]=decode(popTest[0]);
426         popTest[1]=decode(popTest[1]);
427
428         //C.5 calculate fitness + cal + cost
429         popTest[0].fitness=calFitness(popTest[0]);
430         popTest[0].allCost=calCost(popTest[0]);
431         popTest[0].allCal=calCal(popTest[0]);
```

```
432
433         popTest[1].fitness=calFitness(popTest[1]);
434         popTest[1].allCost=calCost(popTest[1]);
435         popTest[1].allCal=calCal(popTest[1]);
436         //C.6 check if it more cal and cost than input go to new selecting
437         //check countTemp
438         if(popTest[0].allCost<inputMoney && popTest[0].allCal<inputCal)
439         {
440         popTemp[countTemp]=popTest[0];
441         resetpopTest(popTest[0]);
442         countTemp++;
443         }
444         if(countTemp==POP_SIZE)
445             {
446                 break;
447             }
448         //check countTemp+1
449         if(popTest[1].allCost<inputMoney && popTest[1].allCal<inputCal)
450         {
451         popTemp[countTemp]=popTest[1];
452         resetpopTest(popTest[1]);
453         countTemp++;
454         }
455         //C.7 else it ok go out from infinity loop while
456         if(countTemp==POP_SIZE)
457             {
458                 break;
459             }
460     }//while
461     //after crossover
462     swopData(&popTemp);
463
464     printf("\nCOM : CrossOver in popTemp Already.\n");
465 }
```

In this function, the program is going to crossover between parents that were random. First the program find crossover point to crossing the 30 bits. Then cross to create 100 offsring and keep them in popTemp. After crossover the program will decode binary number that we got after crossing to decimal number then calculate new fitness value, new cost, and new calories from new population that were crossed. When we got new values from crossover we have to check that the values are not over than constraint. If they can pass condition, we have to sort data by fitness value from increasing to decreasing.

restpopTest function

```
466
467  void resetpopTest(chrom_t popData)
468  {
469      int i=0,k=0;
470
471      for(k=0;k<30;k++)
472          {
473          popData.bit[k]=0;
474          }
475      for(k=0;k<6;k++)
476          {
477          popData.bit6[k]=0;
478          }
479      popData.fitness=0;
480      popData.allCost=0;
481      popData.allCal=0;
482
483  }
```

restpopTest will reset value in popData to get the new popData.

finalShow

```
541  void finalShow(chrom_t popData)
542  {
543      int i,k=1;
544      int day=1;
545      printf("\n* ------------------------The best random for You---------------------------- *\n\n");
546
547      for(i=0;i<6;i++)
548      {
549          printf("DAY %d: Meal %d - > %-40s ",day, k, foods[popData.bit6[i]-1].name);
550          printf(" %-10.2lf kcal\n",foods[popData.bit6[i]].kcal);
551          k++;
552          if(k==4)
553          {
554          k=1;
555          day++;
556          printf("\n");
557          }
558
559      }
560      printf("\nMax fitness is %.2lf points\n",popData.fitness);
561      printf("Total callery is %.2f/%d kcal\n",popData.allCal,inputCal);
562      printf("Total cost is %d/%d bath.\n\n",popData.allCost,inputMoney);
563      printf("COM : Thank you");
564  }
565
```

Display final answer

Experimental results

- **Experiment Result from GA**

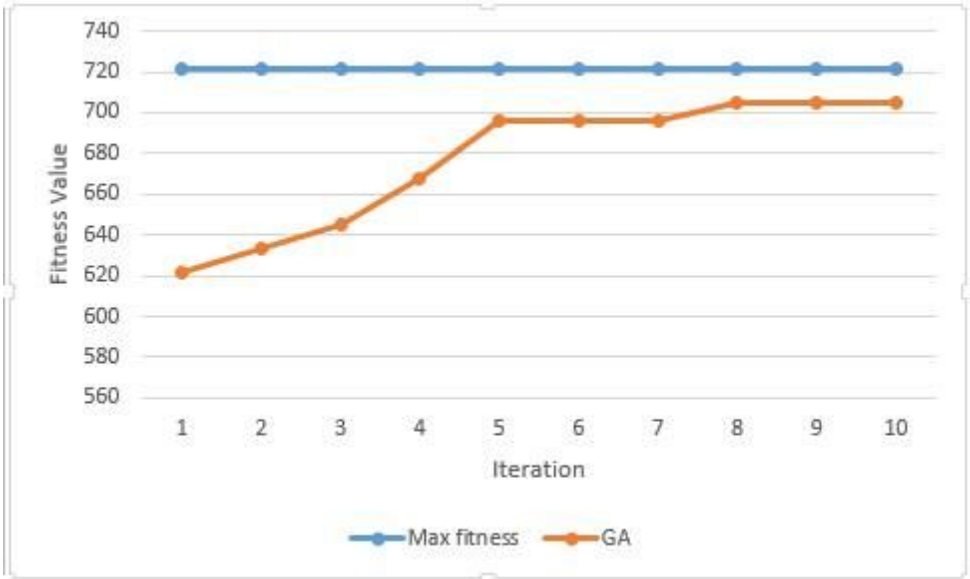
parameter:
- populations = 100
- generations = 20
- mutate rate = 50%
- death rate = 50

with constraints:
- input calory = 1500 kcal
- input budget = 300 baht

#	Fitness	Menu		Calories	Cost	Time (sec)
		Day1	Day2			
1	706.66	[9, 27, 8]	[5, 24, 5]	2999.99	195	20.367
2	712.15	[5, 29, 9]	[28, 26, 5]	2999.15	259	9.646
3	698.40	[5, 29, 5]	[24, 24, 5]	2964.70	199	7.939
4	706.10	[5, 29, 23]	[9, 29, 5]	2998.05	274	7.978
5	705.25	[5, 28, 24]	[5, 26, 9]	2984.95	228	9.846

(time count by using code block excution time)

Discussion

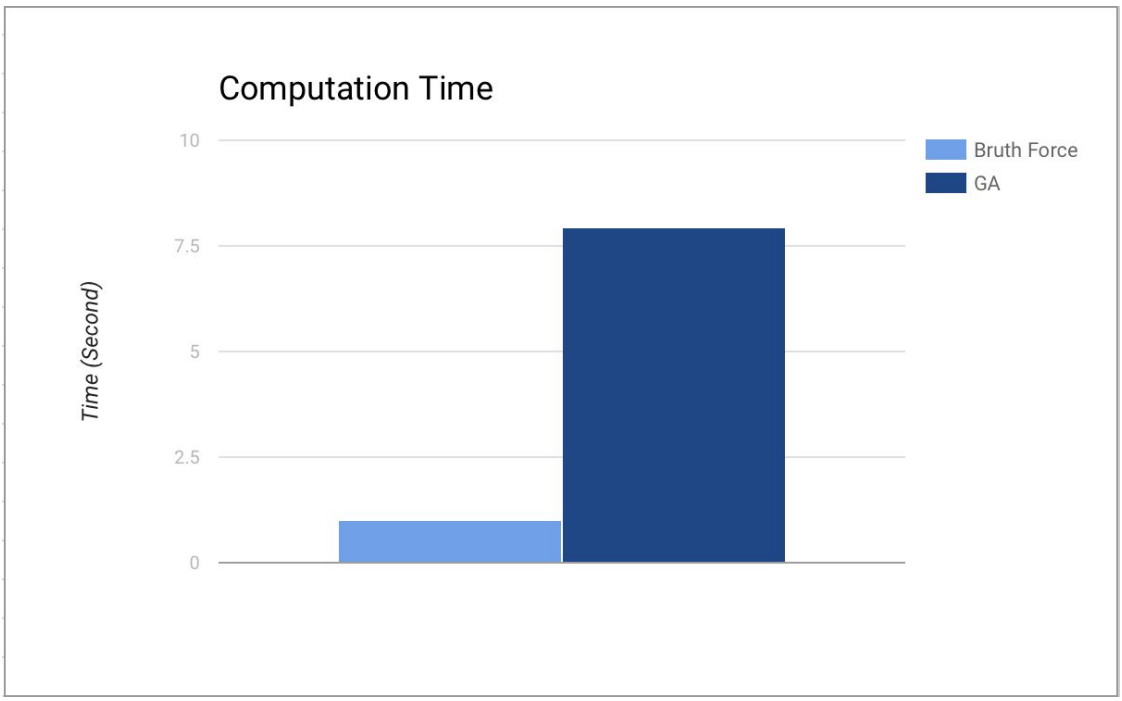


parameter:
- populations = 100
- generations = 10
- mutate rate = 1%
- death rate = 90

with constraints:
- input calory = 1500 kcal
- input budget = 300 baht

From the graph above, we can see that the fitness value in each iteration grow higher or equal to the last fitness value in last iteration. The reason why the result are higher is because we use ‘Elitism’ method to slove our problem. This strategy allow the best individual(s) from the current generation to carry over to the next which help to guarantee that that the population in the next generation won’t have bad fitness than the current generation. Even thought we still give a chance to the poor fitness value individuals to be parrent. This may help increase a chance to keep the good quality of poor fitness individuals to pass to next generation.

- **Computation time**



Normally, bruth force must take longer computation time than GA but we got result like this beacuse in our GA program have many ‘printf()’ so it waste time and make it gain the execution time.

Conclusion

The ploblem of Diet and Nutrition Planning can slove by using Genetic Algorithm. From the experimental result we can see that the output that we get is near to the optimum point and it not over than the result from brute force. Bruth force use a few time to show result but GA use the time more than bruth force because GA have to spend the time to print and display result.

Reference

http://cstheory.stackexchange.com/questions/14758/tournament-selection-in-genetic-algorithms

http://www.theprojectspot.com/tutorial-post/creating-a-genetic-algorithm-for-beginners/3

http://www.studystreet.com/c-program-sort-array-ascending-order/

https://en.wikipedia.org/wiki/Fitness_proportionate_selection

http://stackoverflow.com/questions/1276256/efficient-implementation-of-fitness-proportionate-roulette-selection

http://stackoverflow.com/questions/24609131/implementation-of-roulette-wheel-selection