Subject Code : CSCI964
Student Name : Yixiang Fan
Student Number : 5083898
Assignment Number : 2

Part 1 :

1. This part is to train 3 classifiers according to 3 cases. 1 file("1" is the name of this file) refers to 1-SpiralData1.txt. 2 refers to data2.txt. 3 refers to data3.txt. All data file have the standard format :
<label> <index1>:<value1> <index2>:<value2> ...
As a result, I modified the mlp.cpp in assignment 1 to generate the new data file for this assignment. In 1, there are 192 data. In 2, there are 4177 data. In 3, there are 349 data. Then I use check.py to check their format. After that, I use svm-scale to normalize them and use subset.py to subset them. 60% of 192 is around 115, so 1train file has 115 data for training and 1test file has 77 data for testing. By sampling from 2.scale generated by scaling 2, there 3677 data in 2train for training and 500 data in 2test for testing. 3train contains 299 data for training and 3test contains 50 data for testing.
All commands will be attached in the end of this part.

2. In case 1 and case 3, they are classification problems, so I adopt classifier 0(C-SVC : multi-class classification). I have tried classifier 1, but there is not much improvement. As to case 2, I adopted classifier 3(epsilon-SVR : regression) because it is a regression problem. I am not sure whether the data in case 1 and case 3 can be linear classified, so I adopt the radial basis function. Although it is not a linear kernel , but the specification says the linear function is a special situation of the radial basis function.

3. After scaling and sampling, I use grid.py to find the best C and gama. I change the range of log2c and log2g each time to find the optimal parameters.
For example, in round 1, I execute :
python grid.py -log2c -100,-50,1 -log2g -100,-50,1 -s 0 -v 5 -m 300 1train.
Then in round 2, I execute :
python grid.py -log2c -50,0,1 -log2g -50,0,1 -s 0 -v 5 -m 300 1train.
...
In the end , I will try the parameters by the following command :
./svm-train -s 0 -c 2 -g 64 1train 1.model
./svm-predict 1test 1.model 1.predict

4.

Case 1	74.026%
Case 2	Mean squared error = 0.0270348 (regression) Squared correlation coefficient = 0.292139 (regression)
Case 3	Accuracy = 90% (45/50) (classification)

5. In this part, I found that python is more efficient and more convenient on solving machine learning problems. It is much easier to train a decent classifier than C++. In ass1, the models I trained performed nealy 75% correct rate in case1 and less than 60% correct rate in case 2

and case 3. However, in this assignment, the models performed much better.
In other aspects, MLP model is sensitive to the number of layers and the numbers of neurons in each layer. Compared to MLP, SVM has less parameters. It only has C and gama. Of course, in the training procedure, some other parameters are relative as well, but they can be determined by the property of the problem, such as the type of the kernel function.

6. Command

```
================================================================================
========
python checkdata.py 1
./svm-scale -l -1 -u 1 -s range1 1 > 1.scale
python subset.py 1.scale 115 1train 1test
python grid.py -log2c 0,10,1 -log2g 50,70,1 -s 0 -v 5 -m 300 1train
# c = 2, g = 64

./svm-train -s 0 -c 2 -g 64 1train 1.model
# t = 0 ~ 3 , no difference

./svm-predict 1test 1.model 1.predict
74.026%

================================================================================
=======
python checkdata.py 2
./svm-scale -l -1 -u 1 -s range1 2 > 2.scale
python subset.py 2.scale 3677 2train 2test
python grid.py -log2c 0,10,1 -log2g 0,10,1 -s 0 -v 5 -m 300 2train
# c = 1024, g = 4

./svm-train -s 3 -c 1024 -g 4 2train 2.model
#optimization finished, #iter = 2444024
#nu = 0.151509
#obj = -28631.419538, rho = -0.153321
#nSV = 1358, nBSV = 314

./svm-predict 2test 2.model 2.predict
#Mean squared error = 0.0270348 (regression)
#Squared correlation coefficient = 0.292139 (regression)

================================================================================
=======
python checkdata.py 3
./svm-scale -l -1 -u 1 -s range1 3 > 3.scale
python subset.py 3.scale 299 3train 3test
python grid.py -log2c 0,10,1 -log2g 0,10,1 -s 0 -v 5 -m 300 3train
# c = 1, g = 1

./svm-train -s 0 -c 1 -g 1 3train 3.model
#optimization finished, #iter = 334
#nu = 0.415653
#obj = -75.114993, rho = -0.906771
#nSV = 244, nBSV = 63
#Total nSV = 244

./svm-predict 3test 3.model 3.predict
#Accuracy = 90% (45/50) (classification)
```

Part 2 :

Step 1:

The part of reading file into the program is located from line 63 to line 84 in my code. Then I randomly generate the current population with no repetition which locates from line 180 to line 189 in function `InitPop()`.

In Crossover(), I replaced the repetitive cities with the cities that are not involved from line 294 to line 351. I find all the cities that appear twice in the child and those are not involved. For instance, city 5 appears twice at index 15 and index 25. City 9 is not involved. So I will put city 9 at index 25. Then both cities will appear once. Apparently, as there are only two parents, so the maximum repetitive times of cities in a child is 2 and the number of repetitive city equals to the number of missing city.

In Mutate(), I think the original mutation rate is a little low. So I modify the mutation rate to 2% of the cities. If there are 500 cities, then each round 500 * 2% = 10 cities will mutate.

In EvaluateFitness(), the tour which has the shortest distance has the best fitness. A distance table is initialized at the beginning of the program to facilitate the calculation.

Step 2:

This step has been implemented in step 1.

Step 3:

I create a function `Roulette() to implement roulette wheel from line 229. The possibility of selecting a tour is proportional to the fitness of the tour. At first, subtract the worst fitness which is largest distance from each fitness and store in rFitness. Then normalize the rFitness. The shorter the distance, the larger the proportion.`

Step 4:

I have tried each `Xover on the same parameters and found that the eTwoPoint is the best model for all three cases. Unfortunately, I forgot to record the fitness of every 5th or 10th or 20th generation. I will put the these records of the modified parameter version in Step 5.`

The best tour of 100 cities :

54202

Best Individual:

794194546506651262164013673212118920909144417602230376169550725845433976332453843737535152782467083556881473471285955683110938192421426933669778063259648195299648887217885774235786984929 1897

```
const Xover    CrossoverType = eTwoPoint;
const double cCrossoverRate = 0.95;
const double cMutationRate = 0.9;
const int       cNumGens = 15000;
const int       cPopSize = 100; // must be an even number
const int       cTournamentSize = 5;
const int       Seed = 1234;   //I replace Seed with time
const int       cTargetFitness = 10000; //desired distance of tour
int parSel = 0; //parents selection : 0 - Tournament ; 1 - Roulette
```

The best tour of 200 cities :

```
174703
Best Individual:
59501818519077901044211011801743019518812582187945122178403114229166109986
91696543271321341635796105129738186130196192198107413318646112168157142213
67112012819932971585663851241023411620131135933317141123156021191140189121
70108441431451261514910014414817913117531817416019687816241155283616113159
17523613738115613719415338411915211116747075701031836415615095791642881767
21973917758976621821659199121166127528173172118941385114613910183671074810
6351541491611145417155261808718458147922419325
```

```
const Xover    CrossoverType = eTwoPoint;
const double cCrossoverRate = 0.95;
const double cMutationRate = 0.9;
const int       cNumGens = 15000;
const int       cPopSize = 100; // must be an even number
const int       cTournamentSize = 10;
const int       Seed = 1234;   //I replace Seed with time
const int       cTargetFitness = 10000; //desired distance of tour
int parSel = 0; //parents selection : 0 - Tournament ; 1 - Roulette
```

The best tour of 500 cities :

```
758810
Best Individual:
34163367336316494310772563514273141072792433184162983991232408511147661216
38540334444322944494783657003743231182812939215624119912337345259356338169
38129945220343820828935920049513832333635034432132247087242484304092304803
93703581491765320485261388674314183497254379302117257353712445890687515049
11539614831517043726266563351761353783834052505121841835231308340153122728
71989291211492233188159195226124419317332146114166426712762062472141051961
40329182445380246253106465306133193220172361802232852354122178101255178128
59174471674871022319214163821399745130730538481236194319277121168994218547
23981752524681861571614994824592834112711861086026836629210345241495425165
53212543841140027439047727215171098369155171357792582731154892601893544714
52979146410411926245572145881224881619141022465717911046743164112125201032
15448376281201222282483205444158136311126102393280406402231229227496190267
45414423222448484270209177152371626226447525252752134531207426389154732659
42251323498814133139249310030914118295364147304443642374422071321432383553
62221350184245304393534473463254662421344939613039428430332434942255789329
24623754138229618732742821228113300360180301343131286366202368269498423469
```

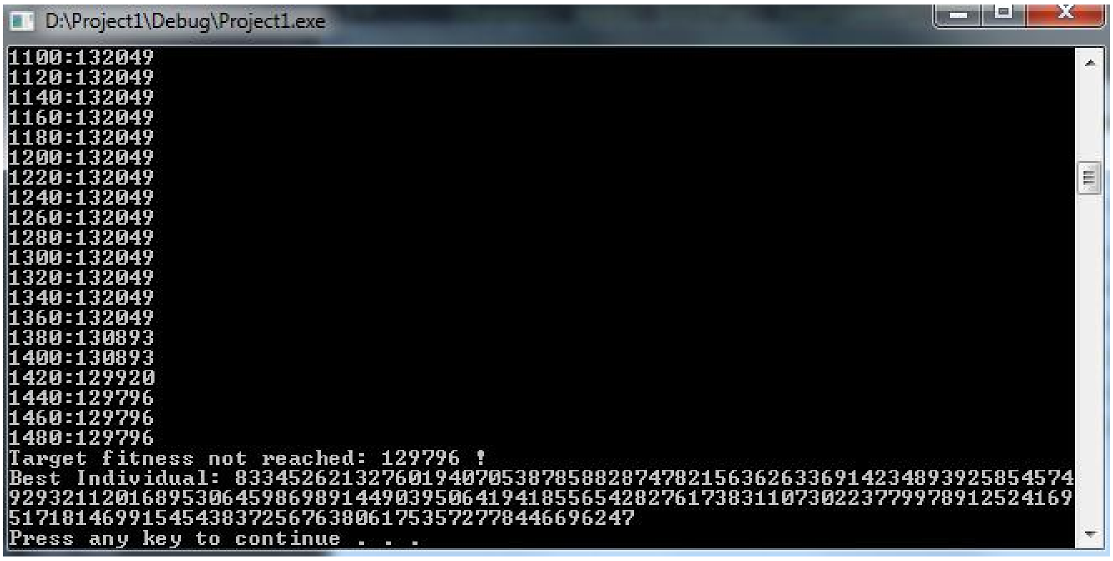
4464154014173392191604361733314742883971373697612742932637333440456391473
34820434738942139314247944145042419748640343429011637740838743366461490463
1939538632840442045746040745818137227829334224925133043584

```
const Xover     CrossoverType = eTwoPoint;
const double cCrossoverRate = 0.95;
const double cMutationRate = 0.5;
const int        cNumGens = 1500;
const int        cPopSize = 100; // must be an even number
const int        cTournamentSize = 25;
const int        Seed = 1234;    //I replace Seed with time
const int        cTargetFitness = 10000; //desired distance of tour
int    cIndividualLength = 80;
int parSel = 0; //parents selection : 0 - Tournament ; 1 - Roulette
```

Step 5:

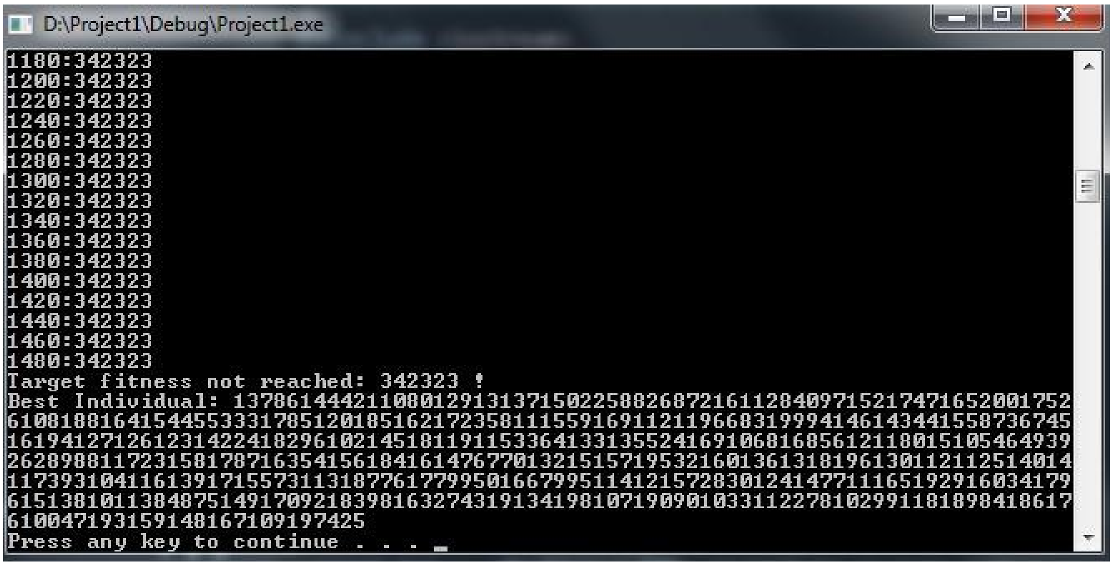
I change the cCrossoverRate to 0.1 and cMutationRate to 0.01, so the children are highly likely to be similar with their parents, which means the evolution is very slow. The following are the results :

100 cities :



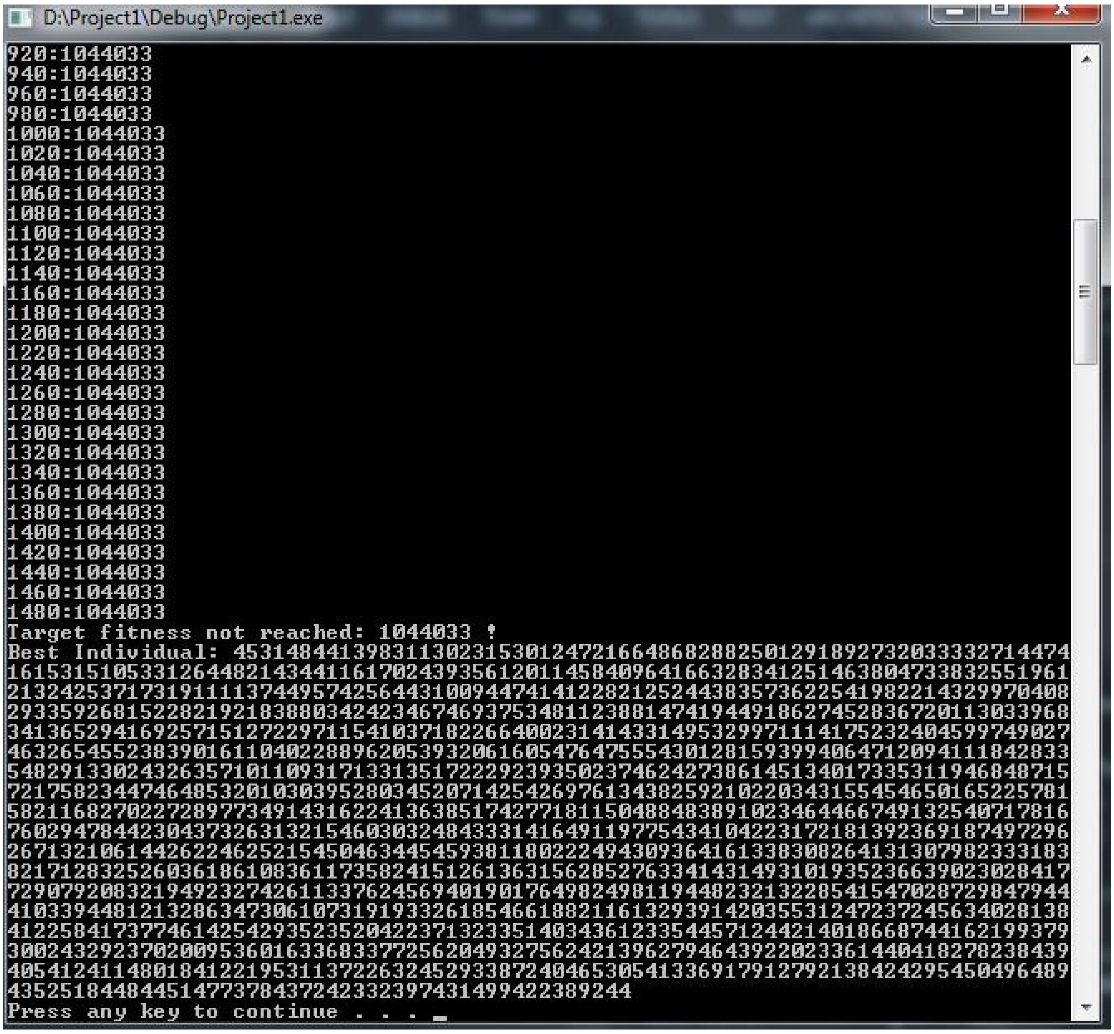
```
1100:132049
1120:132049
1140:132049
1160:132049
1180:132049
1200:132049
1220:132049
1240:132049
1260:132049
1280:132049
1300:132049
1320:132049
1340:132049
1360:132049
1380:130893
1400:130893
1420:129920
1440:129796
1460:129796
1480:129796
Target fitness not reached: 129796 !
Best Individual: 833452621327601940705387858828747821563626336914234893925854574
92932112016895306459869891449039506419418556542827617383110730223779978912524169
51718146991545438372567638061753572778446696247
Press any key to continue . . .
```

200 cities :



```
1180:342323
1200:342323
1220:342323
1240:342323
1260:342323
1280:342323
1300:342323
1320:342323
1340:342323
1360:342323
1380:342323
1400:342323
1420:342323
1440:342323
1460:342323
1480:342323
Target fitness not reached: 342323 !
Best Individual: 137861444211080129131371502258826872161128409715217471652001752
61081881641544553331785120185162172358111559169112119668319994146143441558736745
16194127126123142241829610214518119115336413313552416910681685612118015105464939
26289881172315817871635415618416147677013215157195321601361318196130112112514014
11739310411613917155731131877617799501667995114121572830124147711165192916034179
61513810113848751491709218398163274319134198107190901033112278102991181898418617
610047193159148167109197425
Press any key to continue . . . _
```

500 cities :



```
920:1044033
940:1044033
960:1044033
980:1044033
1000:1044033
1020:1044033
1040:1044033
1060:1044033
1080:1044033
1100:1044033
1120:1044033
1140:1044033
1160:1044033
1180:1044033
1200:1044033
1220:1044033
1240:1044033
1260:1044033
1280:1044033
1300:1044033
1320:1044033
1340:1044033
1360:1044033
1380:1044033
1400:1044033
1420:1044033
1440:1044033
1460:1044033
1480:1044033
Target fitness not reached: 1044033 !
Best Individual: 453148441398311302315301247216648682882501291892732033332714474
16153151053312644821434411617024393561201145840964166328341251463804733832551961
21324253717319111137449574256443100944741412282125244383573622541982214329970408
29335926815228219218388034242346746937534811238814741944918627452836720113033968
34136529416925715127229711541037182266400231414331495329971114175232404599749027
46326545523839016110402288962053932061605476475554301281593994064712094111842833
54829133024326357101109317133135172229239350237462427386145134017335311946848715
72175823447464853201030395280345207142542697613438259210220343155454650165225781
58211682702272897734914316224136385174277181150488483891023464466749132540717816
76029478442304373263132154603032484333141649119775434104223172181392369187497296
26713210614426224625215450463445459381180222494309364161338308264131307982333183
82171283252603618610836117358241512613631562852763341431493101935236639023028417
72907920832194923274261133762456940190176498249811944823213228541547028729847944
41033944812132863473061073191933261854661882116132939142035531247237245634028138
41225841737746142542935235204223713233514034361233544571244214018668744162199379
30024329237020095360163368337725620493275624213962794643922023361440418278238439
40541241148018412219531137226324529338724046530541336917912792138424295450496489
435251844844514773784372423323974314994223892 44
Press any key to continue . . . _
```

From these results, I can conclude two points.

1. The best tour in each generation is highly likely the same.
2. It is not easy to evolve to a better child.

```
/**********************************************************
*        ga.cpp - GA Program for CSCI964 - Ass2
*        Written by: Koren Ward May 2010
*        Modified by: Yixiang Fan
*        Changes: main; InitPop; Tournament; Roulette; EvaluateFitness; Crossover; Mutate. I
modified these functions.
**********************************************************/
#include <iostream>
#include <fstream>
#include <iomanip>
#include <cmath>
#include <cstdlib>
#include <cstdio>
#include <vector>
using namespace std;

const int cDebug = 0;

enum Xover { eRandom, eUniform, eOnePoint, eTwoPoint };

const Xover   CrossoverType = eTwoPoint;
const double cCrossoverRate = 0.95;
const double cMutationRate = 0.9;
const int     cNumGens = 15000;
const int     cPopSize = 100; // must be an even number
const int     cTournamentSize = 5;
const int     Seed = 1234;   //I replace Seed with time
const int     cTargetFitness = 10000; //desired distance of tour
int   cIndividualLength = 80;
int parSel = 0; //parents selection : 0 - Tournament ; 1 - Roulette
int *longitude = NULL;
int *latitude = NULL;
int *cityType = NULL;
int *co = NULL; //uniq in crossover - the index of co is the city number
vector<int> co0;
vector<int> co2;
double weightTable[3][3] = {10, 7.5, 5,
                            7.5, 5, 2.5,
                            5, 2.5, 1};
int distanceTable[1000][1000];

void InitPop(int ***CrntPop, int ***NextPop, int **Fitness, int **BestMenber, double
**rFitness);
void FreeMem(int **CrntPop, int **NextPop, int *Fitness, int *BestMember);
```

```
int Tournament(int *Fitness, int TournamentSize);
int Roulette(double *Fitness);
int EvaluateFitness(int *Member);
void Crossover(int *P1, int *P2, int *C1, int *C2);
void Copy(int *P1, int *P2, int *C1, int *C2);
void Mutate(int *Member);
double Rand01();        // 0..1
int RandInt(int n); // 0..n-1

int main(int argc, char *argv[]) {

    int **CrntPop, **NextPop; // the crnt & next population lives here
    // The possible longest distance between two city [0,999] - [999,0]. The distance is 1412.80.
    int *Fitness, BestFitness = 15000000, *BestMember; // fitness vars
    double *rFitness;
    int i, TargetReached = false;
    char fileName[100];

    ifstream inFile;
    if(argc == 1){
        cout << "Please input file : ";
        cin >> fileName;
        inFile.open(fileName);
    }else{
        inFile.open(argv[1]);
    }
    if (!inFile) {
        cerr << "Unable to open file datafile.txt"; exit(1);
    }

    inFile >> cIndividualLength;
    longitude = new int[cIndividualLength];
    latitude = new int[cIndividualLength];
    cityType = new int[cIndividualLength];
    co = new int[cIndividualLength];

    for(int i = 0; i < cIndividualLength; i++){
        inFile >> longitude[i] >> latitude[i] >> cityType[i];
    }
    inFile.close();

    //initiate the distanceTable
    for(int i = 0; i < cIndividualLength; i++){
        for(int j = 0; j < cIndividualLength; j++){
                int x = longitude[i] - longitude[j];
                int y = latitude[i] - latitude[j];
                double w = weightTable[cityType[i] - 1][cityType[j] - 1];
                distanceTable[i][j] = sqrt(x*x + y*y) * w;
        }
}

InitPop(&CrntPop, &NextPop, &Fitness, &BestMember, &rFitness);
for (int Gen = 0; Gen<cNumGens; Gen++) {
        for (i = 0; i<cPopSize; i++) {
                // Evaluate the fitness of pop members
                Fitness[i] = EvaluateFitness(CrntPop[i]);
                if (BestFitness > Fitness[i]) { // save best member
                        BestFitness = Fitness[i];
                        for (int j = 0; j<cIndividualLength; j++)
                                BestMember[j] = CrntPop[i][j];
                        if (Fitness[i] <= cTargetFitness) {
                                TargetReached = true;
                                break;
                        }
                }
        }
        if (TargetReached)break;

        //Calculate the Roulette wheel for each tour
        double WorstFitness = -1;
        long sumRFitness = 0;
        for (i = 0; i<cPopSize; i++) {
                if (WorstFitness < Fitness[i])
                        WorstFitness = Fitness[i];
                rFitness[i] = Fitness[i];
        }
        for (i = 0; i<cPopSize; i++) {
                rFitness[i] -= WorstFitness;
                sumRFitness += rFitness[i];
        }
        for (i = 0; i<cPopSize; i++) {
                rFitness[i] /= sumRFitness;
        }

        // Produce the next population
        for (i = 0; i<cPopSize; i += 2) {
                int Parent1 = 0;
```

```
            int Parent2 = 0;
            if(parSel == 0){
                Parent1 = Tournament(Fitness, cTournamentSize);
                Parent2 = Tournament(Fitness, cTournamentSize);
            }else{
                Parent1 = Roulette(rFitness);
                Parent2 = Roulette(rFitness);
            }

            if (cCrossoverRate>Rand01())
                Crossover(CrntPop[Parent1], CrntPop[Parent2], NextPop[i], NextPop[i +
1]);
            else
                Copy(CrntPop[Parent1], CrntPop[Parent2], NextPop[i], NextPop[i + 1]);
            if (cMutationRate<Rand01())Mutate(NextPop[i]);
            if (cMutationRate<Rand01())Mutate(NextPop[i + 1]);
        }
        int **Tmp = CrntPop; CrntPop = NextPop; NextPop = Tmp;
        if(Gen % 20 == 0)
            cout << setw(3) << Gen << ':' << setw(5) << BestFitness << endl;
    }
    if (TargetReached)
        cout << "Target fitness reached: " << BestFitness << " !\n";
    else
        cout << "Target fitness not reached: " << BestFitness << " !\n";
    cout << "Best Individual: ";
    for (i = 0; i<cIndividualLength; i++)
        cout << BestMember[i];
    cout << endl;
    FreeMem(CrntPop, NextPop, Fitness, BestMember);
    char s[20]; cin.getline(s, 20);
    system("pause");
    return 0;
}

void InitPop(int ***CrntPop, int ***NextPop, int **Fitness, int **BestMember, double
**rFitness) {
    int i, j, t, tmp;
    srand((int)time(NULL));
    *CrntPop = new int*[cPopSize];
    *NextPop = new int*[cPopSize];
    for (i = 0; i<cPopSize; i++) {
        (*CrntPop)[i] = new int[cIndividualLength];
        (*NextPop)[i] = new int[cIndividualLength];
    }
    *Fitness = new int[cPopSize];
    *rFitness = new double[cPopSize];
    *BestMember = new int[cIndividualLength];
    if (Fitness == NULL || BestMember == NULL)exit(1);
    for (i = 0; i<cPopSize; i++) {
        for (j = 0; j<cIndividualLength; j++)
            (*CrntPop)[i][j] = j;
        for (j = 0; j<cIndividualLength; j++){
            tmp = RandInt(cIndividualLength);     //generate 0..cIndividualLength-1
            t = (*CrntPop)[i][j];
            (*CrntPop)[i][j] = (*CrntPop)[i][tmp];
            (*CrntPop)[i][tmp] = t;
        }
    }
}

void FreeMem(int **CrntPop, int **NextPop, int *Fitness, int *BestMenber) {
    for (int i = 0; i<cPopSize; i++) {
        delete[]CrntPop[i];
        delete[]NextPop[i];
    }
    delete CrntPop;
    delete NextPop;
    delete Fitness;
    delete BestMenber;
}

int EvaluateFitness(int *Member) {
    //Evaluate the distance of
    int p1, p2;
    int TheFitness = 0;
    for(int i = 1; i < cIndividualLength; i++) {
        p1 = Member[i];
        p2 = Member[i-1];
        TheFitness += distanceTable[p1][p2];
    }
    return(TheFitness);
}

int Tournament(int *Fitness, int TournamentSize) {
    int WinFit = 15000000, Winner;
    for (int i = 0; i < TournamentSize; i++) {
        int j = RandInt(cPopSize);
        if (Fitness[j] < WinFit) {
            WinFit = Fitness[j];
            Winner = j;
        }
    }
    return Winner;
}

int Roulette(double *rFitness) {
    double RandomNumber = Rand01();
    double TempSum = 0;
    for(int i = 0; i < cPopSize; i++){
        TempSum += rFitness[i];
        if(TempSum > RandomNumber) return i;
    }
    return RandInt(cPopSize);
}

void Crossover(int *P1, int *P2, int *C1, int *C2) {
    int i, Left, Right;
    switch (CrossoverType) {
    case eRandom: // swap random genes
        for (i = 0; i<cIndividualLength; i++) {
            if (RandInt(2)) {
                C1[i] = P1[i]; C2[i] = P2[i];
            }
            else {
                C1[i] = P2[i]; C2[i] = P1[i];
            }
        }
        break;
    case eUniform: // swap odd/even genes
        for (i = 0; i<cIndividualLength; i++) {
            if (i % 2) {
                C1[i] = P1[i]; C2[i] = P2[i];
            }
            else {
                C1[i] = P2[i]; C2[i] = P1[i];
            }
        }
        break;
    case eOnePoint:    // perform 1 point x-over
        Left = RandInt(cIndividualLength);
        if (cDebug) {
```

```
        printf("Cut points: 0 <= %d <= %d\n", Left, cIndividualLength - 1);
    }
    for (i = 0; i <= Left; i++) {
        C1[i] = P1[i]; C2[i] = P2[i];
    }
    for (i = Left + 1; i<cIndividualLength; i++) {
        C1[i] = P2[i]; C2[i] = P1[i];
    }
    break;
case eTwoPoint:    // perform 2 point x-over
    Left = RandInt(cIndividualLength - 1);
    Right = Left + 1 + RandInt(cIndividualLength - Left - 1);
    if (cDebug) {
        printf("Cut points: 0 <= %d < %d <= %d\n", Left, Right, cIndividualLength - 1);
    }
    for (i = 0; i <= Left; i++) {
        C1[i] = P1[i]; C2[i] = P2[i];
    }
    for (i = Left + 1; i <= Right; i++) {
        C1[i] = P2[i]; C2[i] = P1[i];
    }
    for (i = Right + 1; i<cIndividualLength; i++) {
        C1[i] = P1[i]; C2[i] = P2[i];
    }
    break;
default:
    printf("Invalid crossover?\n");
    exit(1);
}
//uniq child C1
//initiate the co arrays
for (i = 0; i<cIndividualLength; i++)
    co[i] = 0;
co0.clear();
co2.clear();
for (i = 0; i<cIndividualLength; i++) {
    co[C1[i]] += 1;
    if(co[C1[i]] == 2)
        co2.push_back(i);
}
for (i = 0; i<cIndividualLength; i++) {
    if (co[i] == 0)
        co0.push_back(i);
}
```

```
    int s0 = co0.size();
    for (i = 0; i<s0; i++) {
        C1[co2[0]] = co0[0];
        co2.erase(co2.begin());
        co0.erase(co0.begin());
    }
    for (int i = 0; i < cIndividualLength; i++) {
        for (int j = i + 1; j < cIndividualLength; j++) {
            if (C1[i] == C1[j]) {
                cout << "C1" << endl;
                system("pause");
            }
        }
    }
    //uniq child C2
    //initiate the co arrays
    for (i = 0; i<cIndividualLength; i++)
        co[i] = 0;
    co0.clear();
    co2.clear();
    for (i = 0; i<cIndividualLength; i++) {
        co[C2[i]]++;
        if (co[C2[i]] == 2)
            co2.push_back(i);
    }
    for (i = 0; i<cIndividualLength; i++) {
        if (co[i] == 0)
            co0.push_back(i);
    }
    int s2 = co2.size();
    for (i = 0; i<s2; i++) {
        C2[co2[0]] = co0[0];
        co2.erase(co2.begin());
        co0.erase(co0.begin());
    }
    for (int i = 0; i < cIndividualLength; i++) {
        for (int j = i + 1; j < cIndividualLength; j++) {
            if (C2[i] == C2[j]) {
                cout << "C2" << endl;
                system("pause");
            }
        }
    }
}
```

```
void Mutate(int *Member) {
        int num = (int)(cIndividualLength / 50);
        for (int i = 0; i < num; i++) {
                int Pick = RandInt(cIndividualLength);
                int Pick1 = RandInt(cIndividualLength);
                int t = Member[Pick];
                Member[Pick] = Member[Pick1];
                Member[Pick1] = t;
        }
}

void Copy(int *P1, int *P2, int *C1, int *C2) {
        for (int i = 0; i<cIndividualLength; i++) {
                C1[i] = P1[i]; C2[i] = P2[i];
        }
}

double Rand01() { // 0..1
        return(rand() / (double)(RAND_MAX));
}

int RandInt(int n) { // 0..n-1
        return int(rand() / (double(RAND_MAX) + 1) * n);
}
```