MECS 4510: Evolutionary Computation & Design Automation

Homework 1

Instructor: Hod Lipson

Student name: Xingbei Wang

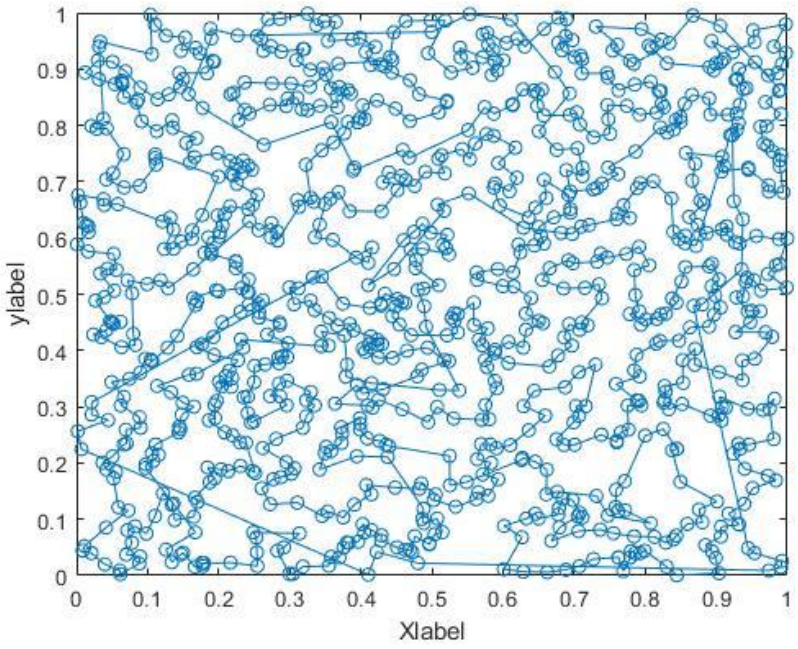
UNI: xw2655

September 29, 2019

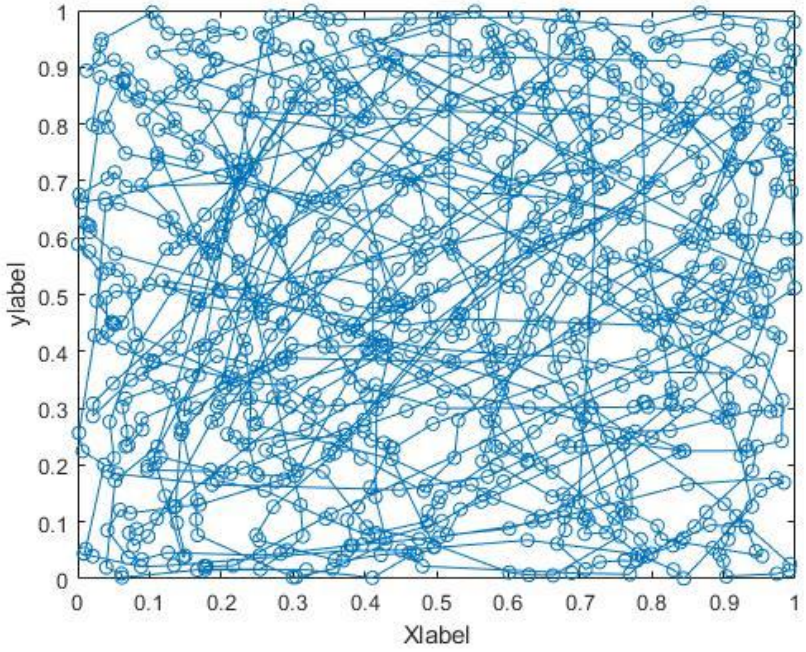
Grace hours used: 0 grace hours remaining: 96

1. Results summary table

Shortest path found

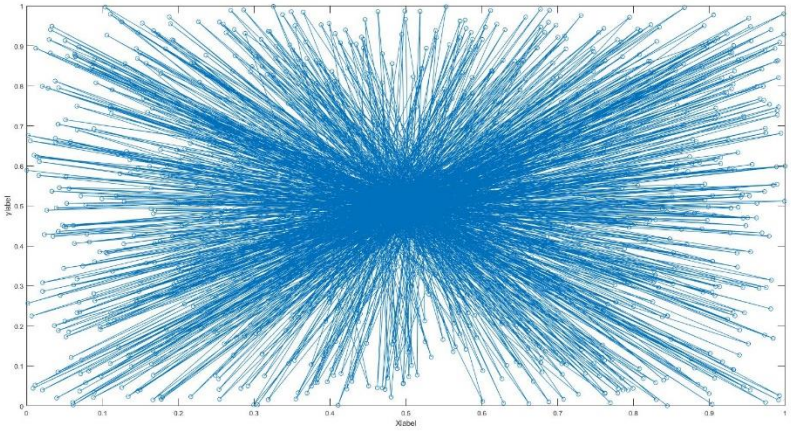


Length = 27.284 generation = 27618 (Super gene)



Length = 71.5584 generation = 276456 (GA hill climbing)

Longest path found



Length = 761.705 generation= 152654 (GA hill climbing)

2. Methods:

1. description

Representation:

In the learning curve of shortest path, the Y axis in the plot is the fitness (1000/distance) and the X axis is log10(evaluations). I also calculate error bar by cycle the program for 5 times and present it in the plot. In the learning curve of longest path, the Y axis is just the distance, because it can represent fitness.

Random search:

I randomly swap two elements in the line of points and measure whether new path's distance is shorter than the current shortest path. If it is shorter. I will record it. After the whole circulation, I will import the shortest path and its length.

Random hill climbing search:

It's almost the same like the former, but we will swap the elements back if new path's distance is longer than the shortest path.

GA Algorithm:

In this code, I create 100 populations and select 50 of them which are shorter than other populations every circle. The variation rate is different from circle to circle. At first the variation rate is 0.5%, when it is over 10000 circles, the variation rate is 50%. Because I think a group of populations with lower diversity can have more variations without destroying its best population and evolve itself. For the crossover, I firstly try one point crossover, but I find it will have a low diversity after 100000 generations. So I decide to use two points crossover which can have a better diversity. My start point is 400, exchanging part is the next 200 points.

This algorithm can arrive 120 after 100000 generations and 80 after 200000 generations.

GA hill climbing Algorithm:

In this code, everything is the same as GA Algorithm. I only apply a judgement on whether the variation can make a population's distance shorter. If not, I cancel this variation.

This algorithm can arrive at 100 after 10000 generations and 80 after 50000 generations.

Super gene GA Algorithm:

In this code, we add a special gene (every point in this gene will choose the next nearest point) in the populations. Others are the same as GA hill climbing Algorithm.

This algorithm starts at 30 and will stop at 26.7(The least one I have arrived) after 20000 generations.

2. Analysis

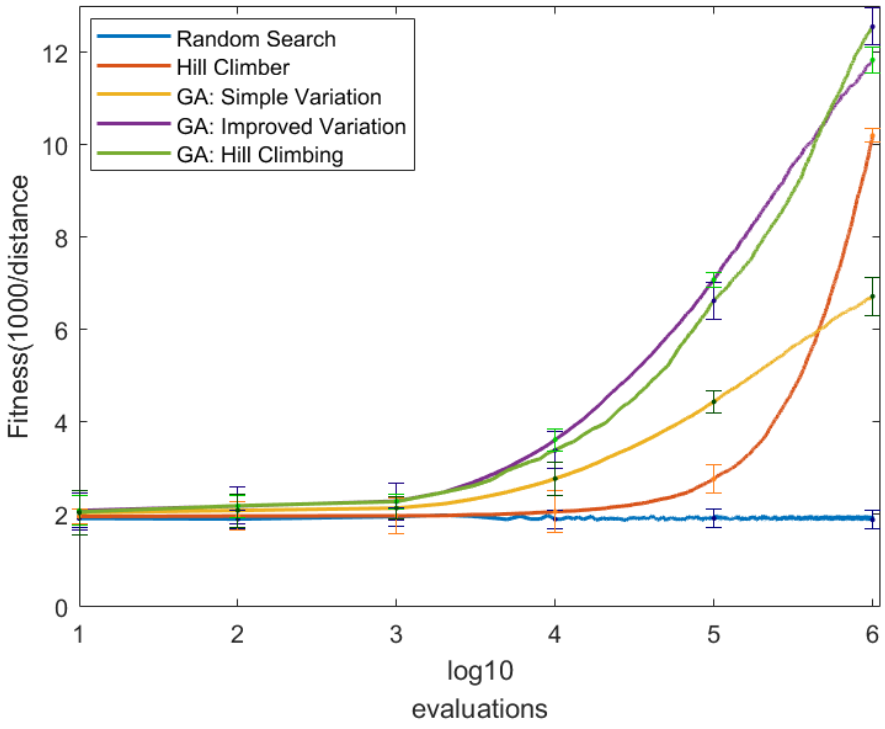
In my opinion, crossover is a powerful tool at the start, it can greatly improve the diversity of the group and rapidly improve the best population, but after 10000 generations of evolvement, the diversity of the group is difficult to be changed by crossover. So the mutation becomes more and more important. With the drop of diversity, mutation becomes very important to generate new elements. High rate of mutation is beneficial for the evolvement. But at the end of evolvement, some paths will crossover each other which can not be solved easily by variation, in that case, we can use crossover to find and cancel these crossovers.

The range of gene variation can also influence the effect of it. I always like to keep the best gene from variation and let others change. In this way, we can maximize the variation of genes and prevent the best from being destroyed.

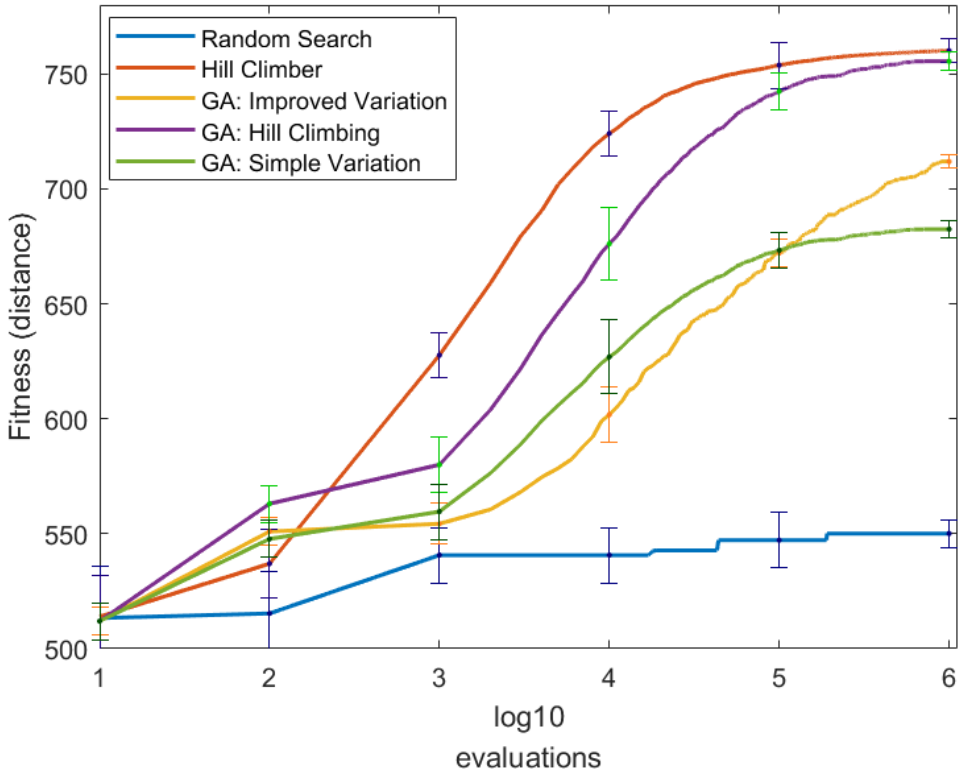
I always like to make variation in populations which are selected as better, and then let them to crossover. In this way, populations can have higher diversity because populations with mutation will generate new populations which will make the whole group more diverse.

3. performance plot

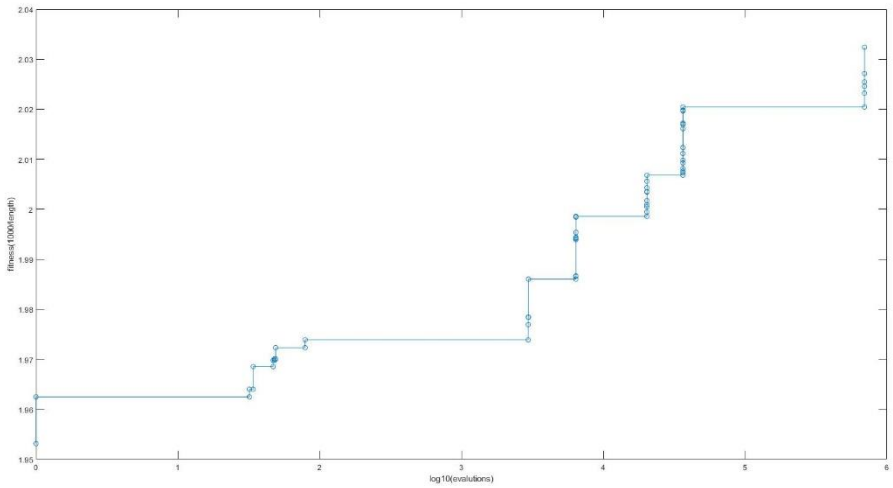
Learning curves of shortest path



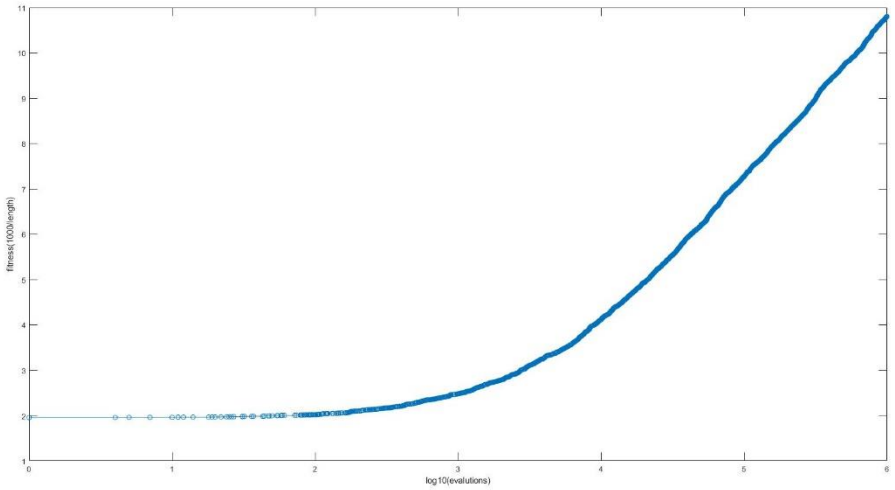
Learning curves of longest path



Baseline curves for comparision

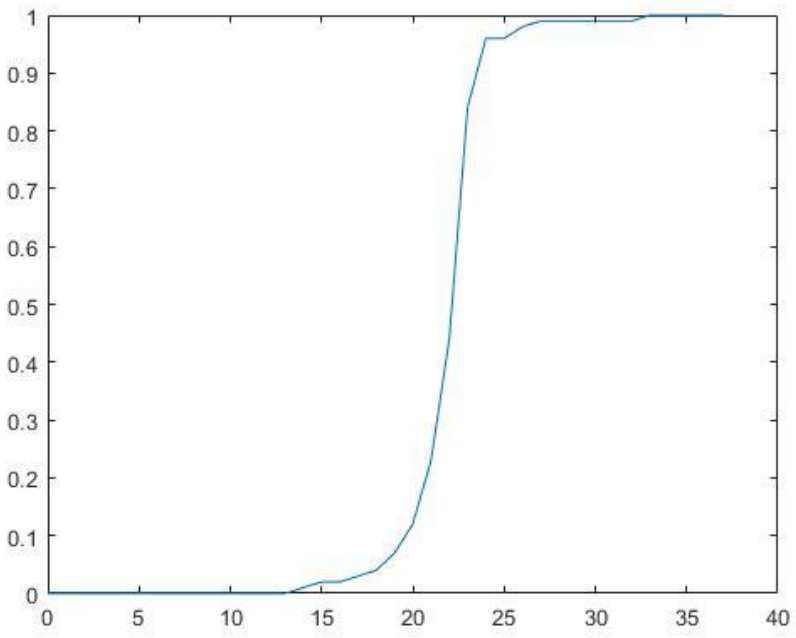


Random search



Random hill climbing search

Convergence plot:



Threshold = 600

Method: GA hill climbing Algorithm

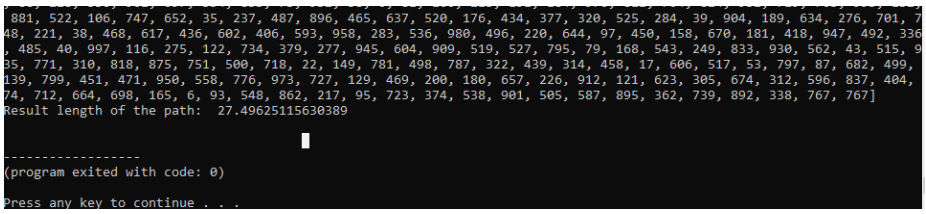
X-axis: generations

Theoretical shortest path using Christofides' algorithm:

Code:

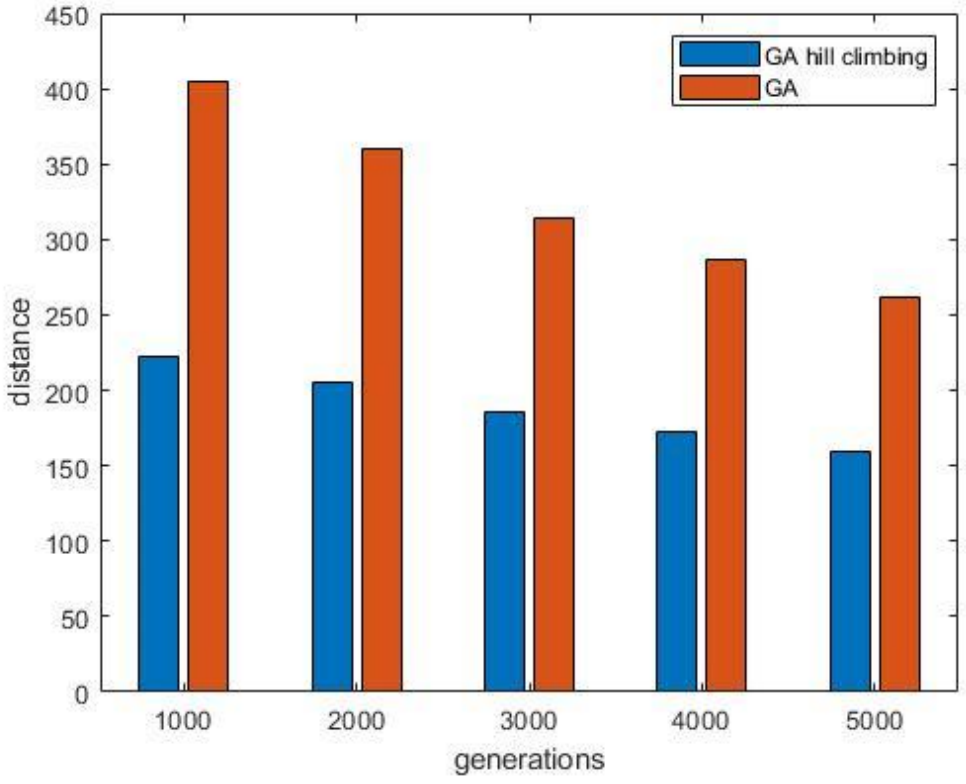
```
file=open('tsp.txt')

dataMat=[]

labelMat=[]
```

```
for line in file.readlines():
        curLine=line.strip().split("\t")
        floatLine = list(map(float,curLine)    )
        dataMat.append(floatLine[0:2])
        labelMat.append(floatLine[-1])
tsp(dataMat)
```



Shortest length is 27.49625115630389

GA hill climbing vs GA (bar chart)



4. Appendix:

random search:

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <ctime>
```

```
#include <cstdlib>
#include <cmath>
#include<windows.h>
using namespace std;


int main()
{
 static double array[1000][2] = { 0.0 };

 ifstream infile;

 infile.open("tsp.txt");

 double* ptr = &array[0][0];

 while (!infile.eof())
 {
     infile >> *ptr;
     ptr++;
 }

 infile.close();

 int k, g, i, p, s = 0;
 double d, dbig = 0, king = 512;
 static double line[1000];
 static double pigg[1000];

 srand((unsigned)time(NULL));
 for (g = 0;g < 100;g++)
 {
     dbig = 0;

     int h = rand() % 999;
     int j = rand() % 999;

     swap(array[h][0], array[j][0]);
     swap(array[h][1], array[j][1]);
     for (i = 0;i < 999;i++)
     {
```

```
          d = sqrt(pow(array[i + 1][0] - array[i][0], 2) + pow(array[i + 1][1] -
array[i][1], 2));

          dbig = dbig + d;

     }

     dbig = dbig + pow(array[0][0] - array[999][0], 2) + pow(array[0][1] -
array[999][1], 2);

     if (dbig < king)

     {

          pigg[s] = 1000 / king;

          pigg[s + 1] = 1000 / dbig;

          king = dbig;

          line[s] = line[s + 1] = g + 1;

          cout << king << endl;

          cout << g << endl;

          s = s + 2;

     }

 }

 for (i = 0;i < s + 3;i++)

 {

     line[i] = log10(line[i]);

 }

 ofstream inFile;

 inFile.open("pddd.txt");

 for (i = 0;i < s;i++)

 {

     inFile << setw(8) << line[i] << "\t";

 }

 inFile.close();

 inFile.open("pxxx.txt");

 for (i = 0;i < s;i++)

 {

     inFile << setw(8) << pigg[i] << "\t";

 }

 inFile.close();

 return 0;

}
```

Random hill climbing:

```
#include <iostream>

#include <fstream>

#include <iomanip>
```

```
#include <ctime>
#include <cstdlib>
#include <cmath>
#include<windows.h>
using namespace std;


int main()
{
 static double array[1000][2] = { 0.0 };
 static double bigking[1000][2] = { 0.0 };
 ifstream infile;

 infile.open("tsp.txt");

 double* ptr = &array[0][0];

 while (!infile.eof())
 {
     infile >> *ptr;
     ptr++;
 }

 infile.close();

 int k,g,i,p,s=0;
 double d, dbig = 0, king =512;

 srand((unsigned)time(NULL));
 for (g = 0;g<1000000;g++)
 {
     dbig = 0;

     int h = rand() % 999;
     int j = rand() % 999;

     swap(array[h][0], array[j][0]);
     swap(array[h][1], array[j][1]);
     for (i = 0;i < 999;i++)
     {
```

```
          d = sqrt(pow(array[i+1][0] - array[i][0], 2) + pow(array[i+1][1] -
array[i][1], 2));

          dbig = dbig + d;

     }

     dbig = dbig + pow(array[0][0] - array[999][0], 2) + pow(array[0][1] -
array[999][1], 2);

     if (dbig < king)

     {

          king = dbig;

          cout << king << endl;

          cout << g << endl;

          s=s+2;

          for (i = 0;i < 100;i++)

          {

               bigking[i][0] = array[i][0];

               bigking[i][1] = array[i][1];

          }

     }

     else

     {

          swap(array[h][0], array[j][0]);

          swap(array[h][1], array[j][1]);

     }

 }

 ofstream inFile;

 inFile.open("pddd.txt");

 for (i = 0;i < 100;i++)

 {

     inFile << setw(8) << bigking[i][0] << "\t";

 }

 inFile << setw(8) << bigking[0][0] << "\t";

 inFile.close();

 inFile.open("pxxx.txt");

 for (i = 0;i < 100 ;i++)

 {

     inFile << setw(8) << bigking[i][1] << "\t";

 }

 inFile << setw(8) << bigking[0][1] << "\t";

 inFile.close();

 return 0;

}
```

Genetic Algorithm:

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <ctime>
#include <cstdlib>
#include <cstdio>
#include <cmath>
using namespace std;

double distance(double arr[], double att[])
{
 double d;
 d = sqrt(pow(arr[0] - att[0], 2) + pow(arr[1] - att[1], 2));
 return d;
}

double fitness(double t)
{
 double h;
 h = 1000 / t;
 return h;
}

void BubbleSort(double* h, int len, int* p)
{

 if (h == NULL) return;
 if (len <= 1) return;
 for (int i = 0; i < len - 1; ++i)
      for (int j = 0; j < len - 1 - i; ++j)
           if (h[j] > h[j + 1])
           {
                swap(h[j], h[j + 1]);
                swap(p[j], p[j + 1]);
           }
 return;
}
void crossover(int* arr1, int* arr2, int* arr3, int* arr4, int bb, int sta)
{
 int i, ch = 0;
```

```
 for (i = 0; i < 1000; i++)
      arr4[i] = arr2[i];
 for (ch = sta; ch < sta + bb; ch++)
 {
      i = 0;
      while (arr4[i] != arr1[ch])
      {
           i++;
      }
      for (; i < 999; i++)
           arr4[i] = arr4[i + 1];
      arr4[999] = 0;
 }
 for (i = 0; i < sta; i++)
      arr3[i] = arr4[i];
 for (sta; i < sta + bb; i++)
      arr3[i] = arr1[i];
 for (sta + bb; i < 1000; i++)
      arr3[i] = arr4[i - bb];
 return;
}
int main()
{
 int y = 50;
 srand((unsigned)time(NULL));
 static double array[1000][2] = { 0.0 };
 static int population[100][1000] = { 0.0 };     //number
 static int elistic[51][1000] = { 0.0 };
 static int mixed[50][1000] = { 0.0 };
 int sub[1000] = { 0.0 };
 double bigking[100] = { 0.0 };
 double king = 1000;
 double bigkingg[1000][2] = { 0.0 };
 int rank[100];
 int t, b, i, j, g, a, s, q, o, m, w = 0;
 ifstream infile;
 infile.open("tsp.txt");
 double* ptr = &array[0][0];
 while (!infile.eof())
 {
      infile >> *ptr;
```

```
    ptr++;
}

infile.close();
ofstream outfile;
ofstream outfile2;
outfile.open("GAHC.txt");
outfile2.open("x_axis.txt");

for (i = 0; i < 2 * y; i++)
{
    for (j = 0; j < 1000; j++)
    {
        population[i][j] = j;
    }
    for (g = 0; g < 1000; g++)
    {
        w = rand() % 999;
        swap(population[i][g], population[i][w]);
    }
}
for (o = 0; o < 50000; o++)
{
    for (i = 0; i < 2 * y; i++)
    {
        double ddl = 0;
        rank[i] = i;
        for (j = 0; j < 999; j++)
        {
            ddl = ddl + distance(array[population[i][j]], array[population[i][j +
1]]);
        }
        ddl = ddl + distance(array[population[i][999]], array[population[i][0]]);
        if (ddl < king)
        {
            king = ddl;
            cout << king << endl;
            cout << o << endl;
        }
        bigking[i] = ddl;
    }
```

```
BubbleSort(bigking, 2 * y, rank);
for (i = 0; i < y; i++)
{
    for (j = 0; j < 1000; j++)
    {
        elistic[i][j] = population[rank[i]][j];
    }
}
for (j = 0; j < 1000; j++)
{
    elistic[y][j] = population[rank[0]][j];
}
for (i = 0; i < y; i++)
    crossover(elistic[i], elistic[i + 1], mixed[i], sub, 400, 400);
for (i = 0; i < y; i++)
{
    for (j = 0; j < 1000; j++)
    {
        population[i][j] = elistic[i][j];
    }
}
for (i = y; i < 2 * y; i++)
{
    for (j = 0; j < 1000; j++)
    {
        population[i][j] = mixed[i - y][j];
    }
}
for (i = 0; i < 40; i++)
{
    q = 1 + rand() % 999;
    w = 1 + rand() % 999;
    a = rand() % 99;
    swap(population[a][w], population[a][q]);
}
if (o > 5000)
{
    for (i = 0; i < 20; i++)
    {
        q = 1 + rand() % 999;
        w = 1 + rand() % 999;
```

```
            a = rand() % 99;
            swap(population[a][w], population[a][q]);
        }
    }


    for (b = 0; b < 1000; b++)
    {
        bigkingg[b][0] = array[population[0][b]][0];
        bigkingg[b][1] = array[population[0][b]][1];
    }
    double temp;
    if (o == 1)
    {
        temp = king;
        cout << "The shortest distance is: " << temp << endl;
        cout << o << endl;
        outfile << 1000 / temp << endl;//save the shortest path to a txt file
        outfile2 << o << endl;//save the run time to a txt file
    }
    if (o == 3)
    {
        temp = king;
        cout << "The shortest distance is: " << temp << endl;
        cout << o << endl;
        outfile << 1000 / temp << endl;//save the shortest path to a txt file
        outfile2 << o << endl;//save the run time to a txt file
    }
    if (o % 5 == 0)
    {
        temp = king;
        cout << "The shortest distance is: " << temp << endl;
        cout << o << endl;
        outfile << 1000 / temp << endl;//save the shortest path to a txt file
        outfile2 << o << endl;//save the run time to a txt file
    }
}
outfile.close();
outfile2.close();
return 0;
}
```

Genetic hill climbing Algorithm

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <ctime>
#include <cstdlib>
#include <cmath>
using namespace std;

double distance(double arr[], double att[])
{
 double d;
 d = sqrt(pow(arr[0] - att[0], 2) + pow(arr[1] - att[1], 2));
 return d;
}

double fitness(double t)
{
 double h;
 h = 1000 / t;
 return h;
}

void BubbleSort(double* h, int len, int* p)
{

 if (h == NULL) return;
 if (len <= 1) return;
 for (int i = 0;i < len - 1;++i)
      for (int j = 0;j < len - 1 - i;++j)
           if (h[j] < h[j + 1])
           {
                swap(h[j], h[j + 1]);
                swap(p[j], p[j + 1]);
           }


 return;
}
```

```
void crossover(int* arr1, int* arr2, int* arr3, int* arr4, int bb, int sta)
{
 int i, ch = 0;
 for (i = 0;i < 1000;i++)
      arr4[i] = arr2[i];
 for (ch = sta; ch < sta + bb;ch++)
 {
      i = 0;
      while (arr4[i] != arr1[ch])
      {
           i++;
      }


      for (;i < 999;i++)
           arr4[i] = arr4[i + 1];
      arr4[999] = 0;

 }
 for (i = 0;i < sta;i++)
      arr3[i] = arr4[i];
 for (sta;i < sta + bb;i++)
      arr3[i] = arr1[i];
 for (sta + bb; i < 1000; i++)
      arr3[i] = arr4[i - bb];
 return;
}


int main()
{
 int y = 50;
 srand((unsigned)time(NULL));
 static double array[1000][2] = { 0.0 };
 static int population[100][1000] = { 0.0 };     //number
 static int elistic[51][1000] = { 0.0 };
 static int mixed[50][1000] = { 0.0 };
 int sub[1000] = { 0.0 };
 double bigking[100] = { 0.0 };
 double king = 100;
```

```
double bigkingg[1000][2] = { 0.0 };
int rank[100];
int t, b, i, j, g, a, s, q, o, m, w = 0;

ifstream infile;

infile.open("tsp.txt");

double* ptr = &array[0][0];

while (!infile.eof())
{
    infile >> *ptr;
    ptr++;
}

infile.close();

for (i = 0;i < 4 * y;i++)
{
    for (j = 0;j < 1000;j++)
    {
        population[i][j] = j;
    }
    for (g = 0;g < 1000;g++)
    {
        w = rand() % 999;
        swap(population[i][g], population[i][w]);
    }
}
for (o = 0;o < 10000;o++)
{
    for (i = 0;i < 2 * y;i++)
    {
        double ddl = 0;
        rank[i] = i;
        for (j = 0;j < 999;j++)
        {
            ddl = ddl + distance(array[population[i][j]], array[population[i][j +
1]]);
        }
```

```
            ddl = ddl + distance(array[population[i][999]], array[population[i][0]]);
            if (ddl > king)
            {
                king = ddl;
                cout << king << endl;
                cout << o << endl;
            }
            bigking[i] = ddl;
        }
        BubbleSort(bigking, 2 * y, rank);
        for (i = 0;i < y;i++)
        {
            for (j = 0;j < 1000;j++)
            {
                elistic[i][j] = population[rank[i]][j];
            }
        }

        for (j = 0;j < 1000;j++)
        {
            elistic[y][j] = population[rank[0]][j];
        }
        if (o > 10000)
        {
            for (i = 0;i < 20;i++)
            {
                q = 1 + rand() % 999;
                w = 1 + rand() % 999;
                a = rand() % 48;
                swap(elistic[a][w], elistic[a][q]);
            }
        }

        for (i = 0;i < y;i++)
            crossover(elistic[i], elistic[i + 1], mixed[i], sub, 400, 400);
    //crossover

        if (1 == rand() % 2)
        {
            q = 1 + rand() % 999;
            w = 1 + rand() % 999;
```

```
            a = rand() % 48;
            swap(elistic[a][w], elistic[a][q]);
        }

        for (b = 0;b < 100;b++)
        {
            i = 1;
            q = 1 + rand() % 999;
            w = 1 + rand() % 999;
            double ddp = 0;
            for (j = 0;j < 999;j++)
            {
                ddp = ddp + distance(array[elistic[i][j]], array[elistic[i][j + 1]]);
            }
            ddp = ddp + distance(array[elistic[i][999]], array[elistic[i][0]]);
            swap(elistic[i][w], elistic[i][q]);
            double ddl = 0;
            for (j = 0;j < 999;j++)
            {
                ddl = ddl + distance(array[elistic[i][j]], array[elistic[i][j + 1]]);
            }
            ddl = ddl + distance(array[elistic[i][999]], array[elistic[i][0]]);
            if (ddl < ddp)
                swap(elistic[i][w], elistic[i][q]);
        }
        for (i = 0;i < y;i++)
        {
            for (j = 0;j < 1000;j++)
            {
                population[i][j] = elistic[i][j];
            }
        }
        for (i = y;i < 2 * y;i++)
        {
            for (j = 0;j < 1000;j++)
            {
                population[i][j] = mixed[i - y][j];
            }
        }
    }
```

```
for (b = 0;b < 1000;b++)
{
     bigkingg[b][0] = array[population[0][b]][0];
     bigkingg[b][1] = array[population[0][b]][1];
}

ofstream inFile;
inFile.open("pddd.txt");
for (i = 0;i < 1000;i++)
{
     inFile << setw(8) << bigkingg[i][0] << "\t";
}
inFile << setw(8) << bigkingg[0][0] << "\t";
inFile.close();
inFile.open("pxxx.txt");
for (i = 0;i < 1000;i++)
{
     inFile << setw(8) << bigkingg[i][1] << "\t";
}
inFile << setw(8) << bigkingg[0][1] << "\t";
inFile.close();

return 0;
}
```

Super gene Genetic Algorithm:

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <ctime>
#include <cstdlib>
#include <cmath>
using namespace std;

double distance(double arr[], double att[])
{
 double d;
 d = sqrt(pow(arr[0] - att[0], 2) + pow(arr[1] - att[1], 2));
 return d;
}
```

```
double fitness(double t)
{
 double h;
 h = 1000 / t;
 return h;
}

void BubbleSort(double* h, int len, int* p)
{

 if (h == NULL) return;
 if (len <= 1) return;
 for (int i = 0;i < len - 1;++i)
      for (int j = 0;j < len - 1 - i;++j)
           if (h[j] > h[j + 1])
           {
                swap(h[j], h[j + 1]);
                swap(p[j], p[j + 1]);
           }


 return;
}

void crossover(int* arr1, int* arr2, int* arr3, int* arr4, int bb, int sta)
{
 int i, ch = 0;
 for (i = 0;i < 1000;i++)
      arr4[i] = arr2[i];
 for (ch = sta; ch < sta + bb;ch++)
 {
      i = 0;
      while (arr4[i] != arr1[ch])
      {
           i++;
      }


           for (;i < 999;i++)
                arr4[i] = arr4[i + 1];
           arr4[999] = 0;
```

```
}
for (i = 0;i < sta;i++)
    arr3[i] = arr4[i];
for (sta;i < sta + bb;i++)
    arr3[i] = arr1[i];
for (sta+bb; i < 1000; i++)
    arr3[i] = arr4[i - bb];
return;
}


int main()
{
int y = 50;
srand((unsigned)time(NULL));
static double array[1000][2] = { 0.0 };
static int population[100][1000] = { 0.0 };    //number
static int elistic[51][1000] = { 0.0 };
static int mixed[50][1000] = { 0.0 };
int sub[1000] = { 0.0 };
double bigking[100] = { 0.0 };
double king = 1000;
double bigkingg[1000][2] = { 0.0 };
int rank[100];
int t,b,i, j, g, a, s, q, o, m, w = 0;

ifstream infile;

infile.open("tsp.txt");

double* ptr = &array[0][0];

while (!infile.eof())
{
    infile >> *ptr;
    ptr++;
}

infile.close();
```

```
population[0][0] = 0;
double ddl3;
for (j = 0;j < 999;j++)
{
     king = 1000;
     for (g = 1;g < 1000;g++)
     {
          int flag = 0;
          for (t = 0;t < j + 1;t++)
          {
               if (g == population[0][t])
                    flag = 1;
          }
          if (flag == 0)
          {
               ddl3 = distance(array[population[0][j]], array[g]);
               if (ddl3 < king)
               {
                    king = ddl3;
                    m = g;
               }
          }
     }
     population[0][j + 1] = m;
}
king = 1000;
for (i = 1;i < 2 * y;i++)
{
     for (j = 0;j < 1000;j++)
     {
          population[i][j] = j;
     }
     for (g = 0;g < 1000;g++)
     {
          w = rand() % 999;
          swap(population[i][g], population[i][w]);
     }
}
for (o = 0;o < 100000;o++)
{
     for (i = 0;i < 2 * y;i++)
```

```
    {
        double ddl = 0;
        rank[i] = i;
        for (j = 0;j < 999;j++)
        {
            ddl = ddl + distance(array[population[i][j]], array[population[i][j +
1]]);
        }
        ddl = ddl + distance(array[population[i][999]], array[population[i][0]]);
        if (ddl < king)
        {
            king = ddl;
            cout << king << endl;
            cout << o << endl;
        }
        bigking[i] = ddl;
    }
    BubbleSort(bigking, 2 * y, rank);
    for (i = 0;i < y;i++)
    {
        for (j = 0;j < 1000;j++)
        {
            elistic[i][j] = population[rank[i]][j];
        }
    }

    for (j = 0;j < 1000;j++)
    {
        elistic[y][j] = population[rank[0]][j];
    }

    if (o > 10000)
    {
        for (i = 0;i < 20;i++)
        {
            q = 1 + rand() % 999;
            w = 1 + rand() % 999;
            a = rand() % 48;
            swap(elistic[a][w], elistic[a][q]);
        }
    }
```

```
for (i = 0;i < y;i++)
    crossover(elistic[i], elistic[i + 1], mixed[i], sub, 200, 400);

if (1 == rand() % 2)
{
    q = 1 + rand() % 999;
    w = 1 + rand() % 999;
    a = rand() % 48;
    swap(elistic[a][w], elistic[a][q]);
}

for (b = 0;b < 200;b++)
{
    i = 1;
    q = 1 + rand() % 999;
    w = 1 + rand() % 999;
    double ddp = 0;
    for (j = 0;j < 999;j++)
    {
        ddp = ddp + distance(array[elistic[i][j]], array[elistic[i][j + 1]]);
    }
    ddp = ddp + distance(array[elistic[i][999]], array[elistic[i][0]]);
    swap(elistic[i][w], elistic[i][q]);
    double ddl = 0;
    for (j = 0;j < 999;j++)
    {
        ddl = ddl + distance(array[elistic[i][j]], array[elistic[i][j + 1]]);
    }
    ddl = ddl + distance(array[elistic[i][999]], array[elistic[i][0]]);
    if (ddl > ddp)
        swap(elistic[i][w], elistic[i][q]);
}
for (i = 0;i < y;i++)
{
    for (j = 0;j < 1000;j++)
    {
        population[i][j] = elistic[i][j];
    }
}
for (i = y;i < 2 * y;i++)
```

```
    {
        for (j = 0;j < 1000;j++)
        {
            population[i][j] = mixed[i - y][j];
        }
    }
}



for (b = 0;b < 1000;b++)
{
    bigkingg[b][0] = array[population[0][b]][0];
    bigkingg[b][1] = array[population[0][b]][1];
}

ofstream inFile;
inFile.open("pddd.txt");
for (i = 0;i < 1000;i++)
{
    inFile << setw(8) << bigkingg[i][0] << "\t";
}
inFile << setw(8) << bigkingg[0][0] << "\t";
inFile.close();
inFile.open("pxxx.txt");
for (i = 0;i < 1000;i++)
{
    inFile << setw(8) << bigkingg[i][1] << "\t";
}
inFile << setw(8) << bigkingg[0][1] << "\t";
inFile.close();
return 0;
}
```