/\*

\*遗传算法(GA) 解决TSP 问题

\*案例参考自《MATLAB 智能算法30个案例分析》

\*本例以14个城市为例，14个城市的位置坐标如下(括号内第一个元素为X坐标，第二个为纵坐标)：1:(16.47,96.10) 2:(16.47,94.44) 3:(20.09,92.54)

\*4:(22.39,93.37) 5:(25.23,97.24) 6:(22.00,96.05) 7:(20.47,97.02) 8:(17.20,96.29) 9:(16.30,97.38) 10:(14.05,98.12) 11:(16.53,97.38)

\*12:(21.52,95.59) 13:(19.41,97.13) 14:(20.09,92.55)

\*遗传算法实现的步骤为:(1)编码 (2) 种群初始化 (3) 构造适应度函数 (4) 选择操作 (5) 交叉操作 (6) 变异操作 (7) 进化逆转操作

\* 具体实现的步骤这里不详细说，参考《MATLAB 智能算法30个案例分析》P38 - P40

\* update in 16/12/4

\* author:Lyrichu

\* email:919987476@qq.com

\*/

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

#include<time.h>

#define maxgen 200 // 最大进化代数

#define sizepop 100 // 种群数目

#define pcross 0.6 // 交叉概率

#define pmutation 0.1 // 变异概率

#define lenchrom 14 // 染色体长度(这里即为城市个数)

double city\_pos[lenchrom][2] = {{16.47,96.10},{16.47,94.44},{20.09,92.54},{22.39,93.37},{25.23,97.24},{22.00,96.05},{20.47,97.02},

{17.20,96.29},{16.30,97.38},{14.05,98.12},{16.53,97.38},{21.52,95.59},{19.41,97.13},{20.09,92.55}}; // 定义二维数组存放14个城市的X、Y坐标

int chrom[sizepop][lenchrom]; // 种群

int best\_result[lenchrom]; // 最佳路线

double min\_distance; // 最短路径长度

// 函数声明

void init(void); // 种群初始化函数

double distance(double \*,double \*); // 计算两个城市之间的距离

double \* min(double \*); // 计算距离数组的最小值

double path\_len(int \*); // 计算某一个方案的路径长度，适应度函数为路线长度的倒数

void Choice(int [sizepop][lenchrom]); // 选择操作

void Cross(int [sizepop][lenchrom]); // 交叉操作

void Mutation(int [sizepop][lenchrom]); // 变异操作

void Reverse(int [sizepop][lenchrom]); // 逆转操作

// 种群初始化

void init(void)

{

int num = 0;

while(num < sizepop)

{

for(int i=0;i<sizepop;i++)

for(int j=0;j<lenchrom;j++)

chrom[i][j] = j+1;

num++;

for(int i=0;i<lenchrom-1;i++)

{

for(int j=i+1;j<lenchrom;j++)

{

int temp = chrom[num][i];

chrom[num][i] = chrom[num][j];

chrom[num][j] = temp; // 交换第num个个体的第i个元素和第j个元素

num++;

if(num >= sizepop)

break;

}

if(num >= sizepop)

break;

}

// 如果经过上面的循环还是无法产生足够的初始个体，则随机再补充一部分

// 具体方式就是选择两个基因位置，然后交换

while(num < sizepop)

{

double r1 = ((double)rand())/(RAND\_MAX+1.0);

double r2 = ((double)rand())/(RAND\_MAX+1.0);

int p1 = (int)(lenchrom\*r1); // 位置1

int p2 = (int)(lenchrom\*r2); // 位置2

int temp = chrom[num][p1];

chrom[num][p1] = chrom[num][p2];

chrom[num][p2] = temp; // 交换基因位置

num++;

}

}

}

// 距离函数

double distance(double \* city1,double \* city2)

{

double x1 = \*city1;

double y1 = \*(city1+1);

double x2 = \*(city2);

double y2 = \*(city2+1);

double dis = sqrt((x1-x2)\*(x1-x2)+(y1-y2)\*(y1-y2));

return dis;

}

// min()函数

double \* min(double \* arr)

{

static double best\_index[2];

double min\_dis = \*arr;

double min\_index = 0;

for(int i=1;i<sizepop;i++)

{

double dis = \*(arr+i);

if(dis < min\_dis)

{

min\_dis = dis;

min\_index = i;

}

}

best\_index[0] = min\_index;

best\_index[1] = min\_dis;

return best\_index;

}

// 计算路径长度

double path\_len(int \* arr)

{

double path = 0; // 初始化路径长度

int index = \*arr; // 定位到第一个数字(城市序号)

for(int i=0;i<lenchrom-1;i++)

{

int index1 = \*(arr+i);

int index2 = \*(arr+i+1);

double dis = distance(city\_pos[index1-1],city\_pos[index2-1]);

path += dis;

}

int last\_index = \*(arr+lenchrom-1); // 最后一个城市序号

int first\_index = \*arr; // 第一个城市序号

double last\_dis = distance(city\_pos[last\_index-1],city\_pos[first\_index-1]);

path = path + last\_dis;

return path; // 返回总的路径长度

}

// 选择操作

void Choice(int chrom[sizepop][lenchrom])

{

double pick;

double choice\_arr[sizepop][lenchrom];

double fit\_pro[sizepop];

double sum = 0;

double fit[sizepop]; // 适应度函数数组(距离的倒数)

for(int j=0;j<sizepop;j++)

{

double path = path\_len(chrom[j]);

double fitness = 1/path;

fit[j] = fitness;

sum += fitness;

}

for(int j=0;j<sizepop;j++)

{

fit\_pro[j] = fit[j]/sum; // 概率数组

}

// 开始轮盘赌

for(int i=0;i<sizepop;i++)

{

pick = ((double)rand())/RAND\_MAX; // 0到1之间的随机数

for(int j=0;j<sizepop;j++)

{

pick = pick - fit\_pro[j];

if(pick<=0)

{

for(int k=0;k<lenchrom;k++)

choice\_arr[i][k] = chrom[j][k]; // 选中一个个体

break;

}

}

}

for(int i=0;i<sizepop;i++)

{

for(int j=0;j<lenchrom;j++)

chrom[i][j] = choice\_arr[i][j];

}

}

//交叉操作

void Cross(int chrom[sizepop][lenchrom])

{

double pick;

double pick1,pick2;

int choice1,choice2;

int pos1,pos2;

int temp;

int conflict1[lenchrom]; // 冲突位置

int conflict2[lenchrom];

int num1,num2;

int index1,index2;

int move = 0; // 当前移动的位置

while(move<lenchrom-1)

{

pick = ((double)rand())/RAND\_MAX; // 用于决定是否进行交叉操作

if(pick > pcross)

{

move += 2;

continue; // 本次不进行交叉

}

// 采用部分映射杂交

choice1 = move; // 用于选取杂交的两个父代

choice2 = move+1; // 注意避免下标越界

pick1 = ((double)rand())/(RAND\_MAX+1.0);

pick2 = ((double)rand())/(RAND\_MAX+1.0);

pos1 = (int)(pick1\*lenchrom); // 用于确定两个杂交点的位置

pos2 = (int)(pick2\*lenchrom);

while(pos1 > lenchrom -2 || pos1 < 1)

{

pick1 = ((double)rand())/(RAND\_MAX+1.0);

pos1 = (int)(pick1\*lenchrom);

}

while(pos2 > lenchrom -2 || pos2 < 1)

{

pick2 = ((double)rand())/(RAND\_MAX+1.0);

pos2 = (int)(pick2\*lenchrom);

}

if(pos1 > pos2)

{

temp = pos1;

pos1 = pos2;

pos2 = temp; // 交换pos1和pos2的位置

}

for(int j=pos1;j<=pos2;j++)

{

temp = chrom[choice1][j];

chrom[choice1][j] = chrom[choice2][j];

chrom[choice2][j] = temp; // 逐个交换顺序

}

num1 = 0;

num2 = 0;

if(pos1 > 0 && pos2 < lenchrom-1)

{

for(int j =0;j<=pos1-1;j++)

{

for(int k=pos1;k<=pos2;k++)

{

if(chrom[choice1][j] == chrom[choice1][k])

{

conflict1[num1] = j;

num1++;

}

if(chrom[choice2][j] == chrom[choice2][k])

{

conflict2[num2] = j;

num2++;

}

}

}

for(int j=pos2+1;j<lenchrom;j++)

{

for(int k=pos1;k<=pos2;k++)

{

if(chrom[choice1][j] == chrom[choice1][k])

{

conflict1[num1] = j;

num1++;

}

if(chrom[choice2][j] == chrom[choice2][k])

{

conflict2[num2] = j;

num2++;

}

}

}

}

if((num1 == num2) && num1 > 0)

{

for(int j=0;j<num1;j++)

{

index1 = conflict1[j];

index2 = conflict2[j];

temp = chrom[choice1][index1]; // 交换冲突的位置

chrom[choice1][index1] = chrom[choice2][index2];

chrom[choice2][index2] = temp;

}

}

move += 2;

}

}

// 变异操作

// 变异策略采取随机选取两个点，将其对换位置

void Mutation(int chrom[sizepop][lenchrom])

{

double pick,pick1,pick2;

int pos1,pos2,temp;

for(int i=0;i<sizepop;i++)

{

pick = ((double)rand())/RAND\_MAX; // 用于判断是否进行变异操作

if(pick > pmutation)

continue;

pick1 = ((double)rand())/(RAND\_MAX+1.0);

pick2 = ((double)rand())/(RAND\_MAX+1.0);

pos1 = (int)(pick1\*lenchrom); // 选取进行变异的位置

pos2 = (int)(pick2\*lenchrom);

while(pos1 > lenchrom-1)

{

pick1 = ((double)rand())/(RAND\_MAX+1.0);

pos1 = (int)(pick1\*lenchrom);

}

while(pos2 > lenchrom-1)

{

pick2 = ((double)rand())/(RAND\_MAX+1.0);

pos2 = (int)(pick2\*lenchrom);

}

temp = chrom[i][pos1];

chrom[i][pos1] = chrom[i][pos2];

chrom[i][pos2] = temp;

}

}

// 进化逆转操作

void Reverse(int chrom[sizepop][lenchrom])

{

double pick1,pick2;

double dis,reverse\_dis;

int n;

int flag,pos1,pos2,temp;

int reverse\_arr[lenchrom];

for(int i=0;i<sizepop;i++)

{

flag = 0; // 用于控制本次逆转是否有效

while(flag == 0)

{

pick1 = ((double)rand())/(RAND\_MAX+1.0);

pick2 = ((double)rand())/(RAND\_MAX+1.0);

pos1 = (int)(pick1\*lenchrom); // 选取进行逆转操作的位置

pos2 = (int)(pick2\*lenchrom);

while(pos1 > lenchrom-1)

{

pick1 = ((double)rand())/(RAND\_MAX+1.0);

pos1 = (int)(pick1\*lenchrom);

}

while(pos2 > lenchrom -1)

{

pick2 = ((double)rand())/(RAND\_MAX+1.0);

pos2 = (int)(pick2\*lenchrom);

}

if(pos1 > pos2)

{

temp = pos1;

pos1 = pos2;

pos2 = temp; // 交换使得pos1 <= pos2

}

if(pos1 < pos2)

{

for(int j=0;j<lenchrom;j++)

reverse\_arr[j] = chrom[i][j]; // 复制数组

n = 0; // 逆转数目

for(int j=pos1;j<=pos2;j++)

{

reverse\_arr[j] = chrom[i][pos2-n]; // 逆转数组

n++;

}

reverse\_dis = path\_len(reverse\_arr); // 逆转之后的距离

dis = path\_len(chrom[i]); // 原始距离

if(reverse\_dis < dis)

{

for(int j=0;j<lenchrom;j++)

chrom[i][j] = reverse\_arr[j]; // 更新个体

}

}

flag = 1;

}

}

}

// 主函数

int main(void)

{

time\_t start,finish;

start = clock(); // 开始计时

srand((unsigned)time(NULL)); // 初始化随机数种子

init(); // 初始化种群

int best\_fit\_index = 0; //最短路径出现代数

double distance\_arr[sizepop];

double dis;

for(int j=0;j<sizepop;j++)

{

dis = path\_len(chrom[j]);

distance\_arr[j] = dis;

}

double \* best\_index = min(distance\_arr); // 计算最短路径及序号

min\_distance = \*(best\_index+1); // 最短路径

int index = (int)(\*best\_index); // 最短路径序号

for(int j=0;j<lenchrom;j++)

best\_result[j] = chrom[index][j]; // 最短路径序列

// 开始进化

double \* new\_arr;

double new\_min\_dis;

int new\_index;

for(int i=0;i<maxgen;i++)

{

Choice(chrom); // 选择

Cross(chrom); //交叉

Mutation(chrom); //变异

Reverse(chrom); // 逆转操作

for(int j=0;j<sizepop;j++)

distance\_arr[j] = path\_len(chrom[j]); // 距离数组

new\_arr = min(distance\_arr);

new\_min\_dis = \*(new\_arr+1); //新的最短路径

if(new\_min\_dis < min\_distance)

{

min\_distance = new\_min\_dis; // 更新最短路径

new\_index =(int)(\*new\_arr);

for(int j=0;j<lenchrom;j++)

best\_result[j] = chrom[new\_index][j]; // 更新最短路径序列

best\_fit\_index = i+1; // 最短路径代数

}

}

finish = clock(); // 计算结束

double duration = ((double)(finish-start))/CLOCKS\_PER\_SEC; // 计算耗时

printf("本程序使用遗传算法求解规模为%d的TSP问题,种群数目为:%d,进化代数为:%d\n",lenchrom,sizepop,maxgen);

printf("得到最短路径为:%d-->%d-->%d-->%d-->%d-->%d-->%d-->%d-->%d-->%d-->%d-->%d-->%d-->%d\n",best\_result[0],best\_result[1],best\_result[2],

best\_result[3],best\_result[4],best\_result[5],best\_result[6],best\_result[7],best\_result[8],best\_result[9],best\_result[10],best\_result[11],

best\_result[12],best\_result[13]);

printf("最短路径长度为:%lf,得到最短路径在第%d代.\n",min\_distance,best\_fit\_index);

printf("程序耗时:%lf秒.\n",duration);

system("PAUSE");

return 0;

}