# ****人工智能第五次实验****

# ****遗传算法思想****

　　借鉴生物进化论，遗传算法将要解决的问题模拟成一个生物进化的过程，通过复制、交叉、突变等操作产生下一代的解，并逐步淘汰掉适应度函数值低的解，增加适应度函数值高的解。这样进化N代后就很有可能会进化出适应度函数值很高的个体。

　　举个例子，使用遗传算法解决“0-1背包问题”的思路：0-1背包的解可以编码为一串0-1字符串（0：不取，1：取） ；首先，随机产生M个0-1字符串，然后评价这些0-1字符串作为0-1背包问题的解的优劣；然后，随机选择一些字符串通过交叉、突变等操作产生下一代的M个字符串，而且较优的解被选中的概率要比较高。这样经过G代的进化后就可能会产生出0-1背包问题的一个“近似最优解”。

**编码**：需要将问题的解编码成字符串的形式才能使用遗传算法。最简单的一种编码方式是二进制编码，即将问题的解编码成二进制位数组的形式。例如，问题的解是整数，那么可以将其编码成二进制位数组的形式。将0-1字符串作为0-1背包问题的解就属于二进制编码。

　　遗传算法有3个最基本的操作：选择，交叉，变异。

**选择**：选择一些染色体来产生下一代。一种常用的选择策略是 **“比例选择”**，也就是个体被选中的概率与其适应度函数值成正比。假设群体的个体总数是M，那么那么一个体Xi被选中的概率为f(Xi)/( f(X1) + f(X2) + …….. + f(Xn) )

运行代码

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include <iostream.h>

#include <fstream.h>

#define Num\_city 48

double coord[Num\_city][2]; //坐标矩阵

double d[Num\_city][Num\_city];

double pc=0.2;

double pm=0.1;

int pop[100][Num\_city]; //编码长度为Num\_city位,分别代表Num\_city个城市

int M\_pos1,M\_pos2;

int pop\_size,maxgen;

static void distance(void)

{

int i,j;

double tem;

for(i=0;i<Num\_city;i )

{

for (j=i;j<Num\_city;j )

{ tem=pow((coord[i][0]-coord[j][0]),2) ;pow((coord[i][1]-coord[j][1]),2);

d[i][j]=sqrt(tem);

d[j][i]=d[i][j];

}

d[i][i]=0;

}

}

static void swap(int m,int n,int mn)

{

int s;

s=pop[m][mn];

pop[m][mn]=pop[n][mn];

pop[n][mn]=s;

}

static void swap\_pop(int m,int n,int mn)

{

int s;

s=pop[mn][m];

pop[mn][m]=pop[mn][n];

pop[mn][n]=s;

}

static void local(void)

{

int ij;

M\_pos1=(int)(Num\_city\*(double)rand()/(RAND\_MAX));

do {

M\_pos2=(int)(Num\_city\*(double)rand()/(RAND\_MAX));

} while (M\_pos1==M\_pos2);

if (M\_pos1>M\_pos2)

{

ij=M\_pos1;

M\_pos1=M\_pos2;

M\_pos2=ij;

}

}

static void crossover(void)

{ double cross\_P=0;

int cross\_pos=0;

int i,j;

for(i=0;i<pop\_size-1;i=i+2){

cross\_P=(rand()%101)/100.0;//随机交叉概率

cross\_pos=rand()%Num\_city;//随机交叉位置

if((cross\_P<=pc)&&(cross\_pos!=0)){//若位置为0，则不进行交叉操作

for(j=cross\_pos;j<Num\_city;j ){//从交叉位始进行交叉

swap(i,i 1,j);

}

}

}

}

static void mutation(void)

{ double mut\_p=0.0;

int q,i;

for(i=0;i<pop\_size;i )

{

mut\_p=(rand()%101)/100.0;//随机变异概率

if(mut\_p<=pm)

local();

for (q=M\_pos1 +1;q<(M\_pos2 M\_pos1 1)/2;q )

{

swap\_pop(q,M\_pos2 M\_pos1 1-q,i);

}

}

}

void main()

{

cout<<"\n 遗传算法解决TSP问题\n";

cout<<endl;

cout<<"请输入种群大小(20-100):";

cout<<endl;

cin>>pop\_size;

cout<<"请输入最大世代数(100-300):";

cout<<endl;

cin>>maxgen;

cout<<"城市坐标如下:"<<"\n";

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*编码操作\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

int code[100][Num\_city];//编解码表

double adapt\_best=100;//最佳个体

int best\_individual[Num\_city];//最优个体

double adapt\_sum[200];//存取历代适应度均值

int i=0;

int j=0;

int t=0;

for(i=0;i<maxgen;i ){

adapt\_sum[i]=0;//初始化

}

for(i=0;i<pop\_size;i ){

for(j=0;j<Num\_city;j ){

pop[i][j]=j;

code[i][j]=j;//初始解码模板

//cout<<pop[i][j]<<" ";

}//cout<<"\n";

}

//cout<<"首次编码输出"<<"\n";

for(i=0;i<pop\_size;i ){

for(j=0;j<Num\_city;j ){

t=rand();

pop[i][j]=t%(Num\_city-j);//grefenstette编码

//cout<<pop[i][j]<<" ";

}//cout<<"\n";

}

ifstream infile("data.txt",ios::in|ios::nocreate);

if(!infile){

cout<<"不能打开文件\n";

exit(1);

}

double tem;

int count=0;

double \*dd;

dd=&coord[0][0];

while(infile>>tem){

count ;

if(count%3==1) continue;

\*dd=tem;

dd ;

}

for(i=0;i<Num\_city;i )

cout<<coord[i][0]<<" "<<coord[i][1]<<endl;

distance();

infile.close();

cout<<"距离矩阵如下"<<endl;

for(i=0;i<Num\_city;i )

{

for(j=0;j<Num\_city;j )

cout<<d[i][j]<<" ";

cout<<endl;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*主循环开始\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

for(int G=0;G<maxgen;G )

{

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*交叉操作\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

crossover();

// cout<<"交叉输出"<<"\n";

// for(i=0;i<pop\_size;i ){

// for(j=0;j<Num\_city;j ){

// cout<<pop[i][j]<<" ";

// } cout<<"\n";

// }

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*解码操作\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

int temp[100][Num\_city];

int k=0;

for(i=0;i<pop\_size;i ){

for(j=0;j<Num\_city;j ){

code[i][j]=j;//解码模板

}

}

for(i=0;i<pop\_size;i ){

for(j=0;j<Num\_city;j ){

temp[i][j]=code[i][pop[i][j]];//解码

if(pop[i][j]!=Num\_city-1)//排除第一次解码出现编号最大值最后一个城市的情况

for(k=(pop[i][j] 1);k<Num\_city;k ){

code[i][k-1]=code[i][k];//后位前移，填补抽取留下空缺

}

}

}

for(i=0;i<pop\_size;i ){

for(j=0;j<Num\_city;j ){

pop[i][j]=temp[i][j];

}

}

// cout<<"解码输出"<<"\n";

// for(i=0;i<pop\_size;i ){

// for(j=0;j<Num\_city;j ){

// cout<<pop[i][j]<<" ";

// } cout<<"\n";

// }

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*变异操作\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

mutation();

// cout<<"变异输出"<<"\n";

// for(i=0;i<pop\_size;i ){

// for(j=0;j<=Num\_city;j ){

// cout<<pop[i][j]<<" ";

// } cout<<"\n";

// }

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*计算适应值\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

double adapt[100];//种群适应值之和

int best\_pos=0;//最佳个体在种群中的位置

for(i=0;i<pop\_size;i ){

adapt[i]=0;

for(j=0;j<Num\_city-1;j ){

adapt[i]=adapt[i] d[pop[i][j]][pop[i][j 1]];

}

adapt[i]=adapt[i] d[pop[i][Num\_city-1]][pop[i][0]];

//cout<<adapt[i]<<" ";

}//cout<<"\n";

best\_pos=0;

adapt\_best=100;

for(i=0;i<pop\_size;i ){

if(adapt\_best>adapt[i]){

if(adapt[i]!=0){

adapt\_best=adapt[i];

best\_pos=i;//记录最佳个体在种群中的位置

}

}

}cout<<"\n"<<"第"<<G 1<<"代最佳路线所需行程"<<adapt\_best;

//cout<<"\n"<<"第"<<G 1<<"代最佳路线在种群中的位置"<<best\_pos;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*保优操作\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

for(j=0;j<Num\_city;j ){

best\_individual[j]=0;//初始化

}

//cout<<"\n"<<"当代最佳个体"<<"\n";

for(j=0;j<Num\_city;j ){

best\_individual[j]=pop[best\_pos][j];

//cout<<best\_individual[j]<<" ";

}//cout<<"\n";

double w;

w=0;

for(j=0;j<Num\_city-1;j ){

w=w d[best\_individual[j]][best\_individual[j 1]];

}

w=w d[best\_individual[Num\_city-1]][best\_individual[0]];

//cout<<"\n"<<"当前最佳路线里程为"<<w<<" ";

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*选择操作\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

double ada\_sum=0;

double ada\_temp=0;//初始化累加值为0

int new\_pop[100][Num\_city];//存取新种群

double r=0;

for(i=0;i<pop\_size;i ){

ada\_sum =adapt[i];//计算种群中个体适应度之和

}//cout<<ada\_sum<<"\n";

adapt\_sum[G]=ada\_sum/pop\_size;//计算种群历代平均适应度

for(i=0;i<pop\_size-1;i ){

r=(rand()%1001)/1000.0;//产生0～1间的小数

r=r\*(pop\_size\*100-ada\_sum);//cout<<r<<"\n";

j=0;

for(ada\_temp=0;ada\_temp<=r;ada\_temp=ada\_temp (100-adapt[j])){

j=j ;

}

j=j-1;//退出循环时的j值即为被选择的个体序号

//cout<<j<<"\n";

for(k=0;k<Num\_city;k ){

new\_pop[i][k]=pop[j][k];//取出被选择的个体存入新种群

}

}

for(j=0;j<Num\_city;j ){

new\_pop[pop\_size-1][j]=best\_individual[j];//最优解存至最后一个个体

}

//cout<<"选择输出"<<"\n";

for(i=0;i<pop\_size;i ){

for(j=0;j<Num\_city;j ){

pop[i][j]=new\_pop[i][j];

//cout<<pop[i][j]<<" ";

}//cout<<"\n";

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*编码操作\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

int l=Num\_city-1;

int m=0;

for(i=0;i<pop\_size;i ){

for(j=0;j<Num\_city;j ){

code[i][j]=j;//编码模板

}

}

for(i=0;i<pop\_size;i ){

for(j=0;j<Num\_city;j ){

for(k=0;k<=l;k ){

if(pop[i][j]==code[i][k]){//查找pop中元素在码表code中的位置

if(k!=Num\_city-1){

temp[i][j]=k;//依照模板进行编码

for(m=k;m<=l-1;m ){

code[i][m]=code[i][m 1];//后位前移，填补抽取留下空缺

}

l=l-1;

}

else {temp[i][j]=k;

l=l-1;

}

}

}

}

l=Num\_city-1;//复原l值

}

//cout<<"编码输出"<<"\n";

for(i=0;i<pop\_size;i ){

for(j=0;j<Num\_city;j ){

pop[i][j]=temp[i][j];

//cout<<pop[i][j]<<" ";

}//cout<<"\n";

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*主循环结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*输出操作\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

cout<<"\n"<<"\n";

cout<<"历代种群平均适应度"<<"\n";

for(i=0;i<maxgen;i ){

cout<<adapt\_sum[i]<<" ";

}

cout<<"\n"<<"\n"<<"最佳路线为"<<"\n";

for(j=0;j<Num\_city;j ){

cout<<best\_individual[j]<<"-";

}

cout<<best\_individual[0];

cout<<"\n"<<"\n"<<"最佳路线所需行程"<<adapt\_best<<"(单位：1000km)"<<"\n";

fstream result;

result.open("输出结果.txt", ios::app);

if(!result)

{

cout<<"can't open the <输出结果.log> file!\n";

exit(0);

}

result<<"\*-------------------------------------------------------------------------\*"<<endl;

result<<"最优路线:"<<endl;

for(j=0;j<Num\_city;j ){

result<<best\_individual[j]<<"-";

}

result<<best\_individual[0];

result<<"\n"<<"\n"<<"最佳路线所需行程"<<adapt\_best<<"(单位：1000km)"<<"\n";

result<<"\*-------------------------------------------------------------------------\*"<<endl<<endl;

result.close();

}

运行截图

