# 遗传算法寻优小结

## 程序要点：

1、基因类

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 类 名：CGenetic

说 明：基因类。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class CGenetic

{

public:

bool IntCode[CodeLen]; //整数编码

bool FloatCode[CodeLen]; //小数编码

bool flag; //正负符号，1为正数，0为负数

double fitness; //适应度

public:

double GetValue(); //解码

void SetValue(double); //编码

void OutPutCode(); //输出编码和实数值

CGenetic() //构造函数

{

SetValue(0);

fitness=0;

}

CGenetic(double v) //构造函数

{

SetValue(v);

fitness=0;

}

CGenetic(const CGenetic &g); //复制构造函数

const CGenetic &operator=(const CGenetic &g);//重载=运算符

};

1.1类中，用整数编码和小数编码来对一个实数进行编码，各编码长度由CodeLen指定。

1.2复制构造函数的写法：

CGenetic::CGenetic(const CGenetic &g)

{

flag=g.flag;

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

{

IntCode[i]=g.IntCode[i];

FloatCode[i]=g.FloatCode[i];

}

fitness=g.fitness;

}

1.3重载等号运算符的写法：

const CGenetic& CGenetic::operator=(const CGenetic &g)

{

if (this==&g)

return \*this;

flag=g.flag;

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

{

IntCode[i]=0;

FloatCode[i]=0;

}

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

{

IntCode[i]=g.IntCode[i];

FloatCode[i]=g.FloatCode[i];

}

fitness=g.fitness;

return \*this;

}

2、函数指针的定义及使用

typedef double(\*f)(double d);

该语句定义了一个函数指针f，该指针有一个实型的参数，返回值也是实型。

例如，有一个名为fit的函数：

double fit(double d)

{

return -(d-120.2568)\*(d-120.2568);

}

该函数满足以上定义的函数指针的格式，在以下函数中：

void Fitness(f func,CGenetic &g)，该函数有两个参数，一个是f类型的函数指针，一个为CGenetic变量，在调用时，可以：

Fitness(fit,g);

3、遗传算法的原理不在赘述，在这里给出示例代码，对于实际问题，需要重新对编码、解码、及适应度函数做相应的修改。本示例是对求其最大值，显然理论最优解为x=120.2568。

4、随机数

#define randf() (rand()/(double)RAND\_MAX) //产生0～1间的实型随机数

#define randr(mini,maxi) (rand()%(int(maxi-mini+1))+int(mini)) //产生mini～maxi间整型随机数

在主程序开始前，需要初始化随机种子：

srand((unsigned)time(NULL));

## 附录：

### Global.h文件

#pragma once

#include "stdafx.h"

#include <stdlib.h>

#include <iostream>

#include <math.h>

using namespace std;

#define CodeLen 8

#define randf() (rand()/(double)RAND\_MAX)

#define randr(mini,maxi) (rand()%(int(maxi-mini+1))+int(mini))

typedef double(\*f)(double d); //函数指针

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 类 名：CGenetic

说 明：基因类。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class CGenetic

{

public:

bool IntCode[CodeLen]; //整数编码

bool FloatCode[CodeLen]; //小数编码

bool flag; //正负符号，1为正数，0为负数

double fitness; //适应度

public:

double GetValue(); //解码

void SetValue(double); //编码

void OutPutCode(); //输出编码和实数值

CGenetic() //构造函数

{

SetValue(0);

fitness=0;

}

CGenetic(double v) //构造函数

{

SetValue(v);

fitness=0;

}

CGenetic(const CGenetic &g); //复制构造函数

const CGenetic &operator=(const CGenetic &g);//重载=运算符

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 函数名：Mutate

说 明：变异算子。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CGenetic Mutate(CGenetic g,double p=0.1);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 函数名：Copulate

说 明：交叉算子。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Copulate(CGenetic\* farther,CGenetic \*\*children, int n=2);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 函数名：Fitness

说 明：适应度算子。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Fitness(f func,CGenetic &g);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 函数名：Select

说 明：选择算子。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Select(CGenetic\* Group,CGenetic \*\*SelectGroup,int M,int N);

### Global.cpp文件：

#pragma once

#include "stdafx.h"

#include "Global.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 函数名：CGenetic

返 回：无

说 明：复制构造函数。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CGenetic::CGenetic(const CGenetic &g)

{

flag=g.flag;

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

{

IntCode[i]=g.IntCode[i];

FloatCode[i]=g.FloatCode[i];

}

fitness=g.fitness;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 函数名：operator=

返 回：CGenetic &

说 明：重载等号运算符。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

const CGenetic& CGenetic::operator=(const CGenetic &g)

{

if (this==&g)

return \*this;

flag=g.flag;

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

{

IntCode[i]=0;

FloatCode[i]=0;

}

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

{

IntCode[i]=g.IntCode[i];

FloatCode[i]=g.FloatCode[i];

}

fitness=g.fitness;

return \*this;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 函数名：SetValue

参 数：v

说 明：编码，将一个浮点型数据进行编码，表示成二进制数，整数部分编制

在IntCode中，小数部分编制在FloatCode中，从左往右都是高位到低

位，编码位数为CodeLen。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void CGenetic::SetValue(double v)

{

//符号

if (v>=0)

flag=1;

else

flag=0;

//整数部分

v=fabs(v);

int intnum=int(v);

int subnum1=0;

int i=0;

for(i=CodeLen-1;i>=0;i--)

IntCode[i]=0;

for(i=CodeLen-1;i>=0;i--)

{

//求第i们前的高位数合

if(subnum1+pow(2.0,i)<=intnum)

{

IntCode[i]=1;

subnum1+=pow(2.0,i);

}

else

{

IntCode[i]=0;

}

}

//小数部分

double floatnum=v-int(v);

float subnum2=0.0;

i=0;

for(i=0;i<=CodeLen-1;i++)

FloatCode[i]=0;

for(i=0;i<=CodeLen-1;i++)

{

if(subnum2+pow(2.0,-1\*(i+1))<=floatnum)

{

FloatCode[CodeLen-i-1]=1;

subnum2+=pow(2.0,-1\*(i+1));

}

else

{

FloatCode[CodeLen-i-1]=0;

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 函数名：GetValue

返 回：浮点型数据

说 明：解码，将类内部的整数编码与小数编码解码为浮点数据。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double CGenetic::GetValue()

{

double IntNum=0,FloatNum=0,BackNum=0;

int i=0;

//整数

for ( i=CodeLen-1;i>=0;i--)

{

if (IntCode[i])

IntNum+=pow(2.0,i);

}

//小数

for (i=CodeLen-1;i>=0;i--)

{

if (FloatCode[i])

FloatNum+=pow(2.0,-1\*(CodeLen-i));

}

//符号

BackNum=IntNum+FloatNum;

if (!flag)

BackNum\*=-1;

return BackNum;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 函数名：OutPutCode

返 回：无

说 明：输出编码和实数值。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void CGenetic::OutPutCode()

{

cout<<"符号："<<flag<<endl;

cout<<"整数编码：";

for (int i=CodeLen-1;i>=0;i--)

cout<<IntCode[i]<<" ";

cout<<endl;

cout<<"小数编码：";

for (int i=CodeLen-1;i>=0;i--)

cout<<FloatCode[i]<<" ";

cout<<endl;

cout<<"十进制数："<<GetValue()<<endl;

cout<<"适应度："<<fitness<<endl;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 函数名：Mutate

参 数：

CGentic &g:要变异的基因

double p :变异概率

返 回：变异后的基因

说 明：输出编码和实数值。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CGenetic Mutate(CGenetic g,double p)

{

CGenetic gg=g;

if (randf()<p) //变异

{

//随机产生变异位

int i=randr(0,2\*CodeLen-1);

if (i>=8)//小数变异

{

gg.FloatCode[i-CodeLen]=!gg.FloatCode[i-CodeLen];

}

else//整数变异

{

gg.IntCode[i]=!gg.IntCode[i];

}

return gg;

}

else //不变异

return g;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 函数名：Copulate

参 数：

CGentic \*farther:输入值，父代基因，共2个基因元素的数组

CGentic \*\*children:返回值，子代基因，指向2个基因元素的数组

int n:交叉的位数

说 明：交叉算子。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Copulate(CGenetic\* farther,CGenetic \*\*children,int n)

{

//随机产生交叉位

int i=randr(0,2\*CodeLen-1-n);

for (int i=0;i<2;i++)

(\*children)[i]=farther[i];

for (int k=i;k<i+n;k++)

{

if (k>CodeLen-1)//小数交叉

{

bool t=(\*children)[0].FloatCode[k-CodeLen];

(\*children)[0].FloatCode[k-CodeLen]=(\*children)[1].FloatCode[k-CodeLen];

(\*children)[1].FloatCode[k-CodeLen]=t;

}

else //整数交叉

{

bool t=(\*children)[0].IntCode[k];

(\*children)[0].IntCode[k]=(\*children)[1].IntCode[k];

(\*children)[1].IntCode[k]=t;

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 函数名：Fitnesss

参 数：

f func:输入值，求适应度的具体函数

CGentic g:输入值，输入基因

说 明：计算输入基因的适应度。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Fitness(f func,CGenetic &g)

{

double t=g.GetValue();

g.fitness=func(t);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 函数名：Select

参 数：

CGenetic\* Group:输入值，备选的基因

CGenetic \*\*SelectGroup:返回值，选出的基因

int M:输入值，备选基因的个数

int N:输入值，选出基因的个数

说 明：对基因进行优选，基于排序法。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Select(CGenetic\* Group,CGenetic \*\*SelectGroup,int M,int N)

{

//希尔排序

int i,j,flag,gap=M;

CGenetic tmp;

while(gap>1)

{

gap=gap/2;

do

{

flag=0;

for (i=0;i<M-gap;i++)

{

j=i+gap;

if (Group[i].fitness<Group[j].fitness)

{

tmp=Group[i];

Group[i]=Group[j];

Group[j]=tmp;

flag=1;

}

}

} while (flag!=0);

}

//将前N项拷贝到选择出来的数组中

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

(\*SelectGroup)[i]=Group[i];

}

### GADemo.cpp文件：

// GADemo.cpp : 定义控制台应用程序的入口点。

//

#include "stdafx.h"

#include <stdlib.h>

#include <iostream>

#include <math.h>

#include <time.h>

#include "Global.h"

#define MAX\_TIMES 50 //最大迭代次数

#define POPULATION 10 //种群数量

#define p 0.2 //变异概率

double fit(double d)

{

return -(d-120.2568)\*(d-120.2568);

}

int \_tmain(int argc, \_TCHAR\* argv[])

{y

srand((unsigned)time(NULL));

CGenetic \*farthergen=new CGenetic[POPULATION];

CGenetic \*childrengen=new CGenetic[POPULATION];

CGenetic \*totalgen=new CGenetic[2\*POPULATION];

//初始化种群

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

{

double t1=randr(0,255);

double t2=randf();

farthergen[i].SetValue(t1+t2);

Fitness(fit,farthergen[i]);

farthergen[i].OutPutCode();

}

//开始遗传算法

for (int t=0;t<MAX\_TIMES;t++)

{

//交叉，随机选择父代的基因进行交叉

for (int k=0;k<POPULATION-2;k+=2)

{

int i=0,j=0;

i=randr(0,POPULATION-1);

do

{

j=randr(0,POPULATION-1);

} while (j==i);

CGenetic \*farther=new CGenetic[2];

CGenetic \*children=new CGenetic[2];

farther[0]=farthergen[i];

farther[1]=farthergen[j];

Copulate(farther,&children);

Fitness(fit,children[0]);

Fitness(fit,children[1]);

childrengen[k]=children[0];

childrengen[k+1]=children[1];

delete [] farther;

delete [] children;

}

//变异

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

{

childrengen[i]=Mutate(childrengen[i],p);

Fitness(fit,childrengen[i]);

}

//选择

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

totalgen[i]=farthergen[i];

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

totalgen[i+POPULATION]=childrengen[i];

Select(totalgen,&farthergen,2\*POPULATION,POPULATION);

//输出最优解

farthergen[0].OutPutCode();

}

delete [] farthergen;

delete [] childrengen;

delete [] totalgen;

system("pause");

return 0;

}