**实验五 遗传算法**

**一、实验目的**

理解遗传算法的基本操作和思想。

**二、实验分析**

遗传算法也称进化算法 。 遗传算法是受达尔文的进化论的启发，借鉴生物进化过程而提出的一种启发式搜索算法。

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。

**选择**：选择一些染色体来产生下一代。

**变异**：在繁殖过程，新产生的染色体中的基因会以一定的概率出错，称为变异。

**交叉**：2条染色体交换部分基因，来构造下一代的2条新的染色体。

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”。

**三、实验内容**

**源代码：**

#include <windows.h>

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

#include<time.h>

#define S 5 //种群的规模

#define Pc 0.8 //交叉概率

#define Pm 0.05 //突变概率

#define KW 1000 //背包最大载重1000

#define N 30 //物体总数

#define T 800 //最大换代数

#define ALIKE 0.05 //判定相似度

/\*数据集一\*/

int stop=0; //初始化函数中初始化为所有价值之和

int t=0; //目前的代数

int value[]={

220,208,198,192,180,180,165,162,160,158,155,130,125,122,120,118,115,110,105,101,100,100,98,96,95,90,88,82,80,77,75,73,72,70,69,66,65,63,60,58,56,50,30,20,15,10,8,5,3,1};

int weight[]={

80,82,85,70,72,70,66,50,55,25,50,55,40,48,50,32,22,60,30,32,40,38,35,32,25,28,30,22,25,30,45,30,60,50,20,65,20,25,30,10,20,25,15,10,10,10,4,4,2,1};

/\*数据集二

int stop=0; //初始化函数中初始化为所有价值之和

int t=0; //目前的代数

int value[]={

220,208,198,192,180,180,165,162,160,158,155,130,125,122,120,118,115,110,105,101,100,100,98,96,95,90,88,82,80,77,75,73,72,70,69,66,65,63,60,58,56,50,30,20,15,10,8,5,3,1};

int weight[]={

80,82,85,70,72,70,66,50,55,25,50,55,40,48,50,32,22,60,30,32,40,38,35,32,25,28,30,22,25,30,45,30,60,50,20,65,20,25,30,10,20,25,15,10,10,10,4,4,2,1};

\*/

/\*数据集三

int stop=0; //初始化函数中初始化为所有价值之和

int t=0; //目前的代数

int value[]={

597,596,593,586,581,568,567,560,549,548,547,529,529,527,520,491,482,478,475,475,466,462,459,458,454,451,449,443,442,421,410,409,395,394,390,377,375,366,361,347,334,322,315,313,311,309,296,295,294,289,285,279,277,276,272,248,246,245,238,237,232,231,230,225,192,184,183,176,171,169,165,165,154,153,150,149,147,143,140,138,134,132,127,124,123,114,111,104,89,74,63,62,58,55,48,27,22,12,6,250};

int weight[]={

54,183,106,82,30,58,71,166,117,190,90,191,205,128,110,89,63,6,140,86,30,91,156,31,70,199,142,98,178,16,140,31,24,197,101,73,16,73,2,159,71,102,144,151,27,131,209,164,177,177,129,146,17,53,64,146,43,170,180,171,130,183,5,113,207,57,13,163,20,63,12,24,9,42,6,109,170,108,46,69,43,175,81,5,34,146,148,114,160,174,156,82,47,126,102,83,58,34,21,14};

\*/

struct individual //个体结构体

{

bool chromsome[N]; //染色体编码

double fitness; //适应度//即本问题中的个体所得价值

double weight; //总重量

};

int best=0;

int same=0;

individual X[S],Y[S],bestindividual;

int comp(individual bestindividual,individual temp)//比较函数

{

int fit=0,w=0;//第一种不变:操作后不满足重量函数，第二种:操作后适应度小于操作前

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

fit+=temp.chromsome[i]\*value[i];

w+=temp.chromsome[i]\*weight[i];

}

if(w>KW)return -1;

return (bestindividual.fitness>fit?-1:1);//如果小于原来值或者不满足重量函数，则返回-1

}

void Checkalike(void)

{

int i=0,j=0;

for( i=0;i<S;i++)//相似度校验

{

for(j=0;j<N;j++)

{

bool temp=X[i].chromsome[j];

for(int k=1;k<S;k++)

{

if(temp!=X[k].chromsome[j])

break;

}

}

if(j==N)

same++;

}

if(same>N\*ALIKE)//大于ALIKE作为判定为早熟

{

int minindex=0;

for(int n=0;n<S;n++)

if(X[n].fitness<X[minindex].fitness)

minindex=n;//确定最小

for (j=0; j<N;j++)//重新生成

{

bool m=(rand()%10<5)?0:1;

X[minindex].chromsome[j]=m;

X[minindex].weight+=m\*weight[j];//个体的总重量

X[minindex].fitness+=m\*value[j]; //个体的总价值

}

}

}

void GenerateInitialPopulation(void)//初始种群,保证每个值都在符合条件的解

{

int i=0,j=0; bool k;

for(i=0;i<N;i++)stop+=value[i];//设置理论最优值

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

{

int w=0,v=0;

for (j=0; j<N;j++)

{

k=(rand()%10<5)?0:1;

X[i].chromsome[j]=k;

w+=k\*weight[j];//个体的总重量

v+=k\*value[j]; //个体的总价值

}

if(w>KW) i--; //如果不是解，重新生成

else

{

X[i].fitness=v;

X[i].weight=w;

if(v==stop)

{

bestindividual=X[i];

return;

}//这种情况一般不会发生

}

}

}

void CalculateFitnessValue()

{

int i=0,j=0;

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

{

int w=0,v=0;

for (j=0; j<N;j++)

{

w+=X[i].chromsome[j]\*weight[j];//个体的总重量

v+=X[i].chromsome[j]\*value[j]; //个体的总价值

}

X[i].fitness=v;

X[i].weight=w;

if(v==stop)

{

bestindividual=X[i];

return;

}//符合条件情况下最优解这种情况一般不会发生

if(w>KW) X[i]=bestindividual;//如果不是解，找最好的一个解代之

}

}

void SelectionOperator(void)

{

int i, index;

double p, sum=0.0;

double cfitness[S];//选择、累积概率

individual newX[S];

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

sum+=X[i].fitness;//适应度求和

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

cfitness[i]=X[i].fitness/sum; //选择概率

for (i=1;i<S; i++)

cfitness[i]=cfitness[i-1]+cfitness[i];//累积概率

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

{

p=(rand()%1001)/1000.0;//产生一个[0,1]之间的随机数

index=0;

while(p>cfitness[index])//轮盘赌进行选择

{

index++;

}

newX[i]=X[index];

}

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

X[i]=newX[i];//新的种群

}

void CrossoverOperator(void)//交叉操作

{

int i=0, j=0,k=0;individual temp;

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

{

int p=0,q=0;

do

{

p=rand()%S;//产生两个[0，S]的随机数

q=rand()%S;

}while(p==q);

int w=1+rand()%N;//[1,N]表示交换的位数

double r=(rand()%1001)/1000.0;//[0,1]

if(r<=Pc)

{

for(j=0;j<w;j++)

{

temp.chromsome[j]=X[p].chromsome[j];//将要交换的位先放入临时空间

X[p].chromsome[j]=X[q].chromsome[j];

X[q].chromsome[j]=temp.chromsome[j];

}

}

if(p==best)

if(-1==comp(bestindividual,X[p]))//如果变异后适应度变小

X[p]=bestindividual;

if(q==best)

if(-1==comp(bestindividual,X[q]))//如果变异后适应度变小

X[q]=bestindividual;

}

}

void MutationOperator(void)

{

int i=0, j=0,k=0,q=0;

double p=0;

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

{

for (j=0; j<N; j++)

{

p=(rand()%1001)/1000.0;

if (p<Pm)//对每一位都要考虑

{

if(X[i].chromsome[j]==1)X[i].chromsome[j]=0;

else X[i].chromsome[j]=1;

}

}

if(i==best)

if(-1==comp(bestindividual,X[i]))//如果变异后适应度变小

X[i]=bestindividual;

}

}

void FindBestandWorstIndividual(void)

{

int i;

bestindividual=X[0];

for (i=1;i<S; i++)

{

if (X[i].fitness>bestindividual.fitness)

{

bestindividual=X[i];

best=i;

}

}

}

int main()

{

DWORD start, stop;

srand((unsigned)time(0));

t=0;

GenerateInitialPopulation(); //初始群体包括产生个体和计算个体的初始值

while (t<=T)

{

FindBestandWorstIndividual(); //保存当前最优解

SelectionOperator(); //选择

CrossoverOperator(); //交叉

MutationOperator(); //变异

Checkalike(); //检查相似度

CalculateFitnessValue(); //计算新种群适应度

t++;

}

FindBestandWorstIndividual(); //找到最优解

printf(" 物品价值：");

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

{

printf(" %d ",value[k]);

}

printf("\n");

printf(" 物品重量:");

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

{

printf(" %d ",weight[k]);

}

printf("\n");

printf("背包容量 %d\n",1000); //输出最优值

printf("-----------------------------\n");

printf("最优价值 %f\n",bestindividual.fitness); //输出最优值

printf("对应重量 %f\n",bestindividual.weight); //对应重量

printf("线性解：");

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

{

if(bestindividual.chromsome[k]==1){ //输出最优解

printf(" %d ",1);

}else{

printf(" %d ",0);

}

}

printf("\n");

printf("\n");

system("pause");

return 0;

}

**运行结果：**

第一个数据集



第二个数据集



第三个数据集

