GA.cpp

// GA.cpp : 定義主控台應用程式的進入點。

//

#include "stdafx.h"

#include <time.h>

#include "GA.h"

int main(int argc,char \*\*argv)

{

int i, j;

srand((unsigned)time(NULL));

initialize(); //初始化

for (i = 0; i < ITERA\_CNT; i++)

{

reproduction(); //選擇(分配式)

//reproduction\_rnd(); //選擇(隨機式),收斂速度慢

crossover(); //交配

mutation(); //突變

}

printf("\n==================================\n");

printf("%3d times...\n", i);

for (j = 0; j < POPULATION\_CNT; j++)

{

printf("(%5.2lf,%5.2lf)", population[j].dec\_value, population[j].fitness);

if (j % 4 == 3) printf("\n");

}

printf("\n==================================\n");

printf(" ever find best gene:");

printf("(%5.2lf,%5.2lf)\n", best\_gene.dec\_value, best\_gene.fitness);

system("pause");

return 0;

}

GA.h

#pragma once

#ifndef \_\_GA\_\_

#define \_\_GA\_\_

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include<string.h>

#define GENETIC\_LENGTH 4 //基因長度

#define POPULATION\_CNT 10 //母群數量

#define ITERA\_CNT 100 //迭代次數

#define CROSSOVER\_RATE 0.5 //交配率

#define MUTATION\_RATE 0.1 //突變率

//--------------------------------------

//定義母體結構

typedef struct parent\_t {

int genes[GENETIC\_LENGTH];

double fitness;

double dec\_value;

}praent\_t;

//--------------------------------------

//GAPosRand(): 隨機取得突變位置

#define GAPosRand() (rand()%GENETIC\_LENGTH)

//BinaryRand(): 隨機產生/1 的整數

#define BinaryRand() (rand()%2)

//SRand(): 隨機產生~1的整數

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

//-----------------------------------------

void initialize(); //進行初始化

void reproduction(); //複製,輪盤式選擇(分配式),決定每個母體複製到交配池的個數

void reproduction\_rnd(); //複製,輪盤式選擇(隨機式)

void crossover(); //交配,交配池中的個體交配,[單點交配,雙點交配,mask]

void mutation(); //突變,逐一bit慢慢確認突變

void cal\_fitness(parent\_t \*x); //計算基因所對應的適應值

void cal\_xvalue(parent\_t \*x); //計算基因對應之進制值

//-----------------------------------------

parent\_t population[POPULATION\_CNT]; //母體數量

parent\_t pool[POPULATION\_CNT]; //交配池

parent\_t best\_gene; //從以前到現在最好的基因

//------------------------------------------

//binary 2 dec,將染色體中的genes轉換為十進制

void cal\_xvalue(parent\_t \*x)

{

int i, dec = 0;

for (i = 0; i < GENETIC\_LENGTH; i++)

{

if (x->genes[i] == 1) dec = dec + (0x01 << i);

}

x->dec\_value = (double)dec;

}

//------------------------------------------

//------------------------------------------

//適應函式,此設為f(x)=x\*x, x為染色體中的十進制, 即dec\_value

void cal\_fitness(parent\_t \*x)

{

double i = x->dec\_value;

x->fitness = i\*i\*i\*i;

}

//------------------------------------------

//------------------------------------------

//初始化

void initialize()

{

int i, j;

for (i = 0; i < POPULATION\_CNT; i++)

{

for (j = 0; j < GENETIC\_LENGTH; j++)

{

population[i].genes[j] = BinaryRand(); //每個母體都是隨機給 /1

}

cal\_xvalue(&population[i]); //計算母體基因之進制值

cal\_fitness(&population[i]); //計算母體對應之適應值

if (i == 0)

{

memcpy(&best\_gene, &population[i], sizeof(parent\_t));

}

else if (population[i].fitness>best\_gene.fitness)

{

memcpy(&best\_gene, &population[i], sizeof(parent\_t));

}

}

}

//-------------------------------------------

//-------------------------------------------

//複製,輪盤式選擇(分配式)

void reproduction()

{

int i, j, cnt, has\_copy = 0;

int Slack = POPULATION\_CNT;//還剩幾個可複製

int pos1, pos2;

double fitness\_sum = 0.0;

for (int i = 0; i < POPULATION\_CNT; i++) //計算所有適應值總和

{

fitness\_sum += population[i].fitness;

}

for (i = 0; i < POPULATION\_CNT && Slack != 0; i++) //計算每個母體應複製幾個到交配池中,並直接作複製

{

cnt = (int)(population[i].fitness / fitness\_sum + 0.5); //計算複製個數,四捨五入

if (cnt > Slack) cnt = Slack;

for (j = 0; j < cnt; ++j, ++has\_copy)

{

memcpy(&best\_gene, &population[i], sizeof(parent\_t));

}

Slack -= cnt;

}

while (has\_copy < POPULATION\_CNT) //若還有沒複製完的

{

pos1 = rand() % POPULATION\_CNT; //隨機挑兩條不同的染色體出來

do

{

pos2 = rand() % POPULATION\_CNT;

} while (pos1 == pos2);

if (population[pos1].fitness>population[pos2].fitness) i = pos1; //比較好的那條丟到交配池

memcpy(&pool[has\_copy++], &population[i], sizeof(parent\_t));

}

}

//--------------------------------------------

//--------------------------------------------

//複製,輪盤式選擇(隨機式)

void reproduction\_rnd()

{

int i, pos;

double fitness\_sum = 0.0; //適應值總和

double column\_prob[POPULATION\_CNT]; //累計機率

double prob; //隨機機率

for (i = 0; i < POPULATION\_CNT; i++)

{

fitness\_sum += population[i].fitness;

}

column\_prob[0] = population[0].fitness / fitness\_sum;

for (i = 0; i < POPULATION\_CNT; ++i)

{

column\_prob[i] = column\_prob[i - 1] + population[i].fitness / fitness\_sum;

}

for (i = 0; i < POPULATION\_CNT; ++i)

{

prob = SRand(); //產生亂數

for (pos = 0; pos < POPULATION\_CNT; ++pos)

{

if (prob >= column\_prob[pos]) break;

}

memcpy(&pool[i], &population[pos], sizeof(parent\_t));

}

}

//--------------------------------------------

//--------------------------------------------

//交配

void crossover()

{

int i, itera;

int cnt = 0;

int pos = 0;

int p1, p2;

double crossover\_if;

for (itera = 0; itera < POPULATION\_CNT; itera++)

{

p1 = rand() % POPULATION\_CNT;//隨機選兩個個體

do

{

p2 = rand() % POPULATION\_CNT;

} while (p2 == p1);

crossover\_if = SRand(); //決定是否交配

if (crossover\_if > CROSSOVER\_RATE)

{

memcpy((void \*)&population[cnt++], (void \*)&pool[p1], sizeof(parent\_t));

memcpy((void \*)&population[cnt++], (void \*)&pool[p2], sizeof(parent\_t));

}

else

{

do

{

pos = GAPosRand(); //單點交配,交配完後丟回母體

} while (pos == 0);

for (i = 0; i < pos; i++) //crossover

{

population[cnt].genes[i] = pool[p1].genes[i];

population[cnt + 1].genes[i] = pool[p2].genes[i];

}

cnt += 2; //以複製完兩條

}

}

}

//--------------------------------------------

//--------------------------------------------

//突變

void mutation()

{

int i;

int pos;

for (i = 0; i < POPULATION\_CNT; i++)

{

double mutation\_if = SRand();

if (mutation\_if <= MUTATION\_RATE)

{

pos = GAPosRand(); //突變位置

population[i].genes[pos] = 1 - population[i].genes[pos];

}

//突變完後再算一次母體適應值

cal\_xvalue(&population[i]); //先計算基因對應之x值

cal\_fitness(&population[i]); //再將進制x值帶入適應函式

//再更新best\_gene

if (i == 0)

{

memcpy(&best\_gene, &population[i], sizeof(parent\_t));

}

else if (population[i].fitness>best\_gene.fitness)

{

memcpy(&best\_gene, &population[i], sizeof(parent\_t));

}

}

}

//--------------------------------------------

#endif // !\_\_GA\_\_