Nume si prenume: Paduraru Andra – Elena

Grupa: II A4

Tema: T2

**DESCRIERE PROBLEMA:**

Odata introdus numele functiei se va apela calculul pentru algoritmul genetic pentru functia respectiva. Se alege o variabila „best” in care se va calcula cel mai bun individ din ultima generatie si se va initializa cu un numar destul de mare, aceasta reprezentand solutia si apoi se va face generarea random a matricei populatiei pentru fiecare rulare, care va fi reprezentata de numerele 0 si 1.

Urmatorul pas este alegerea unui best in care se va memora rezultatul evaluarii uneia dintre cele patru functii, doar daca rezultatul va fi mai mic decat best-ul initial initializat. Evaluarea se va face decondand bitii in numere reale iar apoi calculandu-se minimul pentru functie, daca exista.

Se alege conditia de stop si se incepe selectia pana cand conditia nu va mai fi indeplinita. Se initializeaza fitness-ul total si probabilitatea selectiei, se evalueaza populatia iar apoi se vor calcula initializarile. Selectia in sine reprezinta generarea uniforma a lui „r” si selectarea pentru supravietuire a individului „j” pentru care conditia selection\_probab[j] < r && r <= selection\_probab[j + 1] este indeplinita.

Se actualizeaza populatia si apoi se apeleaza functia de recombinare care are rolul de a calcula mutatia si/sau cross\_over. Ulterior se actualizeaza best-ul si se va afisa. Functia de cross\_over va functiona pe principiul incrucisarea cu punct de taiere aleator.

**PSEUDOCOD:**

dejong (float\* res)

//calcul functie

schwefel (float\* res)

//calcul functie

rastrigin (float\* res)

//calcul functie

sixhumpcamelback (float\* res)

//calcul functie

f(bool\* a, float\* res)

//se utilizeaza reprezentarea binara pentru functiile de optimizare numerica

eval\_popfitness (int number)

//functia fitness pentru minimizare a populatiei

for (int i = 0; i < population\_size; i++)

{

f (population[i], res\_fit);

population\_fitness[i] = 1 / functie\_calcul (res\_fit); //unde functia calcul este una din cele 4 functii initiale

}

eval\_pop (int number)

//evalueaza populatia si returneaza un minim al uneia din cele 4 functii

if (functie\_calcul (res\_eval) < minim)

minim = functie\_calcul (res\_eval);

cross\_over (int i1, int i2)

//incruciseaza vectorii utilizand un punct de taiere aleator

recombine ()

//se va calcula mutatia si/sau cross\_over

//pentru mutatie

for (int i = 0; i < population\_size; i++)

{

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

{

if ((1.0 \* rand () / RAND\_MAX) < mutation\_probab)

{

population[i][j] = 1 - population[i][j];

}

}

}

GeneticAlgorithm (int number)

INITIALIZARE **begin**

***t* := 0**

genereaza *P(t)*

evalueaza *P(t)*

ITERARE **while** (not CONDITIE\_OPRIRE)  **do begin**

*t* := *t* + 1

selecteaza *P(t)* din *P(t-1)*

recombina *P(t)*

evalueaza *P(t)*

**end**

int main ()

{

cout << fixed;

//se citeste numele functiei

//se apeleaza GeneticAlgorithm in functie de functia aleasa

}

**COD IN C++:**

#include <bits/stdc++.h>

#define MAX\_ARRAY 25

#define dimensions 5

#define length 8

#define runs 15

#define sizes (dimensions \* length)

#define population\_size 10

#define cross\_probab 0.3

#define mutation\_probab 0.01

using namespace std;

float A, B;

bool population[population\_size][sizes], new\_pop[population\_size][sizes];

float population\_fitness[population\_size], selection\_probab[population\_size + 1];

float dejong (float\* res)

{

float result = 0;

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

{

result = result + res[i] \* res[i];

}

return result;

}

float schwefel (float\* res)

{

float result = 0;

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

{

result = result - res[i] \* sin(sqrt(abs(res[i])));

}

return result;

}

float rastrigin (float\* res)

{

float result = 10\*dimensions;

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

{

result = result + res[i]\*res[i] - 10\*cos(2\*3.1415\*res[i]);

}

return result;

}

float sixhumpcamelback (float\* res)

{

float result = (4-2.1 \* res[0]\*res[0] + res[0]\*res[0]\*res[0]\*res[0]/3)\*res[0]\*res[0] + res[0]\*res[1] + (-4 + 4\*res[1]\*res[1])\*res[1]\*res[1];

return result;

}

void f (bool\* a, float\* res)

{

//reprezentare binara pt probleme de optimizare numerica

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

{

res[i] = 0;

int p = 1;

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

{

res[i] = res[i] + p\*a[i\*length+j];

p = p\*2;

}

res[i] = res[i]\* (B - A);

res[i] = res[i]/ p;

res[i] = res[i] + A;

}

}

void eval\_popfitness (int number)

{

float res\_fit[sizes];

//se calculeaza prin inversarea functiei, functii pozitive ce necesita minimizare

if (number == 1)

{

for (int i = 0; i < population\_size; i++)

{

f (population[i], res\_fit);

population\_fitness[i] = 1 / dejong (res\_fit);

}

}

if (number == 2)

{

for (int i = 0; i < population\_size; i++)

{

f (population[i], res\_fit);

population\_fitness[i] = 1 / schwefel (res\_fit);

}

}

if (number == 3)

{

for (int i = 0; i < population\_size; i++)

{

f (population[i], res\_fit);

population\_fitness[i] = 1 / rastrigin (res\_fit);

}

}

if (number == 4)

{

for (int i = 0; i < population\_size; i++)

{

f (population[i], res\_fit);

population\_fitness[i] = 1 / sixhumpcamelback (res\_fit);

}

}

}

float eval\_pop (int number)

{

float res\_eval[sizes], minim = 99999999;

for (int i = 0; i < population\_size; i++)

{

f (population[i], res\_eval);

if (number == 1)

if (dejong (res\_eval) < minim)

minim = dejong (res\_eval);

if (number == 2)

if (schwefel (res\_eval) < minim)

minim = schwefel (res\_eval);

if (number == 3)

if (rastrigin (res\_eval) < minim)

minim = rastrigin (res\_eval);

if (number == 4)

if (sixhumpcamelback (res\_eval) < minim)

minim = sixhumpcamelback (res\_eval);

}

return minim;

}

void cross\_over (int i1, int i2)

{

//incrucisarea cu punct de taiere aleator

int ind = rand () % sizes;

for (int i = ind; i < sizes; i++)

{

std::swap (population[i1][i], population[i2][i]);

}

}

void recombine ()

{

//mutation and/or cross\_over

bool x = 0;

int i1, i2;

for (int i = 0; i < population\_size; i++)

{

if ((1.0 \* rand () / RAND\_MAX) < cross\_probab)

{

if (x == 0)

{

x = 1 - x;

i1 = i;

}

if (x == 1)

{

x = 1 - x;

i2 = i;

cross\_over (i1, i2);

}

}

}

for (int i = 0; i < population\_size; i++)

{

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

{

if ((1.0 \* rand () / RAND\_MAX) < mutation\_probab)

{

population[i][j] = 1 - population[i][j];

}

}

}

}

void GeneticAlgorithm (int number)

{

float best = 99999999;

int rulari = 0;

while (++rulari <= runs)

{

//randomizing(generating) vector numbers

for (int i = 0; i < population\_size; i++)

{

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

{

population[i][j] = rand () % 2;

}

}

//choose(evaluating) best

if (eval\_pop (number) < best)

best = eval\_pop (number);

//stop condition, iteration

int t = 0;

while (++t <= 1000) //generation number

{

//starting selection

float total = 0; //initializing total fitness

eval\_popfitness (number); //evaluating population

selection\_probab[0] = 0; //initializing selection probability

//total fitness

for (int i = 0; i < population\_size; i++)

total = total + population\_fitness[i];

//population fitness probability

for (int i = 0; i < population\_size; i++)

population\_fitness[i] = population\_fitness[i] / total;

//selection probability

for (int i = 0; i < population\_size; i++)

selection\_probab[i + 1] = selection\_probab[i] + population\_fitness[i];

//doing selection

for (int i = 0; i < population\_size; i++)

{

float r = 1.0 \* rand () / RAND\_MAX; //generating r between (0,1]

//going to survive

for (int j = 0; j < population\_size; j++)

{

if (selection\_probab[j] < r && r <= selection\_probab[j + 1]) //condition

{

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

{

new\_pop[i][k] = population[j][k];

}

}

}

}

//actualizing population

for (int i = 0; i < population\_size; i++)

{

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

{

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

}

}

//recombining

recombine ();

//actualizing(evaluating) best

if (eval\_pop (number) < best)

best = eval\_pop (number);

}

}

cout << best;

}

int main ()

{

srand (time (0));

cout << fixed;

char message [MAX\_ARRAY];

int number = 0, ok = 0;

cout << "Type in one of the following name for a function:" << '\n' << "dejong" << '\n' << "schwefel" << '\n' << "rastrigin" << '\n' << "sixhump" << '\n';

//choosing function

while (ok == 0)

{

cin.getline (message, MAX\_ARRAY, '\n');

if (strcmp (message, "dejong") == 0)

{

A = -5.12;

B = 5.12;

number = 1;

ok = 1;

GeneticAlgorithm(number);

}

else

if (strcmp (message, "schwefel") == 0)

{

A = -500;

B = 500;

number = 2;

ok = 1;

GeneticAlgorithm(number);

}

else

if (strcmp (message, "rastrigin") == 0)

{

A = -5.12;

B = 5.12;

number = 3;

ok = 1;

GeneticAlgorithm(number);

}

else

if (strcmp (message, "sixhump") == 0)

{

A = -2;

B = 2;

number = 4;

ok = 1;

GeneticAlgorithm(number);

}

else

{

cout << "You typed in the wrong function name. Try again!" << '\n';

number = 0;

}

}

return 0;

}

**DETALII IMPLEMENTARE:**

Reprezentare: siruri de biti transformate in numere reale, vectori/matrici de numere reale.

Procedura de initializare: Se utilizeaza functia random pentru a genera numerele.

Solutia: reprezentata de un numar real.

**REZULTATE EXPERIMENTALE:**























