Nume si prenume: Paduraru Andra – Elena

Grupa: II A4

Tema: T1

**DESCRIERE PROBLEMA:**

* Acest algoritm functioneaza pentru functiile De Jong, Schwefel, Rastrigin si Six-hump camel back.
* In functie de ce functie se doreste a se calcula, se va utiliza una din cele patru functii din cod.
* In functia principala main() se va alege numele functiei si apoi se vor crea apeluri catre Hill Climbing sau Simulated Annealing. Se utilizeaza o metoda euristica pentru a calcula rezultatele.
* Numerele reprezentate pe biti se vor transforma in numere reale.

**PSEUDOCOD PENTRU FUNCTIA ALEASA:**

dejong (float\* res)

//calcul functie

schwefel (float\* res)

//calcul functie

rastrigin (float\* res)

//calcul functie

sixhumpcamelback (float\* res)

//calcul functie

f(bool\* a, int number)

//se transforma din biti in numere reale intr-un vector fiecare x

//se apeleaza o functie pentru a se calcula

first\_improvement(bool\* s, bool\* s\_prim, int number)

//se alege o solutie convenabila

best\_improvement (bool\* s, bool\* s\_prim, int number)

//se alege cea mai buna solutie convenabila

//hill\_climbing

ITERARE HC begin

t := 0

initialize best

repeat

local := FALSE

BEGIN HC select a candidate solution (bitstring) vc at random

evaluate vc

repeat

vn := Improve(Neghborhood(vc))

if eval(vn) is better than eval(vc)

then vc := vn

else local := TRUE

END HC until local

t := t + 1

if vc is better than best

then best := vc

until t = MAX

end

//simulated\_annealing

begin

t := 0

initialize the temperature T

select a current candidate solution (bitstring) vc at random

evaluate vc

repeat

repeat

select at random vn - a neighbor of vc

if eval(vn) is better than eval(vc)

then vc := vn

else if random[0,1) < exp(-|eval(vn)-eval(vc)|/T)

then vc := vn

until (termination-condition)

T := g(T; t)

t := t + 1

until (halting-criterion)

end

int main ()

{

cout << fixed;

//se citeste numele functiei

//se apeleaza Hill Climbing sau Simulated Annealing in functie de functia aleasa

}

**COD IN C++:**

#include <bits/stdc++.h>

#define MAX\_ARRAY 10

#define dimensions 3

#define length 8

#define runs 10

#define sizes (dimensions \* length)

using namespace std;

float A, B;

int i, j;

float dejong (float\* res)

{

float result = 0;

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

{

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

}

return result;

}

float schwefel (float\* res)

{

float result = 0;

for (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 (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;

}

float f(bool\* a, int number)

{

float res[sizes];

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

{

res[i] = 0;

int p = 1;

for (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 - 1);

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

}

if (number == 1)

return dejong(res);

if (number == 2)

return schwefel(res);

if (number == 3)

return rastrigin(res);

if (number == 4)

return sixhumpcamelback(res);

}

void first\_improvement(bool\* s, bool\* s\_prim, int number)

{

memcpy (s\_prim, s, sizes);

int t = 0;

bool found = 0;

while (!found && ++t <= sizes)

{

int ind = rand() % sizes;

s[ind] = 1-s[ind];

if (f (s, number) > f (s\_prim, number))

{

memcpy(s\_prim, s, sizes);

found = 1;

}

s[ind] = 1-s[ind];

}

}

void best\_improvement (bool\* s, bool\* s\_prim, int number)

{

memcpy(s\_prim, s, sizes);

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

{

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

{

s[i\*length+j] = 1-s[i\*length+j];

if (f (s, number) < f (s\_prim, number))

{

memcpy (s\_prim, s, sizes);

}

s[i\*length+j] = 1-s[i\*length+j];

}

}

}

void HillClimbing (int number)

{

bool vc[sizes], vn[sizes], best[sizes];

int t = 0;

while (++t <= runs)

{

bool local = false;

//randomizing

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

vc[i] = rand() % 2;

cout << "First cout: " << f (vc, number) << ", ";

cout << '\n';

int k = 0;

while (local == false)

{

//first\_improvement (vc, vn, number);

best\_improvement (vc, vn, number);

if (f (vn, number) < f (vc, number))

{

memcpy(vc, vn, sizes);

k++;

cout << "i" << k << '\n';

}

else

local = true;

}

if (f (vc, number) < f (best, number))

memcpy(best, vc, sizes);

}

float res[sizes];

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

{

res[i] = 0;

int p = 1;

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

{

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

p = p\*2;

}

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

res[i] = res[i]/ (p - 1);

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

}

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

cout << "x" << i + 1 << " = " << res[i] << '\n';

}

void neighbour (bool\* a, bool\* b)

{

memcpy(b, a, sizes);

int ind = rand() % sizes;

b[ind] = 1 - b[ind];

}

void SimulatedAnnealing (int number)

{

bool vc[sizes], vn[sizes], best[sizes];

int t = 0;

float T = 1;

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

vc[i] = rand() % 2;

f (vc, number);

while ((T = T \* 0.9) >= 0.01)

{

while (++t <= sizes)

{

neighbour (vc, vn);

if (f (vn, number) < f (vc, number))

memcpy(vn, vc, sizes);

else

if ((1.0 \* rand() / (RAND\_MAX + 1)) < exp((-1.0\*abs(f(vn, number) - f(vn, number))) / T))

memcpy (vc, vn, sizes);

}

f (best, number);

}

float res[sizes];

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

{

res[i] = 0;

unsigned int p = 1;

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

{

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

p = p\*2;

}

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

res[i] = res[i]/ (p - 1);

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

}

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

cout << "x" << i + 1 << " = " << res[i] << '\n';

}

int main ()

{

srand (time (0));

cout << fixed;

char message [MAX\_ARRAY];

int number = 0, ok = 0;

while (ok == 0)

{

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

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

{

A = -5.12;

B = 5.12;

number = 1;

HillClimbing (number);

//SimulatedAnnealing (number);

ok = 1;

}

else

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

{

A = -500;

B = 500;

number = 2;

//HillClimbing (number);

SimulatedAnnealing (number);

ok = 1;

}

else

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

{

A = -5.12;

B = 5.12;

number = 3;

//HillClimbing (number);

SimulatedAnnealing (number);

ok = 1;

}

else

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

{

A = -2;

B = 2;

number = 4;

//HillClimbing (number);

SimulatedAnnealing (number);

ok = 1;

}

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 de numere reale.

Notiunea de vecinatate: Vecinii sunt generati la o distanta hamming 1.

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

Conditia de oprire: Se gaseste un minim local sau se depaseste numarul de rulari cu 1.

Parametrii: functiile au ca parametrii vc sau vn, un numar pentru a ajunge la functia dorita, un rezultat, doua siruri s si s\_prim care vor deveni vc sau vn etc.

**REZULTATE EXPERIMENTALE:**

n = 30 dimensiuni:







































