Raport T1

Leonard Olariu, grupa A5

October 31, 2018

Abstract

Acest raport urmareste prezentarea si testarea unor algoritmi (Best/ First Improvement Hill Climbing, Simulated Annealing) ce doresc cercetarea spatiului de solutii si aproximarea minimului global al unor functii multivariabile, uni/multimodale.

1 Context

Cele 4 functii folosite ca date de test sunt urmatoarele:

```
double DeJong (double *X) {
    double sum = 0;
    for (int i = 0; i < dim; ++i) sum += X[i] * X[i];
    return sum;
} // -5.12 <= X[i] <= 5.12
```

De Jong's function: $f(x) = \sum_{i=1}^{n} x_i^2$, unde $-5.12 \leq x_i \leq 5.12$
Global minimum: $f(x) = 0$; $x_i = 0$, $i = 1 : n$

```
double Schwefel (double *X) {
    double sum = 0;
    for (int i = 0; i < dim; ++i) sum += (-X[i])*sin(sqrt(abs(X[i])));
    return sum;
} // -500 <= X[i] <= 500
```

Schwefel's function: $f(x) = \sum_{i=1}^{n} -x_i * \sin\sqrt{|x_i|}$, unde $-500 \leq x_i \leq 500$
Global minimum: $f(x) = 0$; $x_i = 0$, $i = 1 : n$

```
double Rastrigin (double *X) {
    double sum = 10 * dim;
    for (int i = 0; i < dim; ++i) sum += X[i]*X[i] - 10*cos(2 * M_PI * X[i]);
    return sum;
} // -5.12 <= X[i] <= 5.12
```

Rastrigin's function: $f(x) = 10 * n + \sum_{i=1}^{n} (x_i^2 - 10 * \cos(2 * \pi * x_i))$, unde $-5.12 \leq x_i \leq 5.12$
Global minimum: $f(x) = -n * 418.9829$; $x_i = 420.9687$, $i = 1 : n$

```
double SixHump(double *X) {
    return (4 - 2.1 * X[0] * X[0] + X[0] * X[0] * X[0] * X[0] / 3) * X[0] * X[0] + X[0] * X[1] +
    (-4 + 4 * X[1] * X[1]) * X[1] * X[1];
} // -3 <= X[0] <= 3, -2 <= X[1] <= 2
```

Six-Hump Camel Back function: $f(x) = (4 - 2.1 * x_1^2 + 4 * x_1^4/3) * x_1^2 + x_1 * x_2 + (-4 + 4 * x_2^2) * x_2^2$, unde $-3 \leq x_1 \leq 3$, $-2 \leq x_2 \leq 2$
Global minimum: $f(x_1, x_2) = -1.0316$; $(x_1, x_2) = (-0.0898, 0.7126), (0.0898, -0.7126)$

2 Descriere Algoritmi

2.1 Limbaj Formal (cod) + Limbaj Informal (comentarii)

2.1.1 Best Improvement Hill Climbing

```
void BIHC_Exploitation (bool *X, double (*f)(double *), bool *bestX, double &bestY) {
    for (int i = 0; i < dim*bitLen; ++i)
    {
        X[i] = !X[i];

        double *decX = decode(X);
        double currY = f(decX);
        delete []decX;

        //retin in bestY "best improvement neighbour"
        if (currY < bestY) bestY = currY, memcpy (bestX, X, dim*bitLen);

        X[i] = !X[i];
    }
}

double BIHC (double (*f)(double *)) {
    double best = 1e9, Y, bestY;
    bool X[dim * bitLen], bestX[dim * bitLen];

    for (int i = 1; i <= iterations; ++i) {
        //generez aleatoriu punctul de start
        randomExploration(X);

        //retin valoarea din acest punct
        double *decX = decode(X);
        Y = bestY = f(decX);
        delete []decX;

        bool local = false;
        /*ma deplasez prin spatiul vecinilor la distanta Hamming 1
        cat timp acestia imi ofera o sol mai buna*/
        do {
            BIHC_Exploitation (X, f, bestX, bestY);

            if (bestY < Y) Y = bestY, memcpy (X, bestX, dim * bitLen);
            else local = true;
        } while (!local);

        //verific daca optimul iteratiei curente imbunatateste optimul global
        best = min(best, Y);
    }

    return best;
}
```

2.1.2 First Improvement Hill Climbing

```
bool foundBetter;
void FIHC_Exploitation (bool *X, double (*f)(double *), bool *bestX, double &bestY) {
    for (int i = 0; i < dim*bitLen && !foundBetter; ++i)
    {
        X[i] = !X[i];

        double *decX = decode(X);
        double currY = f(decX);
        delete []decX;

        /*retin in bestY "first improvement neighbour" si
        setez conditia de oprire a iteratiei prin vecini*/
        if (currY < bestY) bestY = currY, memcpy (bestX, X, dim*bitLen), foundBetter = true;

        X[i] = !X[i];
    }
}

double FIHC (double (*f)(double *)) {
    double best = 1e9, Y, bestY;
    bool X[dim * bitLen], bestX[dim * bitLen];

    for (int i = 1; i <= iterations; ++i) {
        //generez aleatoriu punctul de start
        randomExploration(X);

        //retin valoarea din acest punct
        double *decX = decode(X);
        Y = bestY = f(decX);
        delete []decX;

        /*ma deplasez prin spatiul vecinilor la distanta Hamming 1
        cat timp acestia imi ofera o sol mai buna*/
        do {
            foundBetter = false;
            FIHC_Exploitation (X, f, bestX, bestY);

            if (bestY < Y) Y = bestY, memcpy (X, bestX, dim * bitLen);
        } while (foundBetter);

        //verific daca optimul iteratiei curente imbunatateste optimul global
        best = min(best, Y);
    }

    return best;
}
```

2.1.3 Simulated Annealing

```
double T;
void SA_Exploitation (bool *X, double (*f)(double *), bool *bestX, double &bestY) {
    for (int i = 0; i < dim*bitLen && !foundBetter; ++i)
    {
        X[i] = !X[i];

        double *decX = decode(X);
        double currY = f(decX);
        delete []decX;

        /*retin in bestY "first improvement neighbour" si
        setez conditia de oprire a iteratiei prin vecini*/
        if (currY < bestY) bestY = currY, memcpy (bestX, X, dim*bitLen), foundBetter = true;
        /*retin in bestY o solutie candidat mai slaba (cu speranta ca ma
        va conduce spre un optim local neexplorat, poate chiar cel global)
        daca trece testul urmator si setez conditia de oprire a iteratiei prin vecini*/
        else if ((rand() / (RAND_MAX + 1e-6) * 1.0) < exp(-abs(currY - bestY) / T))
            bestY = currY, memcpy (bestX, X, dim*bitLen), foundBetter = true;

        X[i] = !X[i];
    }
}

double SA (double (*f)(double *)) {
    double best = 1e9, Y, bestY;
    bool X[dim * bitLen], bestX[dim * bitLen];

    double T = 5; //initializez temperatura

    for (int i = 1; i <= iterations; ++i) {
        randomExploration(X); //generez aleatoriu punctul de start

        //retin valoarea din acest punct
        double *decX = decode(X);
        Y = bestY = f(decX);
        delete []decX;

        /*ma deplasez prin spatiul vecinilor la distanta Hamming 1
        cat timp acestia imi ofera o sol mai "buna"*/
        do {
            foundBetter = false;
            SA_Exploitation (X, f, bestX, bestY);

            if (bestY < Y) Y = bestY, memcpy (X, bestX, dim * bitLen);
        } while (foundBetter);

        //verific daca optimul iteratiei curente imbunatateste optimul global
        best = min(best, Y);
        /*asigur o scadere treptata a temperaturii dupa fiecare
        iteratie prin inmultirea cu o val subunitara*/
        T *= 0.9;
    }

    return best;
}
```

2.2 Detalii Implementare

2.2.1 Reprezentarea Solutiilor

Siruri binare. Spatiul de cautare se va disctretiza pana la o anumita precizie $10^{-precision}$. Un interval [leftMargin, rightMargin] va fi impartit in $N = (rightMargin - leftMargin) * 10^{precision}$ subintervale egale. Pentru a putea reprezenta cele $(rightMargin - leftMargin) * 10^{precision}$ valori, este nevoie de un numar $bitLen = \lceil log2(N) \rceil$ de biti. Lungimea sirului de biti care reprezinta o solutie candidat va fi suma lungimilor reprezentarilor pentru fiecare parametru al functiei de optimizat.

```
const int dim = 5, iterations = 100;
const double leftMargin = -500, rightMargin = 500;

int precision = 5;
const int bitLen = ceil(log2((rightMargin - leftMargin) * pow(10, precision)));
```

2.2.2 Decodificarea Solutiilor

In momentul evaluarii solutiei (apelul functiei de optimizat) este necesara decodificarea fiecarui parametru reprezentat ca sir de biti in numar real, dupa formula:

$$realValue = leftMargin + decimal(X_{biti})/(2^{bitLen} - 1) * (rightMargin - leftMargin)$$

```
double *decode (bool *X) {
    double *realValue = new double[dim]();

    for (int i = 0; i < dim; ++i) {
        int64_t scaleFactor = 0;

        for (int j = 0; j < bitLen; ++j) {
            scaleFactor *= 2;
            scaleFactor += *(X + i*bitLen + j);
        }

        realValue[i] = leftMargin + (double)scaleFactor / ((1LL << bitLen) - 1) *
                       (rightMargin - leftMargin);
    }

    return realValue;
}
```

2.2.3 Explorarea Random

```
void randomExploration (bool *X) {
    for (int j = 0; j < dim * bitLen; ++j) X[j] = rand() & 1;
}
```

3 Rezultate Experimentale

	Best Improvement Hill Climbing			First Improvement Hill Climbing			Simulated Annealing		
dim	5	10	30	5	10	30	5	10	30
best	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.00000	0.00000
worst	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.00000	0.00000
mean	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.00000	0.00000
stDev	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.00000	0.00000
time (s)	8.195	50.174	1253.136	4.841	27.225	628.422	5.679	31.243	665.590

Table 1: De Jong's function Analysis

	Best Improvement Hill Climbing			First Improvement Hill Climbing			Simulated Annealing		
dim	5	10	30	5	10	30	5	10	30
best	-2094.81	-4120.94	-11760.96	-2094.80	-4037.04	-11002.92	-1975.57	-3706.53	-10755.41
worst	-1976.16	-3802.77	-10760.11	-1907.30	-3634.70	-10274.87	-1907.90	-3630.41	-10253.87
mean	-2065.11	-3945.02	-11095.04	-1988.43	-3764.06	-10531.80	-1947.07	-3691.78	-10443.73
stDev	38.357	73.293	214.456	51.841	103.711	173.366	29.447	11.834	79.009
time (s)	27.168	199.299	4749.821	14.470	95.577	2419.865	15.978	104.236	4041.468

Table 2: Schwefel's function Analysis

	Best Improvement Hill Climbing			First Improvement Hill Climbing			Simulated Annealing		
dim	5	10	30	5	10	30	5	10	30
best	0.00000	2.23078	27.94405	0.99496	4.70242	36.99044	1.00001	7.68730	35.41393
worst	2.47164	9.89821	43.19128	3.70746	11.91305	52.85749	2.47164	9.44140	44.18592
mean	1.41014	5.63531	34.91963	2.18971	8.64220	44.56913	1.41829	8.91805	35.70633
stDev	0.64627	1.54848	3.60779	0.78517	1.59362	4.00974	0.60315	0.24958	1.60153
time (s)	10.868	77.720	2280.036	6.680	45.843	1200.638	8.997	47.650	984.528

Table 3: Rastrigin's function Analysis

	Best Improvement Hill Climbing	First Improvement Hill Climbing	Simulated Annealing
dim	2		
best	-1.031628	-1.031628	-1.031628
worst	-1.031628	-1.031538	-1.031538
mean	-1.031628	-1.031617	-1.031611
stDev	0.000000	0.000025	0.000029
time	0.736	0.556	0.695

Table 4: Six-Hump Camel Back function Analysis

3.1 Observatii

- Aceste rezultate au fost obtinute pentru rularea de 30 de ori a programului cu parametrii fixati astfel: $iterations = 100$, $precision = 5$.
- Daca se doreste o aproximare mai buna a minimului global se poate mari $bitLen$ ul (prin cresterea $precision$ ului) sau numarul de iteratii.

4 Coparatie Metode

Desi in teorie metoda 'Simulated Annealing' este mai predispusa sa descopere noi bazine de atractie, constatam ca si pentru un numar relativ mic de iteratii (100) modul in care am definit notiunea de vecinatate (punct la distanta Hamming 1) asigura o cercetare suficient de buna a spatiului de valori indiferent de metoda pentru care optam.