Algoritmi Genetici - Raport Tema 2

Leonard Olariu - grupa A5

December 8, 2018

Abstract

Acest raport urmareste prezentarea si testarea unui algoritm genetic ce doreste cercetarea spatiului de solutii si aproximarea minimului global al unor functii multivariabile, multimodale.

1 Context

Cele 4 functii propuse optimizarii numerice sunt urmatoarele:

Rastrigin's function $f(x) = 10 * n + \sum_{i=1}^{n} (x_i^2 - 10 * \cos(2 * \pi * x_i))$
Domain $-5.12 \leq x_i \leq 5.12$
Global minimum $f(x) = 0$; $x_i = 0$, $i = 1 : n$

Griewangk's function $f(x) = 1 + \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos \frac{x_i}{\sqrt{i}}$
Domain $-600 \leq x_i \leq 600$
Global minimum $f(x) = 0$; $x_i = 0$, $i = 1 : n$

Rosenbrok's Valley function $f(x) = \sum_{i=1}^{n-1} (100 * (x_{i+1} - x_i^2) + (1 - x_i)^2)$
Domain $-2.048 \leq x_i \leq 2.048$
Global minimum $f(x) = 0$; $x_i = 0$, $i = 1 : n$

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

2 Descriere Algoritm

Metoda este inspirata din teoria evolutionista a lui Darwin. Lucreaza cu o populatie de solutii candidat, care evolueaza si se adapteaza unui mediu (in cazul de fata, mediul este functia de optimizat). Genotipul sufera urmatoarele modificari:

- mutatie (modificarea unei gene din codificarea unui individ)
- incrucisare (recombinarea codului genetic a doi indivizi)

Dupa principiul evolutionist "survival of the fittest", de la o generatie la alta se va favoriza supravietuirea celor mai buni (bine adaptati) indivizi.

2.1 Terminologie

- individ (cromozom) = solutie candidat
- cromozomii sunt compusi din gene (trasaturi)
- fiecare gena se gaseste in cromozom la o pozitie (locus/ loci)
- diferitele valori pe care le poate lua o gena se numesc allele

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

```
void geneticAlgorithm (double (*f)(double *)) { //f - fitness function
    initializePopulation();
    evaluatePopulation(f);

    noChange = 0;
    bestFitness = currPop->candidateFitness[bestCandidate];

    /*termination condition
    reaching a maximum of iMax iterations/ no improvement in the last noChangeMax iterations*/
    for (int i = 1; i <= 1000 && noChange < 100; ++i) {
        selectPopulation(); //select P(i) from P(i-1)

        //alter P(i)
        mutatePopulation();
        crossOverPopulation();

        evaluatePopulation(f);
        if (currPop->candidateFitness[bestCandidate] <= bestFitness) ++noChange;
        else bestFitness = currPop->candidateFitness[bestCandidate], noChange = 0;
    }
}
```

2.3 Detalii Implementare

2.3.1 Functia Fitness

Masoara cat de bine adaptat (fit) la mediu este un individ. Aceasta se bazeaza pe functia de optimizat. Trebuie sa fie pozitiva si cu atat mai mare cu cat individul este mai bun (motiv: vezi procedura de selectie).

Indicii pentru obtinerea unei functii fitness cu proprietatile precizate, pe baza unei functii de test oarecare:

- o problema de minimizare a unei functii f(x) este echivalenta cu problema de maximizare a functiei -f(x) sau a functiei 1/f(x)
- o functie f(x) care are si valori negative poate fi "translata" cu o constanta C, astfel incat f(x)+C sa fie mereu pozitiv

2.3.2 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 = 30;
const double leftMargin = -5.12, rightMargin = 5.12;

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

struct population {
    bool candidateSol[popSize][MAXCANDIDATESIZE]; //MAXCANDIDATESIZE >= dim * bitLen
    double candidateFitness[popSize], popFitness, wheelStart[popSize];
};
population *currPop = new population();
```

```
const int popSize = 100; //"standard" parameters
const double mutationProbability = 0.01, crossOverProbability = 0.25;
```

2.3.3 Decodificarea Solutiilor

In momentul evaluarii unei solutii candidat (apelul functiei fitness) 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;
}
```

2.3.4 Initializarea Populatiei

Se genereaza un numar popSize de solutii candidat. Generarea poate fi aleatoare, sau poate folosi o alta metoda (de exemplu Hill Climbing) pentru asigurarea unei calitati mai ridicate a candidatilor initiali.

```
void initializePopulation() {
    for (int i = 0; i < popSize; ++i)
        for (int j = 0; j < dim*bitLen; ++j) currPop->candidateSol[i][j] = rand() & 1;
}
```

2.3.5 Evaluarea Populatiei + Pregatirea "Rotii Norocului"

Fiecare solutie candidat este decodificata si se va calcula pentru ea functia fitness (care poate fi diferita de functia obiectiv!).

```
void evaluatePopulation(double (*f)(double *)) {
    double *decodeCandidate = decode(currPop->candidateSol[0]);
    currPop->candidateFitness[0] = f(decodeCandidate);
    delete []decodeCandidate;

    currPop->popFitness += currPop->candidateFitness[0];
    bestCandidate = 0;

    //prepare the "fortune wheel"
    for (int i = 1; i < popSize; ++i) {
        //compute the fitness of the individual i
        double *decodeCandidate = decode(currPop->candidateSol[i]);
        currPop->candidateFitness[i] = f(decodeCandidate);
        delete []decodeCandidate;
```

```
        //compute the total fitness of the population
        currPop->popFitness += currPop->candidateFitness[i];

        if (currPop->candidateFitness[i] > currPop->candidateFitness[bestCandidate])
            bestCandidate = i;
    }

    //set the starting point on the "wheel" for every individual
    for (int i = 1; i < popSize; ++i)
        currPop->wheelStart[i] = currPop->wheelStart[i-1] +
        //probability of selecting the individual i
        (currPop->candidateFitness[i-1] / currPop->popFitness);
}
```

2.3.6 Selectia Noii Generatii

Indivizii sunt selectati in noua populatie cu o probabilitate proportionala cu fitness-ul, favorizandu-se supravietuirea celor mai buni indivizi. Astfel, unii indivizi pot avea mai multe copii in noua populatie, iar altii niciuna.

Sugestie Optimizare. Cum indivizii sunt deja ordonati dupa punctul de start pe roata, in urma alegerii unei pozitii random putem identifica individul "norocos" printr-o cautare binara.

```
int binarySearch(int left, int right, double val) {
    if (left > right) return -1;

    int mid = (left + right) / 2;
    if ((val >= currPop->wheelStart[mid] && val < currPop->wheelStart[mid+1]) ||
    mid == popSize-1) return mid;

    if (val < currPop->wheelStart[mid]) return binarySearch (left, mid-1, val);
    else return binarySearch (mid+1, right, val);
}

void selectPopulation() {
    population *newPop = new population();
    for (int i = 0; i < popSize; ++i) {
        double fortune = rand() / (RAND_MAX + 1.);
        int luckyCandidate = binarySearch(0, popSize-1, fortune);
        memcpy(newPop->candidateSol[i], currPop->candidateSol[luckyCandidate],
        dim*bitLen * sizeof(bool));
    }

    delete currPop;
    currPop = newPop;
}
```

2.3.7 Alterarea Indivizilor - Mutatia

Altereaza una sau mai multe gene alese arbitrar dintr-un cromozom. Probabilitatea de mutatie este data de parametrul mutationProbability. Numarul de gene care sufera mutatie este estimat de mutationProbability * (dim*bitLen) * popSize. Daca se foloseste reprezentarea ca sir de biti, mutatia consta in schimbarea valorii bitului respectiv din 0 in 1 sau invers.

```
void mutatePopulation() {
    for (int i = 0; i < popSize; ++i)
        for (int j = 0; j < dim*bitLen; ++j)
            if (rand() / (RAND_MAX + 1.) < mutationProbability)
                currPop->candidateSol[i][j] = !currPop->candidateSol[i][j];
}
```

2.3.8 Alterarea Indivizilor - Incrucisarea

Combina trasaturile a doi cromozomi parinti, rezultand urmasi care mostenesc partial aceste trasaturi. Afecteaza cromozomii cu o probabilitate crossOverProbability. Numarul de indivizi care participa la incrucisari intr-o generatie este estimat de crossOverProbability * popSize.

- Incrucisarea cu un punct de taiere, ales aleator

```
void crossOverPopulation() {
    int pos[2], curr = 1;
    for (int i = 0; i < popSize; ++i)
        if (rand() / (RAND_MAX + 1.) < crossOverProbability) {
            curr = (curr+1) % 2;
            pos[curr] = i;

            if (curr) {
                int cut = rand() % (dim*bitLen - 1) + 1;
                for (int j = cut; j < dim*bitLen; ++j)
                swap(currPop->candidateSol[pos[0]][j], currPop->candidateSol[pos[1]][j]);
            }
        }
}
```

- Incrucisare uniforma: pentru fiecare locus se selecteaza probabilist gena unuia dintre parinti

```
void uniformCrossOverPopulation() {
    int pos[2], curr = 1;
    for (int i = 0; i < popSize; ++i)
        if (rand() / (RAND_MAX + 1.) < crossOverProbability) {
            curr = (curr+1) % 2;
            pos[curr] = i;

            if (curr) {
                for (int j = 0; j < dim*bitLen; ++j)
                    if (rand() / (RAND_MAX + 1.) < 0.5) swap(currPop->candidateSol[pos[0]][j],
            }
        }
}
```

3 Rezultate Experimentale

dim	best	worst	mean	stDev	time(s)
5	0.0005	6.520824	2.592754	2.149368	2.819
10	0.014316	16.20774	6.258031	4.054297	6.747
30	268.9049	375.2729	328.9278	67.1577	7.766

Table 1: Rastrigin's function Analysis

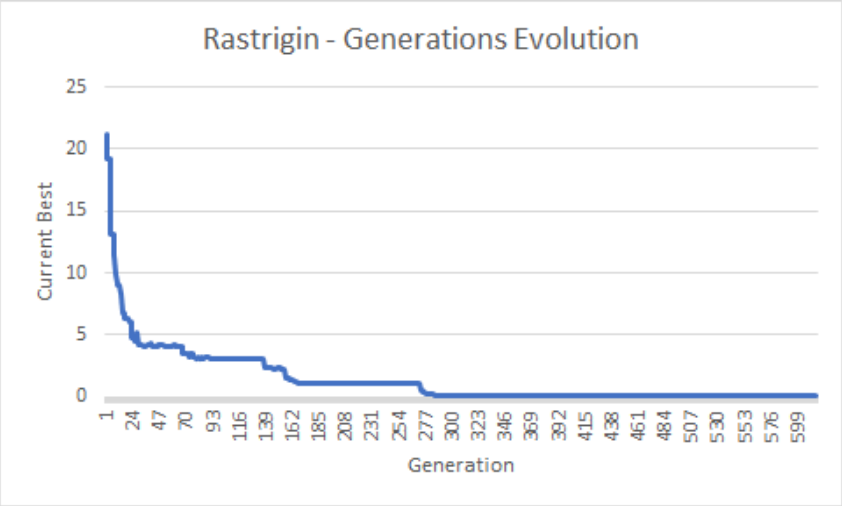

Chart 1: 5 dimensions, 1 run

dim	best	worst	mean	stDev	time(s)
5	0.01494	0.236258	0.07999	0.056983	3.99
10	0.337246	1.221445	1.07868	0.250496	5.924
30	40.26489	185.052	106.4894	41.27796	17.592

Table 2: Griewangk's function Analysis

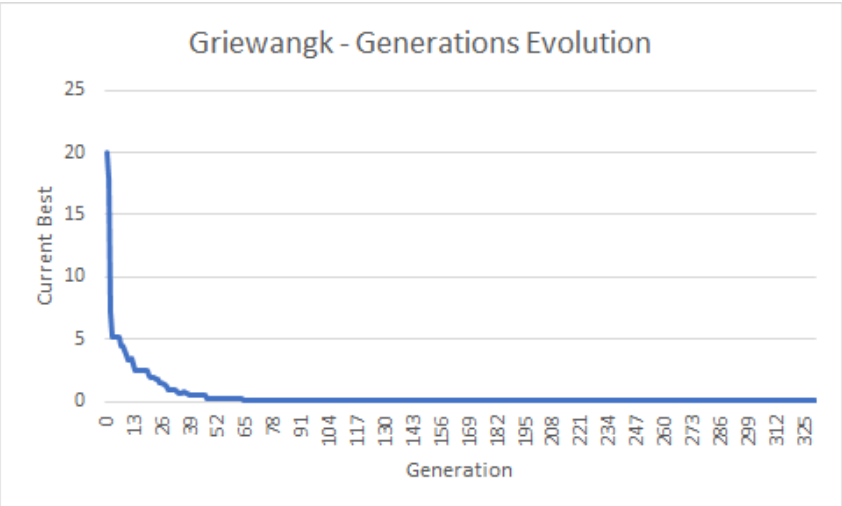

Chart 2: 5 dimensions, 1 run

dim	best	worst	mean	stDev	time(s)
5	1.445085	14.71907	5.114197	2.908234	1.67
10	3.310754	41.71715	12.3806	7.190294	3.772
30	80.86974	891.389	258.8388	161.8607	18.471

Table 3: Rosenbrok's Valley function Analysis

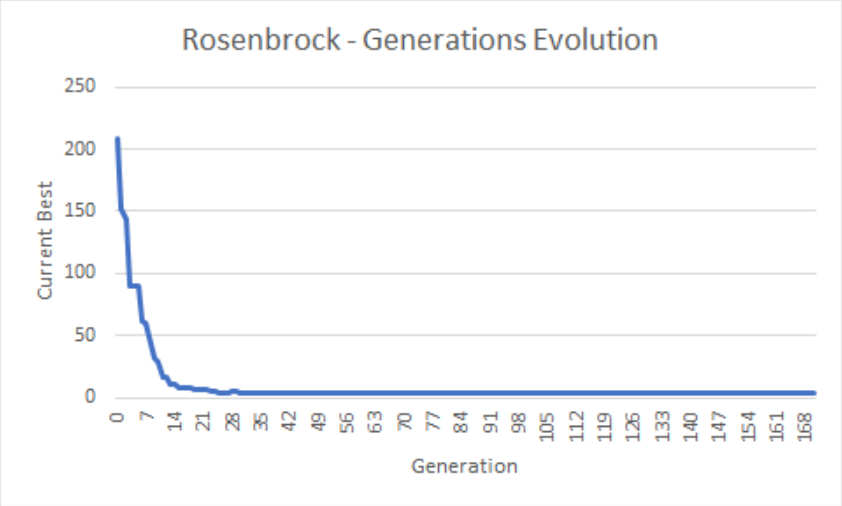

Chart 2: 5 dimensions, 1 run

dim	best	worst	mean	stDev	time(s)
2	-1.03163	-0.99867	-1.02433	0.190541	1.096

Table 4: Six-Hump Camel Back function Analysis

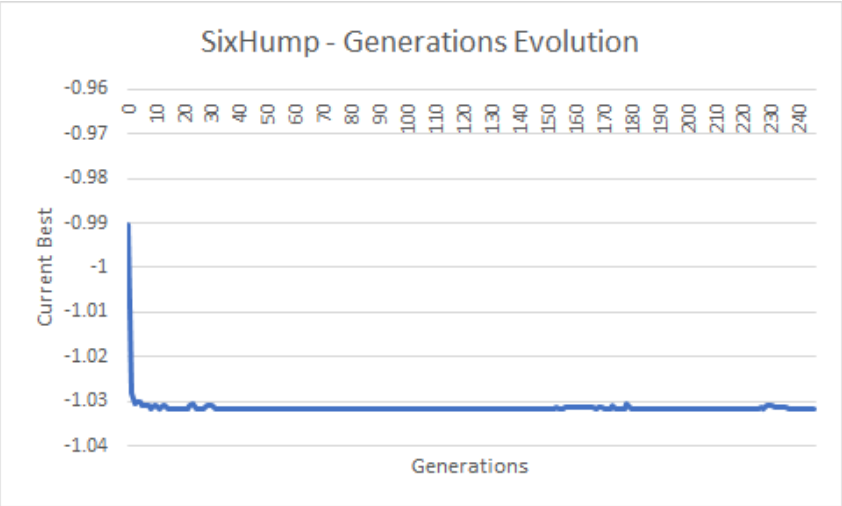

Chart 2: 2 dimensions, 1 run

Observatie. Aceste rezultate au fost obtinute pentru rularea de 30 de ori a programului cu parametrii "standard" sugerati.