Raport tema2 – Algoritmi Genetici

Rameder Carol, IIA2

Folosirea algoritmilor genetici pentru studierea functiilor Rastrigin, Griewangk, Rosenbrock si Six-hump camel back function si compararea rezultatelor obtinute cu Hill Climbing si Simulated Annealing

1.Problema

1.1 Motivatie

Experimentul realizat are rolul de a arata functionalitatea unui Algoritm Genetic cu particularitatile sale prin studierea minimului a patru functii date, Rastrigin, Griewangk, Rosenbrock si Six-hump camel back function si compararea eficientei si a preciziei prin rezultatele obtinute cu alte doua metode euristice Simulated Annealing si Hill Climbing (metodele First Improvement si Best Improvement) .

Un algoritm genetic este o tehnica adaptiva de căutare euristică, care rezolva o problema de optimizare sau căutare, bazate pe principiile geneticii și ale selecției naturale, enunțate de **Darwin** ("supraviețuiește cel care e cel mai bine adaptat"). Mecanismul este similar procesului biologic al evoluției. Acest proces posedă o trăsătură prin care numai speciile care se **adaptează** mai bine la mediu sunt capabile să supraviețuiască și să evolueze peste generații, în timp ce acelea mai puțin adaptate nu reușesc să supraviețuiască și cu timpul dispar, ca urmare a selecției naturale. Probabilitatea ca specia să supraviețuiască și să **evolueze** peste generații devine cu atât mai mare cu cât gradul de adaptare crește, ceea ce în termeni de optimizare înseamnă că soluția se apropie de **optim**.

Condiția esențială pentru succesul a unui algoritm genetic este ca problema de rezolvat să nu ceară obținerea soluției optime, ci să fie suficientă și o soluție apropiată de optim, iar gradul de precizie – **fitness-ul** - unei solutii generata sa poata fi determinat .

Algoritmul genetic se bazeaza pe atat pe operatorii inspirati din genetica de **mutatie**, **crossover** si **selectie,** cat si pe o populatie de solutii candidat, fiind o multime de “indivizi” a caror adaptibilitate este calculata prin functia fitness, aceasta oferindu-le sanse proportionale de a fi selectati in generatia urmatoare .

Mutația este un operator genetic utilizat pentru menținerea diversității genetice de la o generație a unei populații de algoritmi genetici de **cromozomi** la următoarea. Este analog cu mutația biologică. Mutația modifică una sau mai multe valori ale genelor într-un cromozom din starea inițială. În mutație, soluția se poate schimba în totalitate față de soluția anterioară. Această probabilitate stabilita ar trebui să fie scazuta. Dacă este setată prea sus, căutarea se va transforma într-o căutare aleatorie primitivă.

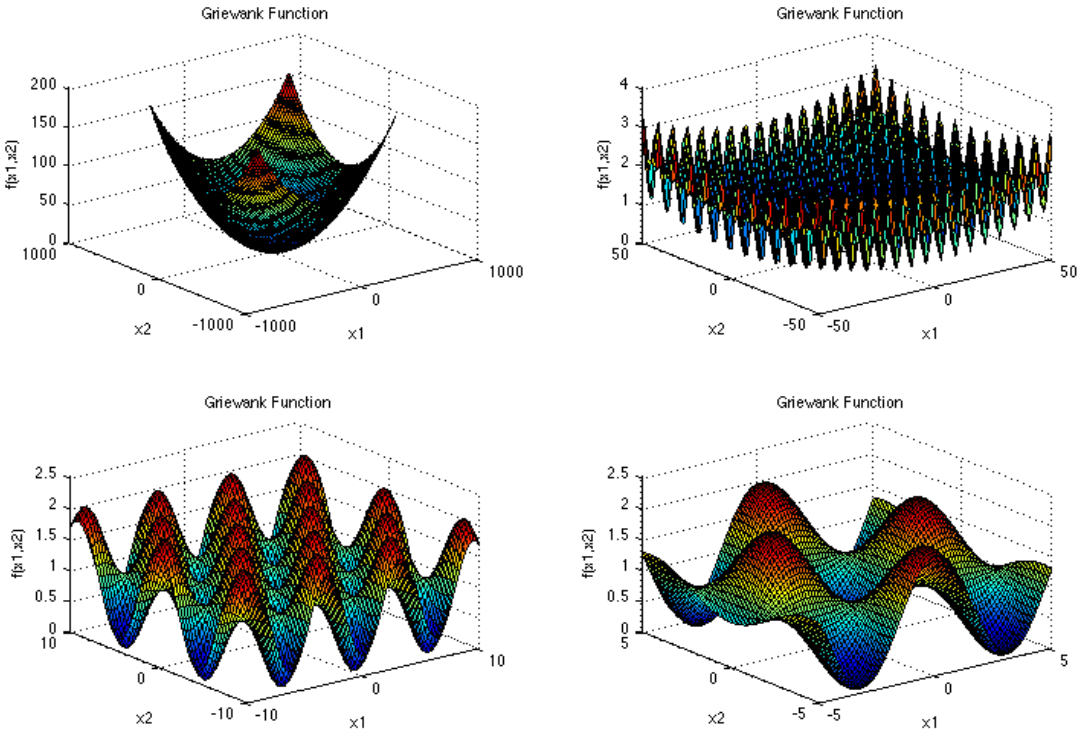
Crossover-ul (sau recombinarea) este un operator genetic utilizat pentru a combina informațiile genetice ale doi părinți pentru a genera noi urmași. Este o modalitate de a genera noi solutii de la o populație existentă și este analog crossoverului care se întâmplă în timpul reproducerii sexuate în biologie. Soluțiile pot fi, de asemenea, generate prin clonarea unei soluții existente, fiind analog reproducerii asexuate.

Selecția este etapa unui algoritm genetic în care cromozomii individuali sunt aleși dintr-o populație pentru reproducerea ulterioară. Dacă această procedură se repetă până când sunt destui indivizi selectați, această metodă de selecție se numește fitness proportionate selection or roulette-wheel selection.

Functiile a caror minim il determinam folosind algoritmul genetic sunt multidimensionale pe multimea numerelor reale si au urmatoarele formule de definiere, domenii de existenta si grafice :

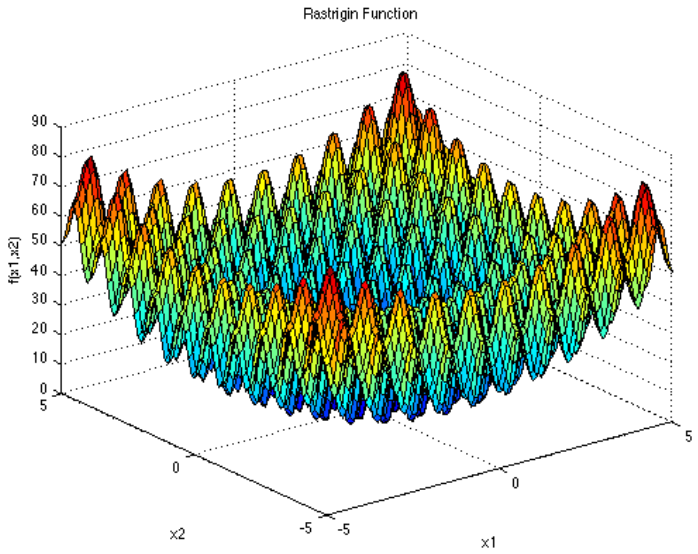
a) **Griewank function**

$$f_8(x) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \qquad -600 \le x_i \le 600$$



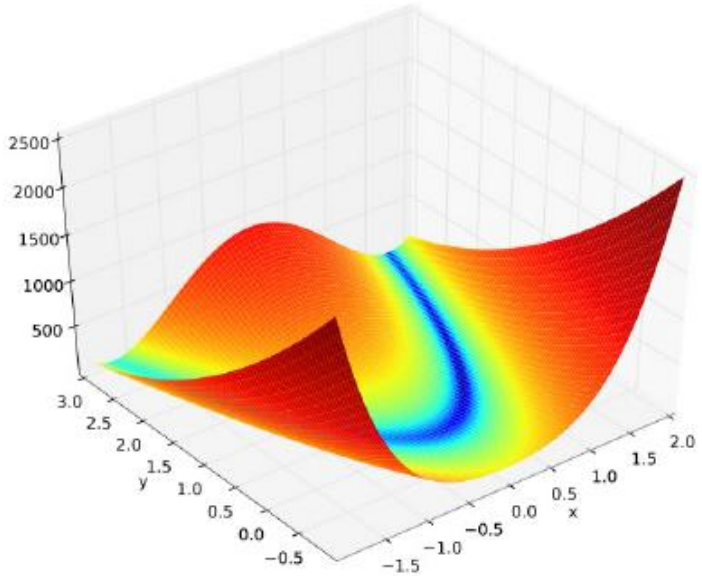
a) **Rastrigin function**

$$f_6(x) = 10 \cdot n + \sum_{i=1}^{n} \left(x_i^2 - 10 \cdot \cos(2 \cdot \pi \cdot x_i) \right) \qquad -5.12 \le x_i \le 5.12$$



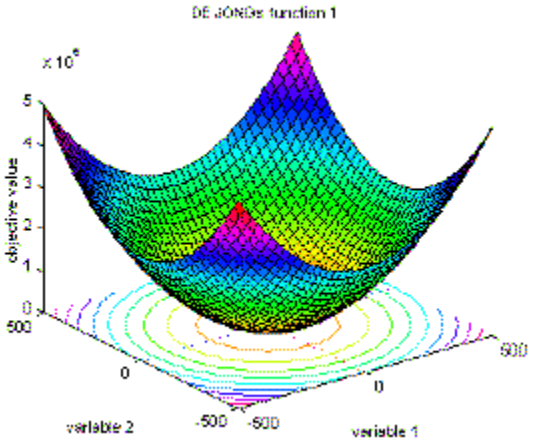
c) Rosenbrock function

$$f_2(x) = \sum_{i=1}^{n-1} 100 \cdot \left(x_{i+1} - x_i^2\right)^2 + \left(1 - x_i\right)^2 \qquad -2.048 \le x_i \le 2.048$$



d) De Jong

$$f_1(x) = \sum_{i=1}^{n} x_i^2 \qquad -5.12 \le x_i \le 5.12$$



2.Algoritmul utilizat

2.1 Pseudocod

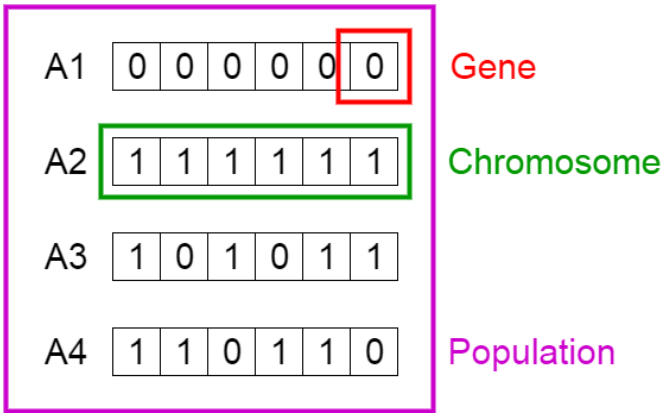
2.1.1 Pseudocodul general al programului

```
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
```

2.1.2 Pseudocodul selectiei

```
EVALUEAZA P             for i:=0 to POP_SIZE
                                eval[i]=f(P(i))
FITNESSUL TOTAL         for i:=0 to POP_SIZE
                                T+=eval[i]
PROB.SEL.INDIVIDUALE    for i:=0 to POP_SIZE
                                p[i]=eval[i]/T
PROB.SEL.CUMULATE       q[0] = 0
                        for i:=0 to POP_SIZE
                                q[i+1]=q[i]+p[i]
SELECTIA                for i:=0 to POP_SIZE
                                genereaza uniform r in (0,1]
                                selecteza pentru supravieture individul j
                                   pentru care q[j]<r<=q[j+1]
```

2.2 Detalii de implementare



Fiecare **solutie candidat** este un vector de tip bool care stocheaza binar cei 5, 10 sau 30 de parametri ai functiei.

Populatia este multimea indivizilor, adica toate solutiile candidat dintr-o generatie, si este o matrice de tip bool, in care punem pe fiecare linie cate un individ, iar in algoritmul implementat vom avea 100 de linii . Populatia se modifica dupa fiecare iteratie, acolo fiind mereu selectati cu sanse mai mari indivizii mai adaptati si este initializata cu valori random de 0 sau 1 cu probabilitati egale.

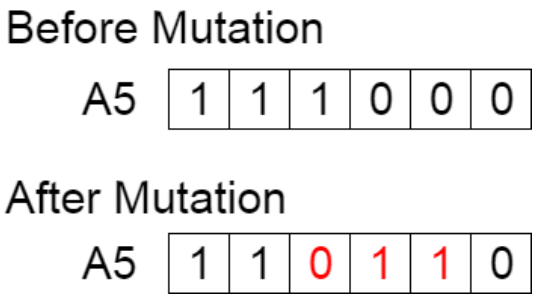
Conditia e oprire a algoritmului este ajungerea la un numar maxim de generatii, considerat in acest caz 1000, deci vor fi operate un numar de 1000 de iteratii .

Importanta in acest algoritm este si **functia de decodificare**, aceasta primeste ca parametru o solutie candidat si intoarce un vector cu valori reale, in ordinea in care se regaseau initial, folosindu-se de urmatorul proces : pentru a afla de cati biti avem nevoie pentru a reprezenta un argument dintr-o solutie candidat impartim intervalul de definitie a functiei pe care o analizam in N = (b-a)*10^d subintervale egale , unde a si b sunt capetele intervalului, iar d este precizia – numarul maxim permis de zecimale. Pentru a putea reprezenta cele (b-a)*10d valori, este nevoie de un număr n = parte_intreaga_superioara(log2(N)) de biţi. Lungimea şirului de biţi care reprezintă o soluţie candidat va fi suma lungimilor reprezentărilor pentru fiecare parametru al funcţiei de optimizat. In momentul evaluarii solutiei (apelul functiei de optimizat) este necesara decodificarea fiecarui parametru reprezentat ca sir de biti in numar real, dupa formula: Xreal = a+decimal(xbiti)*(b-a)/(2n-1) .

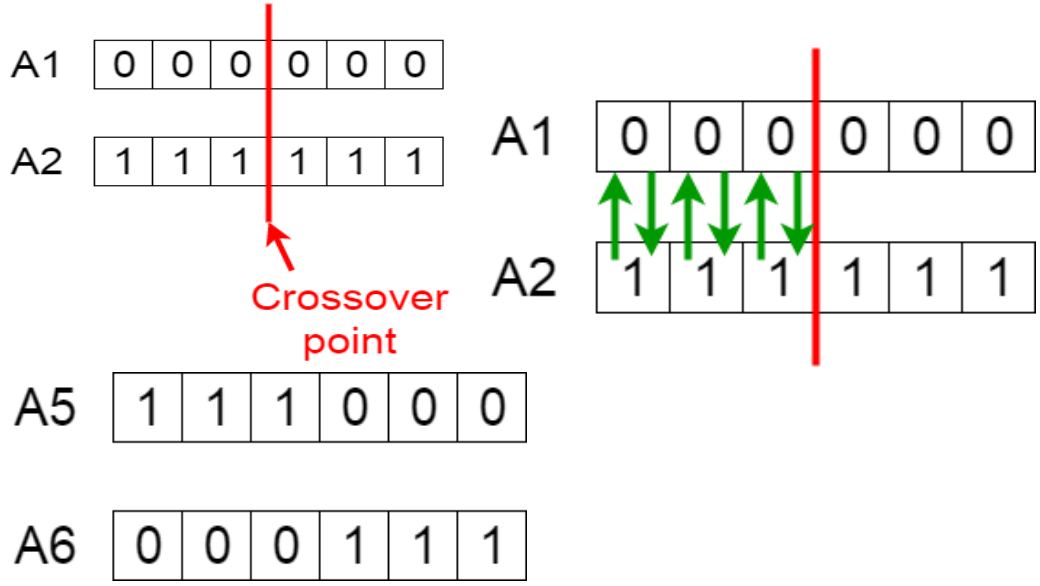
Uitandu-ne inapoi la definitia algoritmului genetic, vom vedea cum au fost implementati cei mai importanti operatori : **mutatia**, **crossover** si **selectia.**

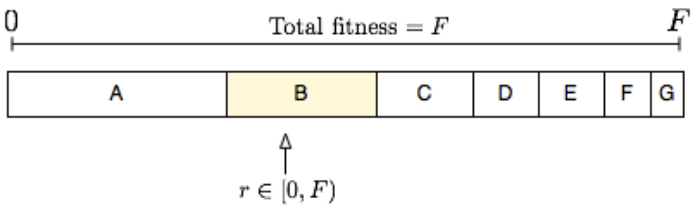
Mutatia se realizeaza cu o probabilitate ce tinde asimptotic spre 0,01, negandu-se bitii din populatie, dupa cum urmeaza :

```
for i:=0 to POP_SIZE
    for j:=0 to Crom_SIZE
        if (rand01()<pcx)
            pop[i][j]=! pop[i][j];
```

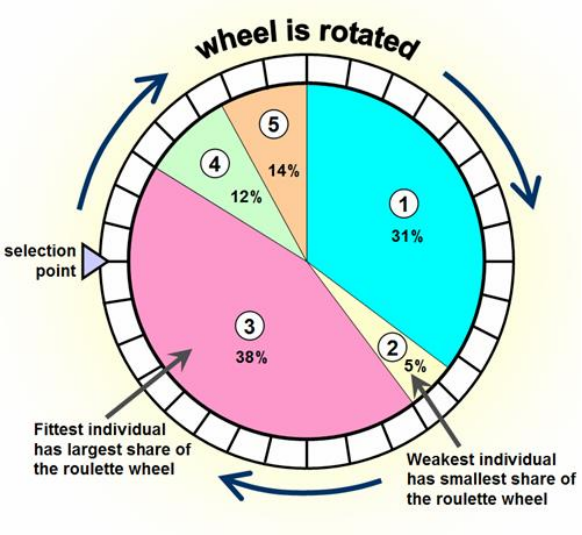


Crossoverul se realizeaza alegand o pozitie aleatoare in ambii cromozomi parinti și desemnata „punct de încrucișare”. Bitii din dreapta acelui punct sunt schimbati între cei doi cromozomi părinți. Rezultă doi urmași, fiecare purtând informații genetice de la ambii părinți.





Selectia se realizeaza pe baza vectorului fit, unde sunt stocate valorile dupa aplicarea presiunii de selectie dupa formula `1.1 * maxf - f[i];` Este construit apoi vectorul pentru Roulette Wheel Selection, dupa formula `fs[i] = fs[i - 1] + fit[i];` astfel, alegand random o pozitie, candidatii au sanse proportionale cu fitness-ul. Fiecare pozitie din noua populatie este aleasa in acest mod, cromozomii mai adaptati avand sansa de a se repeta in urmatoarea generatie, fara sa influenteze evaluarea minimului si a maximului functiei.



3.Rezultate experimentale

Griewangk	5-parametri	10-paramteri	30-paramteri
1	1,58864	6,83953	73,2536
2	0,957591	4,42808	85,9196
3	0,655817	2.92041	75,4219
4	0,767327	2.24804	101,944
5	1,14402	2,51474	77,418
6	0,773316	2,53457	72,5272
7	0,654713	2,68492	85,2486

8	0,758066	3,05672	95,9203
9	0,831876	3,26207	81,2077
10	0,891067	2,58786	68,5032
11	0,920048	4,15849	101,924
12	0,882694	2,74161	66,4805
13	0,778246	3,8268	82,7898
14	0,671739	2,94281	60,5685
15	0,789771	3,20897	63,4832
avg	0,8628332	3,161403	75,46944
max	1,58864	6,83953	101,944
min	0,654713	2,24804	60.5685

Rastrigin	5-parametri	10-paramteri	30-paramteri
1	0,9532	9,23699	169,352
2	1,2386	9,93643	180,495
3	2,19826	7,934	174,02
4	2,08202	10,9445	163,776
5	1,11436	10,7528	168,727
6	0,128636	9,81149	177,185
7	1,22212	9,433	161,967
8	0,500307	11,8403	183,206
9	0,794394	9,48221	172,073
10	1,22131	12,4835	193,704
11	0,176102	11.5573	173,074
12	1,19508	13.8039	153,072
13	1,12247	7.78513	149,749
14	0,0608397	9.69514	167,567
15	0,276002	11.8545	178,534
avg	0,95224	10,43674	170,60773
max	2,19826	13.8039	193,704
min	0.0608397	7.78513	149,749

Rosenbrock	5-parametri	10-paramteri	30-paramteri
1	3,91778	28,0874	555,826
2	4,63613	46,7134	744,071
3	2,74085	31,2402	622,79
4	1,36177	39,8022	568,423
5	1,77564	36,5656	628,432
6	5,10945	30,1661	670,751
7	3,31359	26,2429	555,6
8	1,05097	38,5539	674,959
9	1,00665	28,9076	691,714
10	3,14504	46,6706	530,991
11	0,699611	32,8746	605,118
12	0,477093	14,8741	630,598
13	1,72912	43,6495	608,866
14	0,426067	37,8871	722,171
15	1,45271	34,7656	413,725
avg	2,0742234	34,46672	614,9356
max	5.10945	46.7134	744,071
min	0.426067	14.8741	413.725

De Jong	5-parametri	10-paramteri	30-paramteri
1	0,0211304	0,241254	23,6406
2	0,0416196	0,566954	22,4927
3	0,00630204	0,663027	25,3011
4	0,0337507	0,865723	22,4625
5	0,00784207	0,558464	23,6319
6	0,0307855	0,882974	18,0801
7	0,00408771	0,744625	28,8659
8	0,0270944	0,770967	16,5336
9	0,0406445	0,600104	22,5138
10	0,00291725	1,16126	21,6703
11	0,0148085	0,750365	22,5144
12	0,0266287	0,404682	24,4207
13	0,0163595	0,446529	22,3489
14	0,0106846	0,828134	26,818
15	0,0282758	0,714651	24,0453

avg	0,02363672	0,67998086	23,022653
min	0,00291725	0,241254	16,5336
max	0,0416196	1,16126	28,8659

Rezultate obtinute apeland functia De Jong si varianta first improvement a algoritmului Hill Climbing:

Dimensiune	minim	maxim	avg
5	3,06749*10^-11	6,42784*10^-11	4,5595*10^-11
10	4,07918*10^-11	1,11263*10^-10	7,43347*10^-11
30	1,85058*10^-10	9,14238*10^-10	2,63006*10^-10

Rezultate obtinute apeland functia De Jong si varianta best improvement a algoritmului Hill Climbing:

Dimensiune	minim	maxim	avg
5	9,5947*10^-12	7,45282*10^-12	4,10997*10^-12
10	2,9865*10^-11	1,43160*10^-10	8,25033*10^-10
30	1,48844*10^-10	3,46217*10^-10	2,552*10^-10

Dupa cum putem observa din tabelul cu rezultatele obtinute analizand functia De Jong, Hill Climbing este mai eficient decat un Algoritm genetic, oferind solutii mai apropiate de optim, indiferent de numarul de paramteri.

Eficienta algoritmului genetic scade odata cu cresterea numarului de paramteri, valorile obtinute fiind tot mai mari, deci mai departate de minimul optim .

Bibliografie :

https://www.researchgate.net/figure/Plot-of-the-Rosenbrock-function-for-two-dimensions_fig4_236625120

https://www.sfu.ca/~ssurjano/rosen.html

https://www.sfu.ca/~ssurjano/rastr.html

https://www.sfu.ca/~ssurjano/griewank.html

https://www.sfu.ca/~ssurjano/camel6.html

https://profs.info.uaic.ro/~pmihaela/GA/laborator3.html

https://en.wikipedia.org/wiki/Genetic_algorithm

https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3

https://ro.wikipedia.org/wiki/Algoritm_genetic

https://www.codewars.com/kata/567b21565ffbdb30e3000010