UNIVERSITATEA POLITEHNICA DIN BUCUREŞTI

FACULTATEA DE ŞTIINŢE APLICATE

Matematică şi Informatică Aplicată în Inginerie

PROIECT DE DIPLOMĂ

CONDUCĂTOR ŞTIINŢIFIC,
Lect.univ.dr. Iuliana MUNTEANU

ABSOLVENT,
Alexandru MAREŞ

Bucureşti

2020

UNIVERSITATEA POLITEHNICA DIN BUCUREŞTI
FACULTATEA DE ŞTIINŢE APLICATE
Matematică şi Informatică Aplicată în Inginerie

Aprobat Decan,
Prof.dr. Emil PETRESCU

PROIECT DE DIPLOMĂ

Algoritmi Genetici.
Aplicaţii

CONDUCĂTOR ŞTIINŢIFIC,
Lect.univ.dr. Iuliana MUNTEANU

ABSOLVENT,
Alexandru MAREŞ

Bucureşti

2020

Cuprins

Introducere

În această lucrare prezentăm aplicaţii de optimizare în care algoritimii genetici sunt utilizaţi.

Algoritmii genetici sunt o metodă de căutare şi optimizare inspirată de teoria evoluţionismului a lui Charles Darwin. Aceşti algoritmi imită procesul de selecţie naturală, unde cei mai buni indivizi sunt selectaţi pentru a produce o noua generaţie.

În această lucrare sunt prezentate structura generală a unui algoritm genetic, modul de reprezentare al soluţiilor sub formă de cromozomi, iniţializarea populaţiei de soluţii candidat, generarea soluţiilor, funcţiile principale ale acestuia (selecţie, încrucişare, mutaţie) şi variaţii ale funcţiilor de selecţie şi încrucişare, acestea fiind folosite în funcţie de necesitatea problemei ([2], [3]).

Următorul lucru ce va fi prezentat este aplicarea algoritmilor genetici în găsirea maximului unei funcţii. Pentru aceasta vor fi luate trei cazuri:

- Funcţii cu o variabilă pe un interval dat, pentru acestea fiind uşoară găsirea soluţiei fără algoritmi genetici. Rezolvarea acestei probleme va prezenta baza pentru problemele de maxim ce vor urma.

- Funcţii cu două variabile cu intervale de definire date. Această problemă va prezenta lucrul cu soluţii formate din mai multe numere binare.

- Funcţii cu un număr oarecare de variabile pe un domeniu mărginit. Acest tip de problemă fiind greu de rezolvat, vor fi evidenţiate avantajele algoritmilor genetici faţă de un algoritm de căutare brut, ce verifică toate punctele din domeniu. Această problemă evidenţiază avantajele aduse de posibilitatea de a schimba aspecte ale programului cum ar fi dimensiunea populaţiei şi numărul de generaţii pentru a balansa precizia rezultatului şi timpul de rulare.

În al treilea capitol vor fi prezentate rezolvări pentru următoarele probleme:

- Problema rucsacului, care este o problemă combinatorică clasică. Aceasta oferă o listă de obiecte cu greutăţile lor şi profitul adus din vânzarea acestora şi cere găsirea celei mai bune combinaţii de obiecte pentru a maximiza profitul într-o greutate dată. Rezolvarea acestei probleme este relativ uşoară, vectorul soluţie fiind vectorul de apartenenţă al obiectelor.

- Rezolvarea ecuaţiilor cu coeficienţi întregi. Programul generează soluţii formate din numere întregi sub formă binară. Avantajul dat de algoritmii genetici în acest caz este dat de natura aleatori a acestora, programul oferind în general soluţii diferite după fiecare rulare.

- Problema comis-voiajorului este o problemă combinatorică. Aceasta pimeşte o listă cu drumuri între oraşe şi cere cel mai scurt drum ce străbate toate oraşele şi ajunge înapoi în oraşul de start. Rezolvarea acestei probleme se va abate de la forma normală a unui

algoritm genetic, renunţând la funcţia de încrucişare şi bazându-se pe mutaţie pentru a genera populaţii noi.

Pentru această lucrare va fi folosit pachetul MatLab([6]). Acesta oferă:

- un set de funcţii ce vor fi de ajutor pentru rezolvarea problemelor, cum ar fi: rand, randi, sort, max;
- instrumente de plotare, ce vor ajuta la vizualizarea soluţiei
- posibilitatea de a întrerupe în timpul rulării programului, permiţând observarea variabilelor în mijlocul rulării, ajutând la depanarea programelor

Editorul MatLab este împărţit în mai multe zone acestea fiind:

- bara de instrumente ce permite utilizatorului să:
  - salveze programe şi seturi de date,
  - deschidă\creeze\salveze fişiere,
  - manipuleze grafice,
  - împartă programul în mai multe secţiuni,
  - ruleze programul sau secţiuni din acesta.
- fereastra de comandă, unde utilizatorul poate introduce comenzi de o linie ce vor fi rulate la apăsarea tastei enter.
- fereastra editorului, ce permite utilizatorului să scrie programe, acestea fiind salvate în fişiere, programele pot fi:
  - de sine stătătoare, neavând nevoie să primescă variabile externe, pot afişa valori sau mesaje în ferestra de comandă
  - funcţii ce în general primesc şi întorc variabile şi\sau afişează un grafic. Pot fi create funcţii ce nu primesc variabile, nu întorc variabile şi nu afişează niciun grafic, acestea având aceeaşi funcţionalitate ca un program de sine stătător.
- fereastra workspace, aceasta afişează variabilele salvate în memorie
- fereastra folder curent, ce arată fişierele din folderul curent şi permite reselectarea folderului de lucru.

Pentru a porni un program este nevoie ca acesta să fie în folderul de lucru curent şi, în plus, este necesară apăsarea butonului 'Run' din bara de instrumente din tabul 'Editor'. Programele din această lucrare vor afişa în fereastra de comandă rezultatele obţinute.

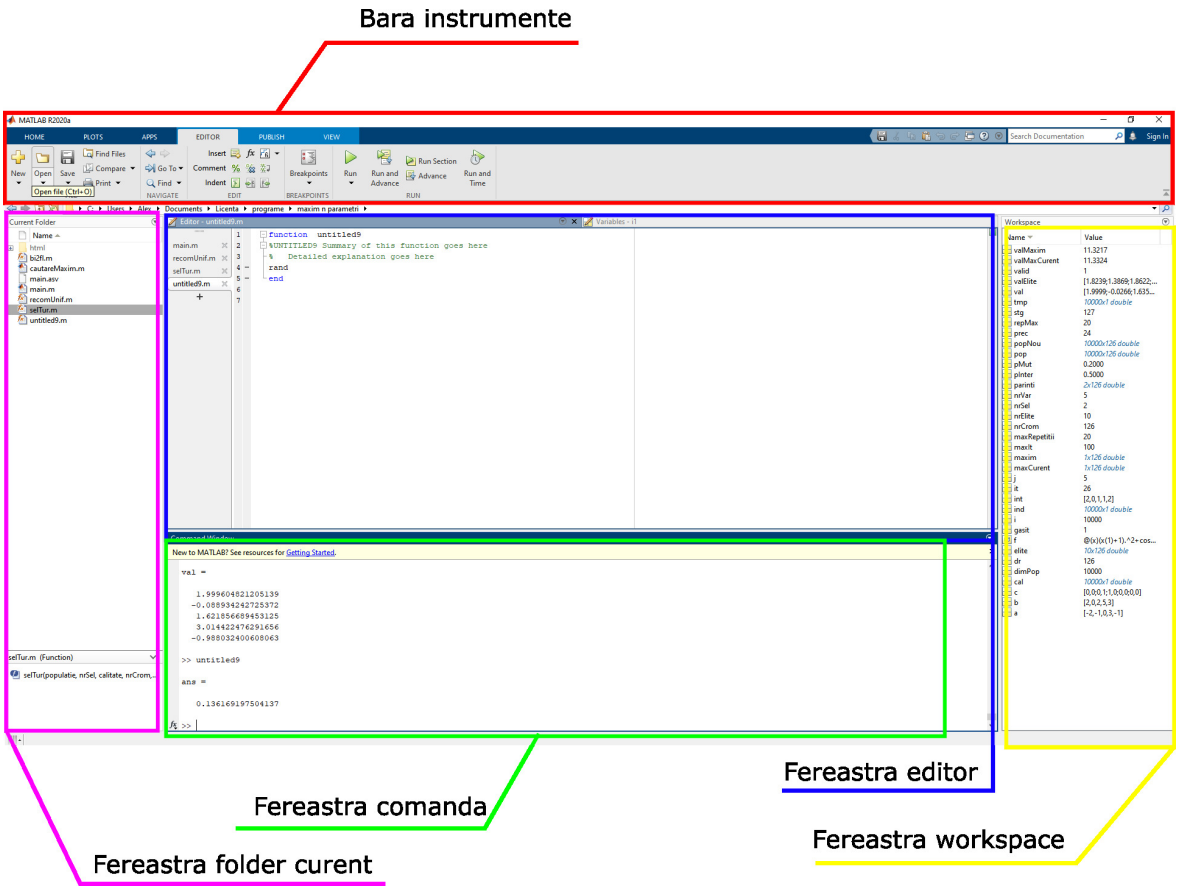


Figura 1: Zone de lucru
MatLab

Capitolul 1

Algoritmi Genetici

1.1 Prezentare generală

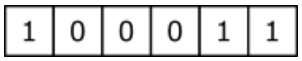
Algoritmii genetici imită procesul de selecţie naturală unde cei mai buni indivizi sunt selectaţi pentru a produce o nouă generaţie. În cazul acesta o nouă mulţime de soluţii candidat. Populaţia iniţială constă dintr-o mulţime de soluţii generate aleator. După generare acesteia este calculată calitatea populaţiei. Pe baza caliţătii sunt selecţati părinţii noii generaţii şi sunt recombinaţi pentru a obţine cromozomii soluţiilor candidat din următoarea generaţie. Înainte de a introduce cromozomii următoarei generaţii este aplicat procesul de mutaţie asupra acestora, proces ce are şanse, de obicei mici, de a modifica cromozomii din soluţiile candidat. In final soluţiile candidat generate sunt considerate generaţia curentă. După aceea se reia de la procesul de selecţie dacă nu a fost găsită soluţie şi nu s-a ajuns la numărul maxim de iteraţii.

- dimpop ← mărime populaţie
- p ← populaţie iniţială
- optim← null
- repetă
- CalculCalitate(p)
- pentru fiecare $p_i \in$ p
- dacă calitate(p_i)>optim
- optim ← p_i
- Temp← $\emptyset$
- pentru i de la 0 până la $dimpop/2$ execută
- p_a = selecţie(p) Părinte a
- p_b = selecţie(p) Părinte b
- $c_a, c_b \leftarrow$ încrucişare(p_a, p_b)
- temp ← temp $\bigcup$ mutaţie(c_a),mutaţie(c_b)
- p ← temp
- până când optim este soluţia ideală sau timpul de rulare a expirat

Figura 1.1: Pseudocod algoritm principal

1.2 Populaţie

Populaţia reprezintă toate soluţiile candidat numite şi indivzi. Fiecare soluţie candidat este formată din cromozomi, acesţia reprezentănd valori sau fiind numere reale. Pentru probleme de optimizare de variabile aceşti cromozomi sunt binari, fiind reprezentarea numerelor în baza doi.



1.2.1 Iniţializarea populaţiei

Aceasta se face de obicei aleator cu ajutorul unei distribuţii. Dar în cazuri în care se cunosc cazuri favorabile, cum ar fi puncte aproape de maxim, acestea se pot introduce ca indivizi în populaţia iniţială.

- $v \leftarrow$ vector nou de lungime l — l-număr cromozomi
- pentru i de la 1 la l execută
- dacă 0.5 > decât un număr generat aleator între 0 şi 1 cu o distribuţie uniformă
- $v_i \leftarrow 1$
- altfel
- $v_i \leftarrow 0$
- întoarce v

1.3 Mutaţie

Deoarece populaţia iniţială nu poate să acopere tot domeniul şi recombinarea nu poate ieşi în afara spaţiului definit de populaţia iniţială, mutaţia este introdusă. Aceasta modifică cu o probabilitate fixă biţii unui individ, producând şi indivizi ce nu aparţin spaţiului generat de populaţia anterioară.

- $v \leftarrow$ vector individ de modificat
- pentru i de la 1 la l execută
- dacă $r >$ decât un număr generat aleator intre 0 şi 1 cu o distribuţie uniformă
  r-rata mutaţie
- $v_i \leftarrow v_i + 1$ — adunare în binar
- întoarce v

Pentru cazul în care cromozomii unui individ nu sunt binari, este determinată în funcţie de problemă metoda de mutaţie.

1.4 Funcţie de încrucişare

Funcţia de încrucişare are rolul de a crea noi indivizi din părinţii selectaţi prin combinarea genelor acestora.

1.4.1 Încrucişare cu un singur punct

Se alege un număr întreg m între 1 si l (lungimea vectorilor). Acesta poate fi ales aleator de fiecare dată sau predeterminat. Se schimbă între ele toate elementele vectorilor părinţi cu

indexul mai mic decât m.

- $v1 \leftarrow$ copie a primului vector părinte
- $v2 \leftarrow$ copie a celui de-al doilea vector părinte
- $m \leftarrow$ un numar întreg aleator intre 1 si l
- pentru i de la 1 la $m-1$ execută
- interschimbă valorile din $v1_i$ şi $v2_i$
- întoarce $v1$
- întoarce $v2$

Dacă m este 1, combinarea nu are loc. Aceasta are probabilitatea de a se întâmpla de $\frac{1}{l}$.

1.4.2 Încrucişare cu două puncte

În cazul încrucişării cu un singur punct; primul si ultimul element au şansele să fie separate foarte mari ($\forall m > 1 \Rightarrow sansa = \frac{l-1}{l}$), iar primul şi al doilea element au şanse foarte mici de a fi separate ($m = 2 \Rightarrow sansa = \frac{1}{l}$). Asta se întâmplă deoarece încrucişarea ţine cont de ordinea în care genele sunt stocate ([1]). Pentru a reduce această diferenţă de şanse, se aleg două numere m şi n, interschimbarea elementelor fiind numai între aceste poziţii.

- $v1 \leftarrow$ copie a primului vector părinte
- $v2 \leftarrow$ copie a celui de-al doilea vector părinte
- $m \leftarrow$ un număr întreg aleator între 1 si l
- $n \leftarrow$ un număr întreg aleator între 1 si l
- dacă $n > m$
- interschimbă m şi n
- dacă $n \neq m$
- pentru i de la n la m-1 execută
- interschimbă valorile din $v1_i$ şi $v2_i$
- întoarce $v1$
- întoarce $v2$

1.4.3 Încrucişare uniformă

Pentru a evita mai bine diferenţa de şanse ca două valori dintr-un cromozom să fie separate, pe fiecare poziţie a vectorilor părinte încrucişarea uniformă are şansa de a interschimba între vectori valorile din aceştia([1]). Şansa de interschimbare este determinată de la început.

- $v1 \leftarrow$ copie a primului vector părinte
- $v2 \leftarrow$ copie a celui de-al doilea vector părinte
- $p \leftarrow$ probabilitate de interschimbare
- pentru i de la 1 la l execută
- dacă $p \geq$ o valoare generată uniform între 0 si 1 atunci
- interschimbă valorile din $v1_i$ şi $v2_i$
- întoarce $v1$
- întoarce $v2$

1.4.4 Încrucişare uniformă pentru N vectori

Pentru a genera o plajă mai mare de soluţii posibile pot fi folosiţi mai mulţi vectori părinte. Acest algoritm amestecă genele de pe aceeaşi poziţie din toţi vectorii iniţiali şi le distribuie vectorilor din noua generatie.

- $V \leftarrow \{V_1, V_2, ..V_N\}$ vectori iniţiali
- $a \leftarrow$ vector de dim l
- $b \leftarrow$ vector de dim N
- $p \leftarrow$ probabilitate de interschimbare
- pentru i de la 1 la l execută
- dacă $p \geq$ o valoare generată uniform între 0 şi 1 atunci
- pentru j de la 1 la N execută
- $a \leftarrow V_j$
- $b_j \leftarrow a_i$
- pentru j de la l la 2 execută
- $k \leftarrow$ un număr întreg ales aleator de la j la 1
- interschimbă b_k şi b_j
- pentru j de la 1 la N execută
- $a \leftarrow V_j$
- $a_i \leftarrow b_j$
- $V_j \leftarrow a$
- întoarce V

1.5 Selecţie

Selecţia are rolul de a determina părinţii generaţiei următoare. Aceasta trebuie să menţină diversitatea populaţiei. Trebuie evitată convergerea la optime locale. In general pentru a selecta se foloseşte o funcţie ce determină calitatea indivizilor din generaţie. După valoarea calităţii se determină părinţii noii generaţii.

1.5.1 Selecţie cu rang

Fiecărui individ i se calculează calitatea, după aceea sunt ordonaţi după valoarea calităţii. Fiecărui individ i se atribuie o "greutate" în funcţie de ordinea acestora. Primul candidat are greutatea 1, iar ceilalţi candidaţi au greutatea egală cu greutatea candidatului anterior sumată cu numarul de ordine al acestuia. Greutatea efectivă al celui de-al n-lea candidat este $greutate_n = \frac{n(n+1)}{2}$ .Candidatul cu cea mai mare valoare a funcţiei de calitate are cea mai mare greutate. Selectarea se face după algoritmul:

- $V \leftarrow \{v_1, v_2, ..v_n\}$ populaţie de n indivizi
- $C \leftarrow \{C_1, C_2, ..C_n\}$ calitate indivizi, C_i aparţine V_i
- ordonez C crescător si ordonez V aşa încât C_i şi v_i îşi păstrează corespondenţa v_i va avea greutatea i
- s =0
- pentru i de la 1 la n
- $s = s + i$
- $k \leftarrow$ un număr întreg de la 1 la s
- ind =1
- val = 1

- dacă k=1
- întoarce v_1
- pentru i de la 2 la n
- $val = val + i$
- $ind = ind + 1$
- dacă $val = k$
- întoarce v_{ind}

1.5.2 Selecţie de tip ruletă

Acest tip de selecţie admite numai valori pozitive pentru elementele vectorului de calitate.

- $V \leftarrow \{v_1, v_2, ..v_n\}$ populaţie de n indivizi
- $C \leftarrow \{C_1, C_2, ..C_n\}$ calitate indivizi, C_i aparţine v_i
- s =0
- pentru i de la 1 la n
- $s = s + C_i$
- $k \leftarrow$ un număr de la 0 la s
- $val \leftarrow C_1$
- dacă $k \leq C_1$
- întoarce V_1
- pentru i de la 2 la n
- $val \leftarrow val + C_i$
- dacă $val \geq k$
- întoarce v_i

1.5.3 Selecţie de tip turneu

Se aleg aleator m candidaţi din toată populaţia, selectându-se cei mai buni k indivizi din cei m.

- $V \leftarrow \{v_1, v_2, ..v_n\}$ populaţie de n
- $C \leftarrow \{C_1, C_2, ..C_n\}$ calitate indivizi, C_i aparţine V_i
- $k \leftarrow$ număr de indivizi selectaţi
- $m \leftarrow$ număr de indivizi candidaţi selectaţi aleator
- $cand \leftarrow \{cand_1, cand_2, ...cand_m\}$ va conţine indicii potenţialilor candidaţi aleşi
- $găsit \leftarrow 1$
- cât timp $găsit = 1$
- $găsit \leftarrow 0$
- pentru i de la 1 la m
- dacă $C_{cand_i} < C_{cand_{i+1}}$
- interschimbă $cand_i$ şi $cand_{i+1}$
- $găsit \leftarrow 1$
- întoarce $\{cand_1, cand_2, ...cand_k\}$

1.5.4 Elitism

Pentru a asigura că cei mai buni indivizi sunt păstraţi în populaţia următoare aceştia sunt introduşi direct în populaţia următoare, fără a fi modificaţi. Se mai pot adăuga şi alte părţi ale populaţiei pentru a păstra diversitatea acesteia.[5]

Capitolul 2

Aplicaţii ale algoritmilor genetici pentru probleme de maxim

Pentru început tratăm cea mai uşoară problemă de maxim, cea pentru o funcţie de o singură variabilă.

2.1 Găsirea maximului pentru o funcţie cu o singură variabilă

Vom lua ca exemplu funcţia $f(x) = \cos(x^2) + e^{\sin(x)} - \sin(x^2)$ pe intervalul $[-4, 4]$. Aceasta are multe puncte de maxim local, putând să creeze probleme algoritmului genetic, ce are şanse să rămână blocat pe un punct de maxim local.

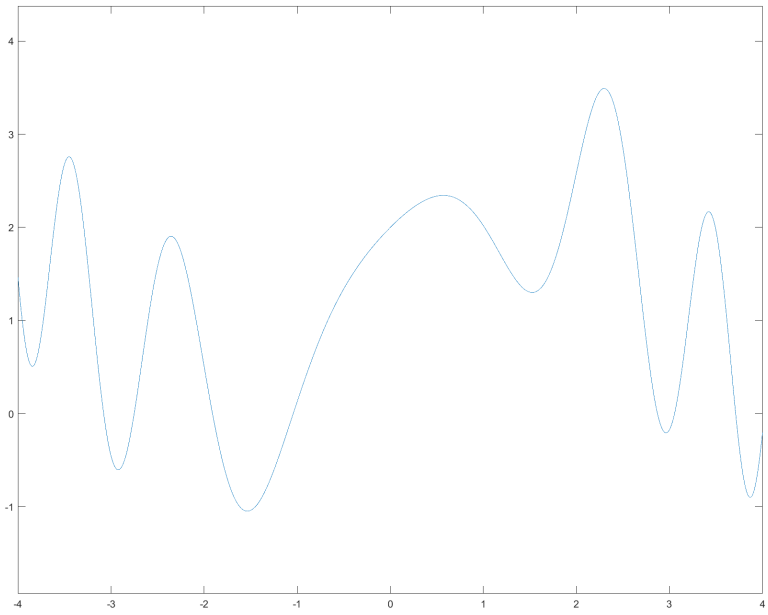


Figura 2.1: $f(x) = \cos(x^2) + e^{\sin(x)} - \sin(x^2)$

Pentru aceast tip de problemă, primii cromozomi ai unui individ vor reprezenta partea întreagă a acestuia, şi cei rămaşi vor reprezenta partea fracţionară. Calculul valorii reale este făcut de funcţia de conversie numită **bi2fl** ce va fi folosită pe parcursul lucrării.

```
function float=bi2fl(binar,virgula)
n=length(binar);
int=0;
putereDoi=1;
for i=1:virgula-1
      int=int+2^(i-1)*binar(i);
      putereDoi=putereDoi*2;
end
fra=0;
for i=1:length(binar(virgula:n))
           fra=fra+2^(-i)*binar(virgula+i-1);
end
float=int+fra;
end
```

Figura 2.2: Transformare din număr binar în număr zecimal

2.1.1 Program

2.1.1.1 Date iniţiale

Numărul de cromozomi necesari pentru partea întreagă este cel mai mic multiplu de doi mai mare decât dimensiunea intervalului în care este cautată soluţia. Numărul de cromozomi ce determină partea fracţionară este ales de utilizator prin modificarea variabilei **prec**, variabilă care determină cât de bine este acoperit intervalul $[0, 1)$ al părţii fracţionare. Algoritmul are de la început un număr maxim de iteraţii la care să se oprească, o probabilitate a evenimentului de mutaţie şi un număr maxim de repetiţii ale aceleiaşi soluţii.

```
n=10000;
maxIt=100;
f=@(x) cos(x.^2)+exp(sin(x))-sin(x.^2);
a=-4;
b=4;
prec=16;
int=ceil(log2(abs(b-a)));
nrCrom=int+prec;
pMut=0.1;
maximRepetitii=5;
```

Figura 2.3: Date iniţiale

2.1.1.2 Iniţializare populaţie

Populaţia este stocată într-o matrice având pe fiecare line cromozomii unui individ. Înainte de a fi pornit algoritmul principal, trebuie generată o populaţie iniţială. Fiecare individ reţine o valoare de la 0 la $b - a$, adăugându-se la aceasta valoarea lui a după calculul valorii in baza 10 cu ajutorul funcţiei bi2fl. În timpul iniţializării condiţia **if** verifică dacă inidividul generat se află în intervalul de definiţie al funcţiei pentru care căutăm maximul.

```
pop=zeros(n,nrCrom);
for i=1:n
    valid=0;
    while valid==0
        pop(i,:)=randi([0 1],1,nrCrom);
        if (a<=(bi2fl(pop(i,:),int+1)+a))&&((bi2fl(pop(i,:),int+1)+a)<=b)
            valid=1;
        end
    end
end
```

Figura 2.4: Iniţializarea populaţiei

Algoritmul ia la început ca valoare de calitate maximă $-\infty$ şi individul ce are calitate maximă va fi iniţializat cu zerouri, conţinutul acestuia la început nefiind important. După iniţializarea datelor programul calculează valoara zecimală a fiecărui individ şi le stochează in vecorul **val**. În cazul problemelor de maxim, funcţia de calitate este chiar funcţia pe care se lucrează. Pentru problema de minim, funcţia de calitate este opusul funcţiei pe care se lucrează.

Înainte de a putea începe bucla principală **while**, este calculată calitatea fiecărui individ.

2.1.1.3 Recombinare cu un punct

Algoritmul de recombinare cu un singur punct, ce este folosit în acest program, primeşte două argumente care sunt părinţii (**par1** şi (**par2**) şi un argument ce transmite numărul de cromozomi (**nrCrom**).

```
function [out1 , out2] = recom1p(par1,par2,nrCrom)
pct=randi(nrCrom);
out1=zeros(1,nrCrom);
out2=zeros(1,nrCrom);
for i=1:pct
    out1(i)=par1(i);
    out2(i)=par2(i);
end
for i=pct+1:nrCrom
    out1(i)=par2(i);
    out2(i)=par1(i);
end
end
```

Figura 2.5: Recombinare cu 1 punct **recom1p**

2.1.1.4 Selecţie de tip turneu

Algoritmul primeşte ca date de intrare populaţia din care selectează părinţii (**populatie**), caliatea indivizilor din populaţie (**calitate**), numărul de indivizi ce vor fi selectaţi (**nrSel**) şi numărul de cromozomi ai unui individ (**nrCrom**). Funcţia întoarce indivizii selectaţi.

```
function out = selTur(populatie,nrSel,calitate,nrCrom,n)
k=30;
out=zeros(nrSel,nrCrom);
cand=zeros(k,1);
for i=1:k
    gasit=1;
    while gasit
        cand(i)=randi(n);
        if length(unique(cand(1:i)))==i
            gasit=0;
        end
    end
end
gasit=1;
while gasit
    gasit=0;
    for i=1:k-1
        if(calitate(cand(i))<calitate(cand(i+1)))
            cand(i)=cand(i)+cand(i+1);
            cand(i+1)=cand(i)-cand(i+1);
            cand(i)=cand(i)-cand(i+1);
            gasit=1;
        end
    end
end
for i=1:nrSel
    out(i,:)=populatie(cand(i),:);
end
end
```

Figura 2.6: Selecţie tip turneu **selTur**

2.1.1.5 Bucla principală

Primul lucru pe care bucla principală îl face este să creeze o nouă populaţie din cea anterioară. Acest lucru îl face selectând părinţii doi câte doi cu ajutorul funcţiei **selTur** şi îi recombină cu ajutorul funcţiei de încrucişare cu un singur punct **recom1p**. După crearea noii populaţii, algoritmul calculează calitatea fiecărui candidat nou şi cu aceasta caută un maxim nou. Dacă maximul este acelaşi ca la iteraţia anterioară, se incrementează cu unu contorul de repetiţii al maximului, altfel contorul este resetat la valoare unu şi noul maxim este salvat în variabila **maxim**.

Bucla principală **while** se va opri după numărul maxim de iteraţii sau dacă un anumit individ cu valoare maximă se repetă ca maxim al unei generaţii de numărul maxim de repetiţii **maxRepetitii**.

```
calMax=-inf;
maxim=zeros(1,nrCrom);
repMax=0;
it=1;
val=zeros(n,1);
popNou=pop;
for i=1:n
    val(i)=bi2fl(pop(i,:),int+1)+a;
end
cal=f(val);
while it<=maxIt&&repMax<maxRepetitii
    calMaxCurent=maxim;
    for i=1:2:n
    parinti=selTur(pop,2,cal,nrCrom,n);
    [popNou(i,:),popNou(i+1,:)]=recom1p(parinti(1,:),parinti(2,:),nrCrom);
        for j=1:nrCrom
            if(rand<=pMut)
                popNou(i,j)=mod(popNou(i,j)+1,2);
            end
            if(rand<=pMut)
                popNou(i+1,j)=mod(popNou(i+1,j)+1,2);
            end
        end
    end
    pop=popNou;
    for i=1:n
        val(i)=bi2fl(pop(i,:),int+1)+a;
    end
    cal=f(val);
    for i=1:n
        if cal(i)>calMax
            calMax=cal(i);
            maxim=pop(i,:);
            repMax=1;
        end
    end
    if calMaxCurent==calMax
        repMax=repMax+1;
    end
    it=it+1;
end
x=bi2fl(maxim,int+1)+a
f(x)
```

Figura 2.7: Bucla principală

2.1.2 Rezultate

Se poate observa din tabelul de mai jos că algoritmul dă un rezultat constant, care este acelaşi cu cel adevărat, maximul fiind f(x)=3.4918 cu x=2.2979.

Nr. Crt	Timp(s)	x	f(x)
1	13.13	2.2927	3.4918
2	13.18	2.2927	3.4918
3	13.18	2.2927	3.4918
4	13.24	2.2927	3.4918
5	13.32	2.2927	3.4918
6	13.19	2.2927	3.4918
7	13.18	2.2927	3.4918
8	13.14	2.2927	3.4918
9	13.18	2.2927	3.4918
10	13.19	2.2927	3.4918

2.2 Găsirea maximului pentru o funcţie cu două variabile

Pentru acest tip de problemă vom lua ca exemplu $f(x, y) = \cos(x^2) + e^{\sin(xy)} - \sin(y)$ pe domeniul $[-3, 3] \times [-3, 3]$. Această funcţie creează o suprafaţă foarte neregulată. Maximul acestei funcţii pe intervalele studiate este de 4.7181 in punctul $(-2.5062, -0.6294)$

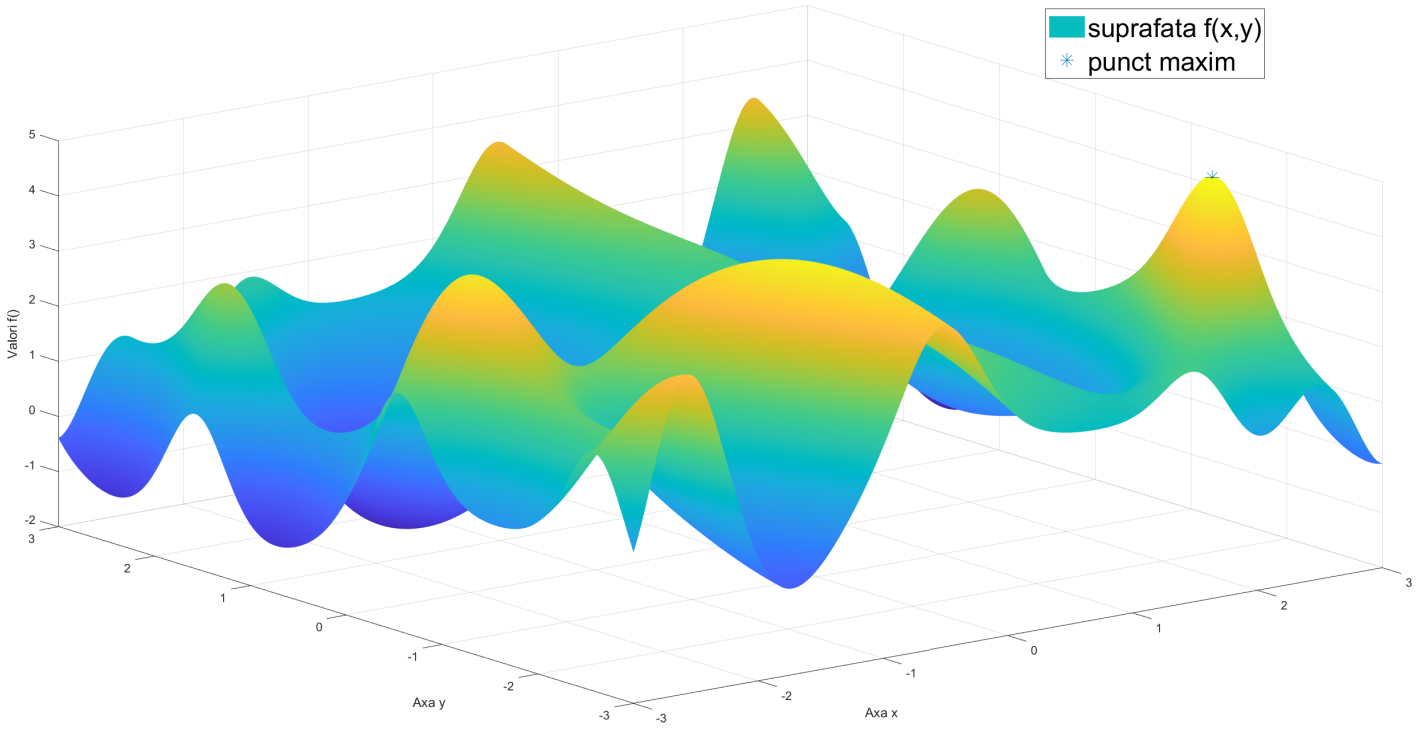


Figura 2.8: $f(x, y) = \cos(x^2) + e^{\sin(xy)} - \sin(y)$

Am adăugat şi proiecţiile pe planele xz şi yz pentru a fi clar unde se află maximul funcţiei studiate.

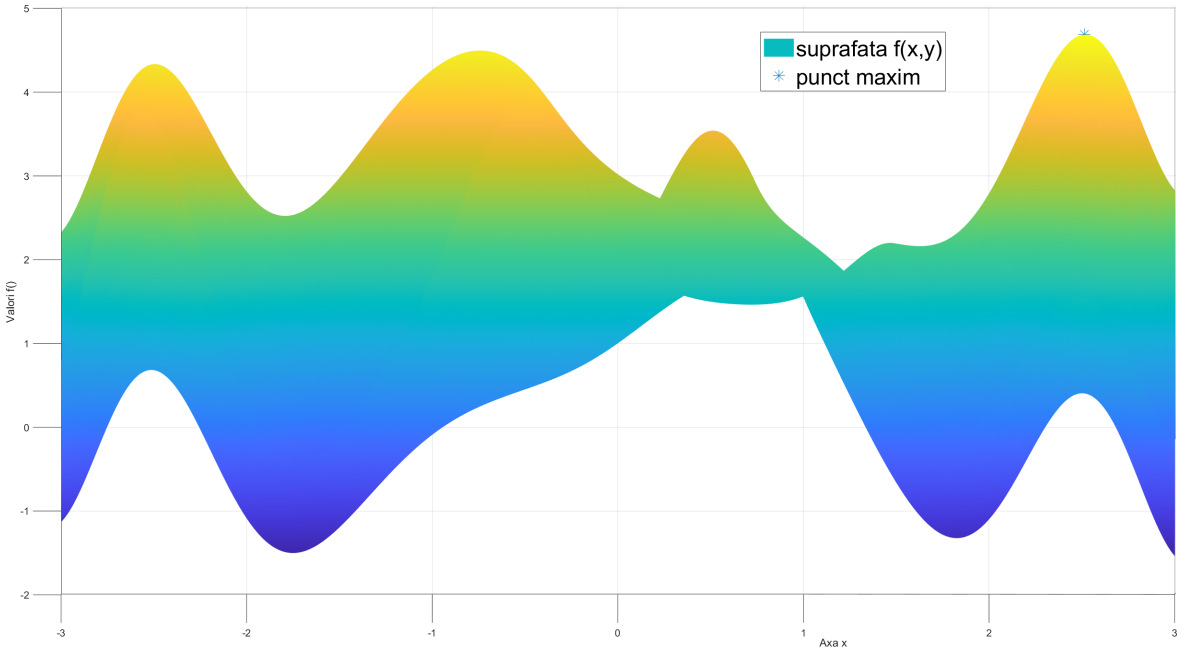


Figura 2.9: Vedere plan fx

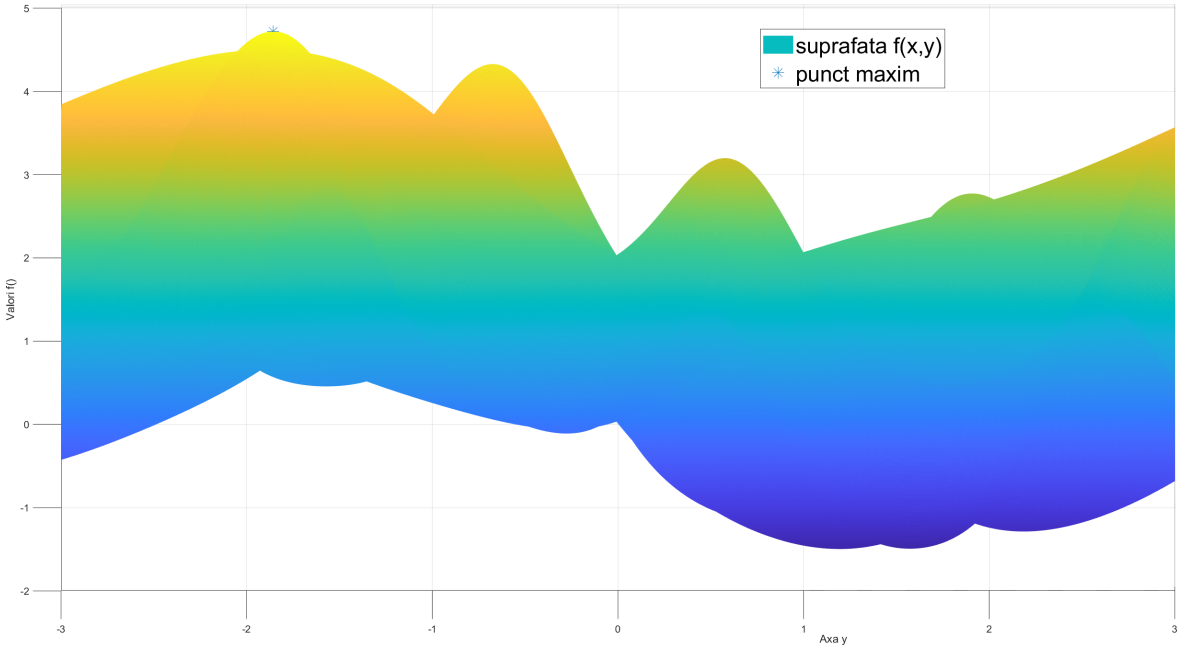


Figura 2.10: Vedere plan fy

Maximul adevărat cu primele 3 zecimale corecte a fost găsit cu ajutorul unui program simplu de calcul. Acest program caută valoarea maximă în puncte echidistante, cu distanţa minimă între două puncte fiind 0.0001, folosind două bucle **for**. Pentru această funcţie, algoritmul a avut nevoie de 6 minute în medie pentru a găsi maximul.

```
f=@(x,y) cos(x.^2)+exp(sin(x*y))-sin(y.^(-1));
a1=-3;
b1=3;
a2=-3;
b2=3;
maxim=-inf;
sol=zeros(1,2);
tic
for i=a1:0.0001:b1
    for j=a2:0.0001:b2
        if f(i,j)>maxim
            maxim=f(i,j);
            sol(:)=[i,j];
        end
    end
end
toc
maxim
sol
```

Figura 2.11: Algoritm căutare maxim

2.2.1 Program

Algoritmul genetic pentru această problemă va utiliza selecţia de tip turneu (**selTur** fig. 2.6) deoarece este nevoie ca valoarea calităţii să poată lua valori negative. Pentru a mări şansele de a crea indivizi diferiţi faţă de părinţi, am apelat la algoritmul de recombinare cu două puncte. Algoritmul de recombinare cu un singur punct era suficient pentru căutarea maximului unei funcţii cu o singură variabilă, acesta lucrând cu un singur număr binar. Recombinarea cu două puncte poate să amestece biţii din ambele numere binare, deşi nu se va întâmpla mereu asta.

```
function [out1 , out2] = recom2p(par1,par2,nrCrom)
out1=zeros(1,nrCrom);
out2=zeros(1,nrCrom);
pct1=randi([1,nrCrom],1);
pct2=randi([1,nrCrom],1);
if pct1>pct2
    pct1=pct1+pct2;
    pct2=pct1-pct2;
    pct1=pct1-pct2;
out1=par1;
out2=par2;
```

- for i=pct1:pct2
- out1(i)=par2(i);
- out2(i)=par1(i);
- end
- end

Figura 2.12: Recombinare cu două puncte **recom2p**

2.2.1.1 Date iniţiale

La începutul algoritmului sunt definite intervalele $[a1, b1]$ şi $[a2, b2]$ în care sunt cuprinse x şi respectiv y. Dimensiunea populaţiei şi numărul maxim de iteraţii sunt de asemenea definite la început, acestea influenţând timpul de rulare şi şansele de a obţine o valoare cât mai apropiată de valoarea adevărată. Şansa de mutaţie este setată la 0.1. Parametrul precizie (**prec**), ca în programul anterior, determină dimensiunea sectorului alocat părţii fracţionare pentru fiecare număr; în acest caz sunt două numere, respectiv două sectoare a câte 32 de biţi. După ce are aceste date şi funcţia de studiat, programul determină numărul de cromozomi necesari pentru a acoperi întregul domeniu şi alocă memorie matricei populaţie.

- n=10000;
- maxIt=100;
- f=@(x,y) cos(x.^2)+exp(sin(x.*y))-sin(y.^(-1));
- a1=-3;
- b1=3;
- a2=-3;
- b2=3;
- pMut=0.1;
- prec=32;
- maxRepetitii=20

Figura 2.13: Date iniţiale problema maxim cu doi parametri

2.2.1.2 Iniţializarea populaţiei

Programul, având datele de intrare, determină numărul de cromozomi necesari pentru a acoperi întreg domeniul şi alocă memorie matricei populaţie.

Cromozomul unui individ din populaţie este despărţit astfel: primii **int1** cromozomi alcătuiesc partea întreagă a primului număr, urmaţi de o secventă de **prec** cromozomi alocaţi părţii fracţionare a primului număr. Următorii **int2** cromozomi sunt dedicaţi părţii întregi a celui de-al doilea număr, iar restul de **prec** cromozomi sunt dedicaţi părţii fracţionare a celui de-al doilea număr.

Populaţia este iniţializată foarte asemănător cu problema de dinainte, diferenţa constând în faptul că trebuie verificat că ambele numere au fost generate în intervalele cerute de problemă.

```
int1=ceil(log2(abs(b1-a1)));
int2=ceil(log2(abs(b2-a2)));
nrCrom=int1+int2+prec*2;
pop=zeros(n,nrCrom);
for i=1:n
    valid=0;
    while valid==0
        pop(i,:)=randi([0 1],1,nrCrom);
        nr1=bi2fl(pop(i,1:int1+prec),int1+1)+a1;
        nr2=bi2fl(pop(i,int1+prec+1:nrCrom),int2+1)+a2;
        if (a1<=nr1)&&(nr1<=b1)&&(a2<=nr2)&&(nr2<=b2)
            valid=1;
        end
    end
end
```

Figura 2.14: Iniţializarea populaţiei

2.2.1.3 Bucla principală

Având populaţia iniţială creată, algoritmul determină valoarea de calitate a fiecărui individ, o salvează în vectorul **cal** şi determină maximul primei generaţii.

După aceea, în bucla **while** generează următoarea generaţie a populaţiei şi calculează calitatea fiecărui candidat din aceasta, determinând în acelaşi timp cea mai bună soluţie. Selectarea părinţilor pentru noua populaţie se face cu ajutorul algoritmului de selecţie de tip turneu **selTur** (fig. 2.6) şi recombinarea se face cu ajutorul algoritmului de recombinare cu două puncte (**recom2p** fig. 2.12).

```
calMax=-inf;
maxim=zeros(1,nrCrom);
repMax=0;
it=1;
val=zeros(n,2);
popNou=pop;
for i=1:n
    val(i,1)=bi2fl(pop(i,1:int1+prec),int1+1)+a1;
    val(i,2)=bi2fl(pop(i,int1+prec+1:nrCrom),int2+1)+a2;
end
cal=f(val(:,1),val(:,2));
while it<=maxIt&&repMax<maxRepetitii

    calMaxCurent=maxim;
    for i=1:2:n
    parinti=selTur(pop,2,cal,nrCrom,n);
    [popNou(i,:),popNou(i+1,:)]=recom2p(parinti(1,:),parinti(2,:),nrCrom);
        for j=1:nrCrom
            if(rand<=pMut)
                popNou(i,j)=mod(popNou(i,j)+1,2);
            end
            if(rand<=pMut)
                popNou(i+1,j)=mod(popNou(i+1,j)+1,2);
            end
        end
    end
    pop=popNou;
    for i=1:n
    val(i,1)=bi2fl(pop(i,1:int1+prec),int1+1)+a1;
    val(i,2)=bi2fl(pop(i,int1+prec+1:nrCrom),int2+1)+a2;
    if  ((a1<=val(i,1))&&(val(i,1)<=b1)&&(a2<=val(i,2))&&(val(i,2)<=b2))
        pop(i,:)=zeros(1,nrCrom);
        val(i,:)=[a1,a2];
    end
    end
    cal=f(val(:,1),val(:,2));
    for i=1:n
        if cal(i)>calMax
            calMax=cal(i);
            maxim=pop(i,:);
            repMax=1;
        end
    end
```

```
    if calMaxCurent==calMax
        repMax=repMax+1;
    end
    it=it+1;
end
calMax
sol=[bi2fl(maxim(1:int1+prec),int1+1)+a1,bi2fl(maxim(int1+prec+1:nrCrom),int2+1)+a2]
```

2.2.2 Rezultate

Rulând algoritmul de 10 ori pentru a putea observa deviaţia standard şi eraorea aboslută medie se constată ca acestea sunt foarte mici. Eroarea absolută este comparată cu valoarea rezultată din algoritmul de maxim simplu cu pasul iteraţiilor egal cu 0.0001.

Pas Iteraţie	Timp(s)	x	y	Valoare Maxim
0.0001	297.37 s	2.5149	-1.8573	4.6743
0.00001	58546.39	2.5149	-1.85733	4.67432

Figura 2.15: Valori obţinute cu algoritmul simplu

Nr. crt.	Timp(s)	x	y	Valoare Maxim
1	146.65	2.514898	-1.857304	4.674322
2	160.56	2.514919	-1.857324	4.674322
3	149.42	2.514913	-1.857346	4.674322
4	151.26	2.514883	-1.857380	4.674322
5	145.55	2.514817	-1.857342	4.674321
6	152.41	2.514894	-1.857278	4.674322
7	152.94	2.514898	-1.857375	4.674322
8	148.91	2.514903	-1.857321	4.674322
9	144.58	2.514853	-1.857346	4.674322
10	148.79	2.514921	-1.857282	4.674322
Valoare Medie	150.107	2.5148899	-1.8573298	4.6743219
	Media erorii absolute	0.0000101	0.0000298	0.0000219

Figura 2.16: Valori obţinute cu algoritmul genetic şi erori

Eroarea medie absolută faţă de punctul de maxim calculat atât pentru x cât şi pentru y este de ordinul 10^{-5}. Din acest fapt şi din observaţia anterioară, se poate vedea că primele 3 zecimale ale rezultatelor obţinute cu ajutorul algoritmului genetic sunt corecte. Aceste rezultate au fost obţinute în jumătate din timpul în care algoritmul de maxim cu pasul de iteraţie 0.0001 ar fi obţinut un rezultat la fel de apropiat.

Din acestea se observă că, în cazul acestei probleme, algoritmul genetic este mult mai rapid.

2.3 Găsirea maximului pentru o funcţie cu N variabile

Pentru demonstrarea algoritmului generalizat, vom lua funcţia de 5 variabile

$$f : D \to \mathbb{R}; f(x_1, x_2, x_3, x_4, x_5) = (x_1 + 1)^2 + cos(x_2) + sin(x_3)^2 - x_4^{-1} * x_5$$

$$\text{Cu } D = ([-2,2] \bigcup [-1,0] \bigcup [0,2] \bigcup [3,5] \bigcup [-1,3]).$$

Această problemă necesită foarte mult timp pentru a găsi o valoare de maxim în moduri clasice, acestea fiind ineficiente([3]). Prin verificarea fiecărei valori posibile cu un increment de 0.01, a fost găsit un maxim de 11.3333 în punctul (2, 0, 1.57, 3, -1), dar timpul de rulare a fost de 22 de ore. Dacă se doreşte mărirea preciziei ar fi nevoie de zile întregi pentru a rula un algoritm de increment 0.001, deoarece complexitatea unui astfel de algoritm este $O(n^5)$.

2.3.1 Program

Programul va folosi ca funcţie de calitate funcţia f în sine, va utiliza selecţia tip turneu (**selTur 2.6**), aceasta permiţând folosirea calităţii negative.

2.3.1.1 Date iniţiale şi iniţializare

Programul primeşte de la început funcţia cu care va lucra şi intervalele în care se află variabilele. Acesta numară câte variabile sunt şi câtă memorie trebuie alocată cu ajutorul vectorilor **a** şi **b**, aceştia reprezentând limita inferioară şi pe cea superioară a intervalelor de definire ale variabilelor. Aşa se obţine vectorul **int** ce stochează numărul de biţi alocaţi fiecărui număr pentru partea întreagă şi numărul de total de cromozomi ai unui individ (**nrCrom**).

Iniţializarea populaţie de start se face cu ajutorul funcţie MatLab **randi**; în acelaşi timp se asigură că valorile obţinute sunt în domeniul de definiţie. În timpul iniţializării se calculează calitatea fiecărui individ. După iniţializare se determină cel mai bun candidat din populaţia iniţială.

Aici mai sunt definite şi alte variabile cum ar fi numărul de iteraţii ale algoritmului, probabilitatea de mutaţie, probabilitatea de interschimbare a cromozomilor părinţilor, numărul de cromozomi alocaţi părţii binare a candidaţilor şi numărul de părinţi selectaţi de către algoritmul de selecţie.

```
format long
dimPop=10000;
maxIt=100;
f=@(x) (x(1)+1).^2+cos(x(2))+sin(x(3)).^2-x(4).^(-1)*x(5);
a=[-2 -1 0 3 -1 ];
b=[2 0 2 5 3];
int=zeros(1,length(a));
nrCrom=0;
prec=24;
pInter=0.5;
for i=1:length(a)
int(i)=ceil(log2(abs(b(i)-a(i))));
nrCrom=nrCrom+prec+int(i);
end
nrVar=length(a);
maxRepetitii=20;
pMut=0.1;
val=zeros(nrVar,1);
pop=zeros(dimPop,nrCrom);
cal=zeros(dimPop,1);
nrSel=2;
for i=1:dimPop
    valid=1;
    gasit=1;
    pop(i,:)=randi([0,1],1,nrCrom);
    stg=1;
    for j=1:nrVar
        valid=1;
        dr=stg+int(j)+prec-1;
        while valid
            valid=0;
            val(j)=bi2fl(pop(i,stg:dr),int(j)+1)+a(j);
            if a(j)>val(j)||b(j)<val(j)
                pop(i,stg:dr)=randi([0,1],int(j)+prec);
                valid=1;
            end
        end
        stg=dr+1;
    end
    cal(i)=f(val);
end
[valMaxim,ind]=max(cal);
maxim=pop(ind,:);
repMax=1;
it=1;
```

2.3.1.2 Încrucişare uniformă

Funcţia de încrucişare uniformă primeşte doi părinţi prin variabila matriceală **parinti** şi probabilitatea de interschimbare a cromozomilor prin variabila **pI**. Această funcţie întoarce matricea **copii** ce conţine pe fiecare linie cromozomii celor doi copii obţinuţi.

```
function copii = recomUnif(parinti,pI)
copii=parinti;
for i=1:size(parinti,2)
    if rand<=pI
        copii(1,i)=parinti(2,i);
        copii(2,i)=parinti(1,i);
    end
end
end
```

Figura 2.17: Recombinare Uniformă

2.3.1.3 Bucla principală

Bucla principală creează o nouă populaţie selectând doi părinţi cu funcţia **selTur**(fig. 2.6), îi recombină cu funcţia **recomUnif**(fig. 2.6), aplică procedeul de mutaţie pe copiii obţinuţi şi la final îi adăugă în populaţie.

Pentru a păstra cei mai buni candidaţi dintr-o iteraţie este introdus elitismul (secţiunea 1.5.4). Algoritmul va adăuga cei mai buni **nrElite** indivizi (în cazul acesta 10) din populaţia veche în populaţia nouă. Procesul de selecţie al indivizilor se face cu ajutorul sortării vectorului calitate, acest lucru fiind obţinut prin apelul funcţiei **sort** ([tmp,ind] =sort(cal,'descend');). Valorile stocate în **ind** din apelul funcţiei **sort** sunt indici ordonaţi ai populaţiei ordonate descrecător după calitate, iar argumentul **tmp** nu are importaţă, deoarece nu avem nevoie de vectorul calitate ordonat.

Cu ajutorul conţinutului vectorului **ind** se obţin cei mai buni indivizi din populaţia iniţială, metoda folosindu-se şi în bucla principală.

După crearea noii generaţiic programul converteşte datele binare stocate de cromozomii candidaţilor în valori zecimale pentru a calcula calitatea indivizilor şi pentru căutarea unei soluţii mai bune. Dacă aceeaşi soluţie se repetă de mai multe ori bucla, se opreşte mai devreme.

După terminarea buclei principale programul afişează în consolă valoarea maximă şi punctul în care s-a obţinut aceasta.

```
while it<=maxIt&&repMax<maxRepetitii
    popNou=pop;
    maxCurent=maxim;
    for i=1:2:dimPop
        parinti=selTur(pop,nrSel,cal,nrCrom,dimPop);
        popNou(i:i+1,:)=recomUnif(parinti,pInter);
        for j=1:nrCrom
            if(rand<=pMut)
                popNou(i,j)=mod(popNou(i,j)+1,2);
            end
            if(rand<=pMut)
                popNou(i+1,j)=mod(popNou(i+1,j)+1,2);
            end
        end
    end
    pop=popNou;
    for i=1:dimPop
        stg=1;
        for j=1:nrVar
            valid=1;
            dr=stg+int(j)+prec-1;
            val(j)=bi2fl(pop(i,stg:dr),int(j)+1)+a(j);
            stg=stg+int(j)+prec;
        end
        cal(i)=f(val);
        if cal(i)>valMaxim
            valMaxim=cal(i);
            maxim=pop(i,:);
            repMax=1;
        end
    end
    if maxCurent==maxim
        repMax=repMax+1;
    else
        repMax=1;
    end
    it=it+1;
end
maxim;
valMaxim
```

- stg=1;
- for j=1:nrVar
- valid=1;
- dr=stg+int(j)+prec-1;
- val(j)=bi2fl(maxim(stg:dr),int(j)+1)+a(j);
- stg=stg+int(j)+prec;
- end
- val

2.3.2 Rezultate

Pentru testări repetate am luat dimensiunea populaţiei de 10000 de indivizi, cu un număr de 100 iteraţii, numărul de biţi alocaţi părţii binare fiind 24. Aceste valori au fost alese pentru a avea un timp redus de rulare.

Nr. crt.	Timp(s) Rulare	x1	x2	x3	x4	x5	Valoare maxim
1	80.57	1.999997	-0.016293	1.58745	3.001657	-0.996694	11.332641
2	81.32	1.999839	-0.00343	1.580692	3.004409	-0.999574	11.33163763
3	56.64	1.999915	-0.033611	1.57502	3.001136	-0.994551	11.3303
4	64.57	1.999964	-0.014225	1.547028	3.003632	-0.996856	11.331003
5	60.38	1.999915	-0.016746	1.575348	3.005529	-0.998059	11.331408
6	65.09	1.99996	-0.036757	1.570221	3.009053	-0.998972	11.331074
7	73.97	1.999972	-0.017887	1.58505	3.000047	-0.999124	11.332508
8	87.2	1.999876	-0.000861	1.574012	3.001061	-0.999819	11.332401
9	56.02	1.999959	-0.019165	1.573626	3.004105	-0.99951	11.332277
10	93.34	1.999948	-0.019232	1.550932	3.001153	-0.9989467	11.331967
Val. Med.	71.91	1.9999345	-0.0178207	1.5719379	3.0031782	-0.99821057	11.3317216
Dev. Standard		0.00004821	0.0111697	0.0132588	0.00273846	0.001687891	0.00077145

Mărind populaţia la 100000 de indivizi, algoritmul a produs rezultatul 11.332293201465934 în punctul (1.99995225, -0.02100956, 1.58513551, 3.0026758, -0.99996685). Timpul de rulare în acest caz a fost de 894.24 secunde (aproximativ 14 minute).

Comparând rezultatele obţinute cu cele ale algoritmului simplu de căutare, putem observa că sunt asemănătoare, acestea având prima zecimală comună. Pentru a obţine rezultate mai bune:

- se poate renunţa la condiţia de repetiţie a maximului, crescând şansele de a apărea mutaţii sau combinări noi;
- se poate creşte mărimea populaţiei pentru a creşte şansele de generare a unei soluţii mai bune, dar acest lucru necesită mai multă memorie şi mai mult timp de procesare;
- se pot ajusta şansele de mutaţie şi în cazul acesta de recombinare uniformă. Acest lucru poate avea impact negativ, modificând prea mult candidaţii pentru a putea converge la un punct de maxim;
- poate fi crescut numărul de iteraţii, dând timp algoritmului să genereze o soluţie candidat mai bună.

Capitolul 3

Aplicaţii diverse pentru algoritmi genetici

3.1 Problema rucsacului

Se dau diferite obiecte, fiecare cu valoarea şi greutatea sa. Se cere a se găsi într-o anumită greutate valoarea maximă aleagând din obiectele date. Pentru acest exemplu luăm greutatea maximă egală cu 55 şi următoarele valori ale proprietăţilor obiectelor.

Nr. obiect	1	2	3	4	5	6	7	8	9	10	11
Greutăţi	8	7	10	11	2	5	11	10	3	2	8
Profit	48	28	7	8	13	43	13	41	13	47	13

3.1.1 Program

Pentru recombinare va fi folosit algoritmul de recombinare cu două puncte prezentat anterior (**recom2p** Fig. 2.12) iar pentru seleceţie va fi folosit algoritmul de selecţie tip turneu (**selTur** Fig. 2.6), deoarece acesta poate lucra cu mai multe valori de calităţi egale.

3.1.1.1 Date iniţiale

Programul are declarate pentru început greutatea maximă, dimensiunea populaţiei, numărul maxim de iteraţii, probabilitatea de mutaţie si doi vectori **greutati** şi **profit**, aceştia având greutăţiile si valorile de câstig ale fiecarui obiect.

Numărul de cromozomi este determinat din lungimea vectorului **greutati**. Cromozomii unui individ sunt organizaţi astfel: 0 însemnă că obiectul nu este în rucsac, iar 1 înseamnă că este în rucsac. Populaţia este iniţializată cu ajutorul funcţiei predefinite MatLab **randi([0,1],dimPop,nrCrom)**, apelul acesteia creând o matrice de valori 0,1 de **dimPop** linii şi **nrCrom** coloane.

Calitatea unui individ este calculată adunând profitul fiecărui obiect găsit în soluţia candidat, dar dacă greutatea totală depăşeşte greutatea maximă, atunci calitatea este 0.

```
maxIt=100;
dimPop=100;
pMut=0.1;
maxRep=100;
greutati=[8 7 10 11 2 5 11 10 3 2 8 ];
profit=[48 28 7 8 13 43 13 41 13 47 18 ];
greutateMax=55;
nrCrom=length(profit);
pop=randi([0,1],dimPop,nrCrom);
cal=zeros(dimPop,1);
for i=1:dimPop
    cal(i)=sum(profit.*pop(i,:));
    if sum(greutati.*pop(i,:))>greutateMax
        cal(i)=0;
    end
end
```

Figura 3.1: Date Iniţiale Problemă rucsac

Înainte de a intra în bucla principală, algoritmul determină cel mai bun individ din populaţia iniţială.

```
[valMaxim,ind]=max(cal);
maxim=pop(ind,:);
```

3.1.1.2 Bucla principală

Algoritmul principal se află în interiorul buclei **while**. Acesta creează o populaţie nouă cu ajutorul selecţiei de tip turneu şi al recombinării cu două puncte, aplicând procedeul de mutaţie asupra indivizilor înainte de a fi introduşi în populaţia nouă.

După crearea întregii populaţii, algoritimul determină calitatea acesteia prin procedeul descris anterior şi determină cel mai bun candidat din populaţia curentă.

```
while it<maxIt&&repetitii<maxRep
    popNou=zeros(dimPop,nrCrom);
    for i=1:2:dimPop-1
        parinte1=selTur(pop,1,cal,nrCrom,dimPop);
        parinte2=selTur(pop,1,cal,nrCrom,dimPop);
        [popNou(i,:),popNou(i+1,:)]=recom2p(parinte1,parinte2,nrCrom);
        for j=1:nrCrom
            if(rand<=pMut)
                popNou(i,j)=mod(popNou(i,j)+1,2);
            end
            if(rand<=pMut)
                popNou(i+1,j)=mod(popNou(i+1,j)+1,2);
            end
        end
    end
```

```
    pop=popNou;
    for i=1:dimPop
        cal(i)=sum(profit.*pop(i,:));
        if sum(greutati.*pop(i,:))>greutateMax
            cal(i)=-1;
        end
        if cal(i)== valMaxim
            if maxim==pop(i,:)
                repetitii=repetitii+1;
            else
                repetitii=1;
            end
        end
        if cal(i)>valMaxim
            valMaxim=cal(i);
            maxim=pop(i,:);
            repetitii=1;
        end
    end
    test(it)=max(cal);
end
contor=1;
solutie=0;
for i=1:nrCrom
    if maxim(i)
        solutie(contor)=i;
        contor=contor+1;
    end
end
solutie
valMaxim
```

După terminarea buclei principale este format vectorul **solutie**, ce conţine numărul de ordine al fieccărui element inclus în soluţia optimă, si îl afişează împreună cu profitul maxim salvat în variabila **valMax**.

3.1.2 Rezultate

Pentru exemplul luat, rezultatele au fost consistente şi rapide.

Nr. Crt	Timp(s)	Vector Soluţii	Profit
1	1.04	(1, 2, 3, 5, 6, 8, 9, 10, 11)	258
2	1.03	(1, 2, 3, 5, 6, 8, 9, 10, 11)	258
3	1.02	(1, 2, 3, 5, 6, 8, 9, 10, 11)	258
4	1.04	(1, 2, 3, 5, 6, 8, 9, 10, 11)	258
5	1.04	(1, 2, 3, 5, 6, 8, 9, 10, 11)	258

3.2 Soluţii pentru ecuaţii cu coeficienţi întregi

Se dă o ecuaţie de tipul $a + 5b + 4c + 3d + 6e = 100$. Pentru a avea un algoritm mai eficient vom caută numai numere întregi de la 0 la 127 (cel mai apropiat număr binar mai mare de 100).

Pentru a crea funcţia de calitate, este construită funcţia

$$f(a, b, c, d, e) = a + 5b + 4c + 3d + 6e - 100$$

ce va fi folosită pentru a calcula calitatea unei soluţii candidat. Calitatea va fi inversul $|f| + 1$.([4])

$$calitate(individ) = \frac{1}{1 + |f(individ)|}$$

3.2.1 Program

Soluţia va fi stocată într-un vector, împărţit în segemente egale pentru fiecare număr.

3.2.1.1 Date iniţiale şi iniţializare

Programul începe cu numărul de cromozomi alocat pentru fiecare număr dintr-o soluţie candidat (**int**) şi numărul de necrunoscute (**nrCrom**). Cu ajutorul acestor variabile determină numărul total de cromozomi ai unui individ. Populaţia este iniţializată cu ajutorul funcţiei MatLab **randi**. În timpul iniţializării este calculată calitatae indivizilor generaţi cu ajutorul funcţiei **bi2fl** (Fig2.1), aceasta fiind folosită pentru a transforma numerele binare în întregi.

```
f=@(x) x(1)+5*x(2)+4*x(3)+3*x(4)+6*x(5)-100;
maxIt=50;
dimPop=10000;
pMut=0.1;
int=8;
nrNec=5;
nrCrom=nrNec*int;
val=zeros(nrNec,1);
pop=zeros(dimPop,nrCrom);
cal=zeros(dimPop,1);
nrSel=5;
pInter=0.4;
for i=1:dimPop
    gasit=1;
    pop(i,:)=randi([0,1],1,nrCrom);
    for j=1:nrNec
        val(j)=bi2fl(pop(i,(int)*(j-1)+1:(int)*j),int+1);
    end
    cal(i)=1/(1+abs(f(val)));
end
```

Figura 3.2: Date iniţiale

3.2.1.2 Selecţie tip ruletă

Programul are funcţia de calitate strict pozitivă, acest lucru permiţând utilizarea unui algoritm de selectie multiplă de tip ruletă.

Numărul de părinţi selectaţi de algoritm este predefinit de variabila **nrSel (3.2)** din programul principal.

```
function [selectati] = selRulN(parinti,cal,nrSel)
dimPop=size(parinti,1);
nrCrom=size(parinti,2);
s=0;
selectati=zeros(nrSel,nrCrom);
for i=1:dimPop
    s=s+cal(i);
end
for i=1:nrSel
    k=s*rand;
    ind=2;
    val=cal(1);
    if k<=cal(1)
        selectati(i,:)=parinti(1,:);
    else
        gasit=1;
        while gasit
            val=val+cal(ind);
            if val>=k
                selectati(i,:)=parinti(ind,:);
                gasit=0;
            end
            ind=ind+1;
        end
    end
end
end
```

Figura 3.3: Algoritm de selecţie multiplă tip ruletă

3.2.1.3 Recombinare uniformă a N părinţi

Programul va folosi un algoritim de recombinare uniformă, deoarece, având multe numere într-un individ, un algoritm de recombinare simplu nu ar fi amestecat toate numerele. Algoritmul amestecă toţi cromozomii părtinţilor de pe o poziţie şi le distribuie aleator copiilor. Şansa ca amestecul să se întâmple pe o poziţie este dată de variabila de intrare **pI**.

```
function [copii] = recUnifMultip(parinti,pI)
copii=parinti;
nrPar=size(parinti,1);
nrCrom=size(parinti,2);
a=zeros(nrCrom,1);
b=zeros(nrPar,1);
for i=1:nrCrom
    if(rand>pI)
        for j=1:nrPar
            a=copii(j,:);
            b(j)=a(i);
        end
        for j=nrCrom:2
            k=randi([1,j]);
            b(j)=b(j)+b(k);
            b(k)=b(j)-b(k);
            b(k)=b(j)-b(k);
        end
        for j=1:nrPar
            a=copii(j,:);
            a(i)=b(j);
            copii(j,:)=a;
        end
    end
end
end
```

Figura 3.4: Recombinare uniformă

3.2.1.4 Algoritm principal

După iniţializarea populaţiei, programul determină maximul populaţiei iniţiale şi începe bucla principală **while**.

În bucla principală, algoritmul crează o nouă populaţie cu ajutorul funcţiei de selecţie de tip ruletă (**selRulN** Fig. 3.2.1.2) şi funcţia de recombinare uniformă (**recUnifMultip** Fig. 3.2.1.3).

În urma creării noii generaţii de soluţii candidat, programul determină calitatea lor şi cea mai bună soluţie pâna la generaţia curentă.

La final, acesta întoarce soluţia găsită, aceasta fiind memorată în variabila **maxim**.

```
[valMaxim,ind]=max(cal);
maxim=pop(ind,:);
it=1;
while it<maxIt
    popNou=pop;
    for i=1:nrSel:dimPop
        parinti=selRulN(pop,cal,nrSel);
        copii=recUnifMultip(parinti,pInter);
        for j=1:nrCrom
            for k=1:nrSel
                if(rand<=pMut)
                    copii(k,j)=mod(copii(k,j)+1,2);
                end
            end
        end
        popNou(i:i+nrSel-1,:)=copii;
    end
    pop=popNou;
    for i=1:dimPop
        for j=1:nrNec
            val(j)=bi2fl(pop(i,(int)*(j-1)+1:(int)*j),int+1);
        end
        cal(i)=1/(1+abs(f(val)));
        if cal(i)>valMaxim
            maxim=val;
            valMaxim=cal(i);
        end
    end
    it=it+1;
end
maxim
```

Figura 3.5: Bucla Principală

3.2.1.5 Valori negative

Pentru a permite numere negative se mai adaugă câte un cromozom de paritate pentru fiecare număr din soluţie, acesta indicând prin zero că valoarea este pozitivă, prin 1, valoarea negativă. Pe prima poziţie din sectoarele individului alocate fiecărui număr din soluţie este cromozomul de paritate, iar celelalte poziţii determină modulul numărului.

Diferenţele între algoritmi sunt minime, schimbându-se numărul de cromozomi, algoritmul de calcul al valorilor efective şi al calităţii, ultimele două fiind calculate în aceeaşi buclă **for**.

```
nrCrom=nrNec*int+nrNec;
for i=1:dimPop
    gasit=1;

    pop(i,:)=randi([0,1],1,nrCrom);
    for j=1:nrNec
        val(j)=bi2fl(pop(i,(int+1)*(j-1)+2:(int)*j),int+1);
        if pop(i,(int)*(j-1)+1:(int)*j)==1
            val(j)=-val(j);
        end
    end
    cal(i)=1/(1+abs(f(val)));
end
```

3.2.2 Rezultate

Programul a oferit pentru exemplul luat soluţii diferite, toate valide şi într-un timp scurt.

Nr. Crt	Timp	Vector Soluţie	Validitate Soluţie
1	11.72	(15,1,2,12,6)	valid
2	11.71	(68,1,0,1,4)	valid
3	11.5	(39,11,0,0,1)	valid
4	11.51	(53,2,1,9,1)	valid
5	11.68	(2,6,5,16,0)	valid
6	11.62	(9,8,0,15,1)	valid
7	11.75	(20,7,3,3,4)	valid
8	11.65	(42,5,0,9,1)	valid
9	11.67	(53,1,3,4,3)	valid
10	11.6	(6,5,12,7,0)	valid

3.3 Problema comis-voiajorului

Se dă o lista de oraşe şi lungimile drumurilor dintre acestea, dacă există. Cerinţa este determinarea celei mai scurte rute ce trece prin toate oraşele şi se întoarce în oraşul de început.

Aceasta este o problemă de combinatorică, ceea ce face găsirea unei metode de încrucişare grea. De aceea nu va fi folosită încrucişarea, ci doar mai multe tipuri de mutaţie.

Deoarece soluţia este o buclă, soluţiile $[1, 5, 3, 6, 7, 2, 4]$ şi $[5, 3, 6, 7, 2, 4, 1]$ sunt echivalente. Din acest motiv, toate soluţiile vor începe în oraşul 1.

Pentru a uşura calculul calităţii soluţiile vor fi salvate şi cu nodul destinaţie ca de exemplu $[1, 5, 3, 6, 7, 2, 4, 1]$.

Distanţele dintre oraşe vor fi salvate într-o matrice a drumurilor. Pentru această rezolvare, dacă nu există drum între două oraşe distaţa între cele două va fi infinit.

Pentru demonstraţie vom lua matricea drumurilor:

Oraş	1	2	3	4	5	6	7	8	9
1	0	5	∞	3	7	8	10	13	∞
2	5	0	3	6	∞	2	5	5	5
3	∞	3	0	10	2	4	6	15	3
4	3	6	10	0	7	1	∞	6	6
5	7	∞	2	7	0	∞	8	7	8
6	8	2	4	1	∞	0	3	2	12
7	10	5	6	∞	8	3	0	∞	3
8	13	7	15	6	7	2	∞	0	4
9	∞	5	3	6	8	12	3	4	0

Programul lucrează cu matrici asimetrice. În acest caz particular am ales o matrice simetrică.

3.3.1 Program

Pentru această problemă am ales calitatea ca fiind inversul distanţei totale, astfel caliatea cea mai mare corespunde celui mai scurt drum; dacă este generat un drum imposibil (adică există un drum între două oraşe cu valoare infinită), calitatea acestuia va fi 0.

- function cal = calitate(individ,drumuri)
- val=0;
- for i=1:length(individ)-1
- val=val+drumuri(individ(i),individ(i+1));
- end
- cal=1/val;
- end

Figura 3.6: Funcţie calitate

3.3.1.1 Date iniţiale şi iniţializare

Iniţializarea indivizilor se face cu o funţie separată. Un individ va fi de forma $[1, a, b, c, d, ..., 1]$ cu $a, b, c, d, ...$ diferite intre ele şi diferite de 1, acestea luând valori distincte între 2 şi numărul de oraşe. Numărul de cromozomi ai unui individ este numărul de oraşe plus 1, deoarece este adăugat 1 la capăt.

```
function individ = init(nrCrom)
crom=ones(1,nrCrom);
crom(1)=1;
poz=2;
while(poz<nrCrom)
    crom(poz)=randi([2,nrCrom-1]);
    if poz==length(unique(crom))
        poz=poz+1;
    end
end
crom(poz)=1;
individ=crom;
end
```

Programul are setate de la început matricea de drumuri, numărul de iteraţii şi dimensiunea populaţiei. Aceasta determină de la început numărul de cromozomi din dimensiunea matricii de drumuri. Populaţia este iniţializată şi este determinat cel mai bun individ.

```
clear all;
drumuri=[ 0 5 inf 3 7 8 10 13 inf;
    5 0 3 6 inf 2 5 7 5;
    inf 3 0 10 2 4 6 15 3;
    3 6 10 0 7 1 inf 6 6;
    7 inf 2 7 0 inf 8 7 8;
    8 2 12 1 inf 0 3 2 12;
    10 5 6 inf 8 3 0 inf 3;
    13 7 15 6 7 2 inf 0 4;
    inf 5 3 6 8 12 3 4 0];
maxIt=20;
nrCrom=size(drumuri,1)+1;
dimPop=30;
pop=zeros(dimPop,nrCrom);
cal=zeros(1,dimPop);
for i=1:dimPop
    pop(i,:)=init(nrCrom);
    cal(i)=calitate(pop(i,:),drumuri);
end
[valMax,ind]=max(cal);
maxim=pop(ind,:);
```

3.3.1.2 Selecţie rang

Având în vedere calitatea pozitivă a unui individ, programul va utiliza selecţia cu rang, deoarece aceasta permite să fie aleşi şi indivizii cu calitate 0, dar având mult mai multe şanse să aleagă indivizi cu calitatea mare.

```
function [selectat] = selRang(candidati,cal)
[cal,ordine]=sort(cal);
dimPop=length(cal);
candOrd=zeros(size(candidati));
for i=1:dimPop
    candOrd(i,:)=candidati(ordine(i),:);
end
s=dimPop*(dimPop+1)/2;
k=randi([1,s],1);
val=1;
i=2;
gasit=1;
if(k==1)
    gasit=0;
    selectat=candOrd(1,:);
end
while(gasit)

    val=val+i;
    if val>=k
    gasit=0;
    selectat=candOrd(i,:);
    end
    i=i+1;
end
end
```

3.3.1.3 Bucla principală

După definirea datelor de start, programul începe bucla principală **while**. În această buclă el începe generarea unei noi generaţii, selectând cate un individ şi aplicându-i un proces de mutaţie sau introducându-l direct în populaţie. Procesele de mutaţie sunt

- interschimbarea a doi cromozomi aleatori ([2])
- permutarea unui număr aleatoriu de cromozomi din primele poziţii în ultimele
- inversarea ordinii de parcurgere, aceasta nu are mare efect în cazul în care matricea drumurilor este simetrică, acestea fiind neafectate de inversarea ordinii din cauză că drumul este egal în ambele sensuri. Totuşi afectează ce indivizi pot fi generaţi aplicând mutaţia din nou

După crearea noii generaţii, programul calculează calitatea acesteia şi determină dacă a fost găsit un nou maxim, apoi revenind la începutul buclei. Bucla se încheie când se ajunge la numărul maxim de iteraţii.

```
it=1;
while it<maxIt
    tempPop=pop;
    for i=1:dimPop
        parinte=selRang(pop,cal);
        switch randi([1,4])
            case 1
                pct1=randi([2,6]);
                pct2=randi([2,6]);
                copil=parinte;
                copil(pct1)=parinte(pct2);
                copil(pct2)=parinte(pct1);
            case 2
                pct=randi([2,6]);
                copil=parinte;
                copil(2:nrCrom+1-pct)=parinte(pct:nrCrom-1);
                copil(nrCrom+2-pct:nrCrom-1)=parinte(2:pct-1);
            case 3
                copil=flip(parinte);
            otherwise
                copil=parinte;
        end
        tempPop(i,:)=copil;
    end
    pop=tempPop;
    for i=1:dimPop
        cal(i)=calitate(pop(i,:),drumuri);
    end
    [maxCurent,ind]=max(cal);
    if maxCurent>valMax
        valMax=maxCurent;
        maxim=pop(ind,:);
    end
    it=it+1;
end
1/valMax
maxim
```

3.3.2 Rezultate

Din rezultatele de mai jos se poate observa că programul nu dă o soluţie optimă. Rezultatele acestea au fost obţinute cu dimensiunea populaţiei de 30 şi 20 de iteraţii. Aceste rezultate nu sunt cele mai satisfăcătoare, variind între 30 şi 37, dar au întors rezultatul după un timp de rularea de 0.035 secunde.

Nr.Crt	Distanţă drum	Timp(s)	Drum
1	30	0.035011	(1, 5, 3, 2, 7, 9, 8, 6, 4, 1,)
2	35	0.034753	(1, 5, 3, 7, 9, 8, 2, 6, 4, 1,)
3	37	0.034811	(1, 4, 8, 6, 2, 3, 9, 7, 5, 1,)
4	31	0.034996	(1, 2, 7, 9, 3, 5, 8, 6, 4, 1,)
5	37	0.035785	(1, 5, 7, 9, 3, 2, 6, 8, 4, 1,)
6	34	0.036824	(1, 2, 6, 8, 9, 7, 3, 5, 4, 1,)
7	33	0.035961	(1, 4, 8, 6, 2, 7, 9, 3, 5, 1,)
8	34	0.037493	(1, 4, 5, 3, 7, 9, 8, 6, 2, 1,)
9	30	0.034979	(1, 4, 6, 8, 9, 7, 2, 3, 5, 1,)
10	36	0.035354	(1, 5, 8, 9, 7, 3, 2, 6, 4, 1,)
11	36	0.034907	(1, 5, 3, 9, 7, 6, 2, 8, 4, 1,)
12	35	0.037596	(1, 2, 3, 9, 7, 5, 8, 6, 4, 1,)
13	34	0.035193	(1, 4, 6, 8, 5, 3, 7, 9, 2, 1,)
14	33	0.035222	(1, 4, 8, 6, 2, 7, 9, 3, 5, 1,)
15	35	0.035042	(1, 4, 5, 3, 6, 8, 9, 7, 2, 1,)

Dacă mărim dimensiunea populaţie la 100 şi numărul de iteraţii la 50, obţinem:

Nr.Crt	Distanţă drum	Timp(s)	Drum
1	31	0.12602	(1, 4, 6, 8, 9, 7, 5, 3, 2, 1)
2	30	0.13275	(1, 5, 3, 2, 7, 9, 8, 6, 4, 1)
3	30	0.11942	(1, 4, 6, 8, 9, 7, 2, 3, 5, 1)
4	30	0.11235	(1, 4, 6, 8, 9, 7, 2, 3, 5, 1)
5	30	0.12192	(1, 5, 3, 2, 7, 9, 8, 6, 4, 1)
6	30	0.11161	(1, 5, 3, 2, 7, 9, 8, 6, 4, 1)
7	30	0.11519	(1, 5, 3, 2, 7, 9, 8, 6, 4, 1)
8	31	0.11375	(1, 2, 3, 5, 8, 9, 7, 6, 4, 1)
9	30	0.10936	(1, 5, 3, 2, 7, 9, 8, 6, 4, 1)
10	30	0.11069	(1, 5, 3, 2, 7, 9, 8, 6, 4, 1)
11	30	0.1096	(1, 5, 3, 2, 7, 9, 8, 6, 4, 1)
12	31	0.12131	(1, 4, 6, 8, 9, 7, 5, 3, 2, 1)
13	31	0.11329	(1, 2, 3, 5, 8, 9, 7, 6, 4, 1)
14	30	0.12254	(1, 5, 3, 2, 7, 9, 8, 6, 4, 1)
15	31	0.11329	(1, 2, 3, 5, 8, 9, 7, 6, 4, 1)

Se observă că durata medie s-a triplat, dar, variaţia rezultatelor fiind mult mai mică, acestea aflându-se între 30 şi 31. Cu aceste modificări am obţinut un program mult mai precis.

Concluzii

Prin exemplele de mai sus am prezentat modalităţi de utilizare a algoritmilor genetici, avantaje şi dezavantaje. Aceşti algoritmi pot găsi soluţii aproximative la probleme care nu au metode rapide de rezolvare.

Una din problemele cele mai mari ale algoritmilor genetici este găsirea unei funcţii de calitate, după cum s-a putut observa în problema 3.2, unde a trebuit construită funcţia de calitate din ecuaţia dată.

O altă problemă este natura aleatorie a algoritmilor, ce nu garantează cea mai bună soluţie nici după multe iteraţii.

Pe de altă parte, algoritmii genetici pot fi modificaţi prin parametri simpli precum: rata de mutaţie, dimensiunea populaţiei, numărul de iteraţii, algoritmul de selecţie, algoritmul de recombinare, includerea elitismului. Aceste modificări permit o plajă mare de alegeri ce ajută la optimizarea şi la mularea algoritmului pe problema dată.

Pentru algoritmi care au deja metode de rezolvare, algoritmii genetici pot fi utilizaţi pentru a reduce timpul de calcul în schimbul acurateţii soluţiei.

Din cele prezentate se poate observa că algoritmii genetici utilizaţi cu atenţie permit o rezolvare uşoară şi rapidă a unor probleme şi dau opţiunea de a le ajusta timpul şi memoria folosită pentru a lucra în diferite situaţii.

Anexe

A.1 Funcţii MatLab utilizate

- **rand** Apelul aceste funcţii întoarce un număr aleator din intervalul [0,1];
- **randi**

  Această funcţie are mai multe apeluri cu diferite rezultate pe parcursul lucrării:

  - **randi(imax)** întoarce o valoare întreagă între 1 şi imax
  - **randi([imin, imax])**întoarce o valoare întreagă între imin şi imax
  - **randi([imin, imax], n, m)** întoarce o matrice de dimensiune n*m cu valori întregi aleatorii între imin şi imax
- **zeros(n,m)** întoarce o matrice n*m de zerouri
- **[A,B]=sort(C,ordine)** Transmite în variabila A vectorul C ordonat, iar, în variabila B indici ordonaţi ai valorilor din vectorul C. Dacă C_i este A_1, atunci B_1 este i.

Pentru plotarea graficelor folosite în capitolul 2 au fost folosite funţiile plot, plot3d şi surf.

Bibliografie

[1] Sean Luke, *Essentials of Metaheuristics*, Lulu, second edition, 2013
http://cs.gmu.edu/~sean/book/metaheuristics/Essentials.pdf

[2] M. Breabăn, H. Luchian , *Algoritmi genetici*,
http://students.info.uaic.ro/~vladut.ungureanu/Algoritmi-genetici-ID.pdf

[3] D. E. Goldberg *Genetic Algorithms in Search, Optimization and Machine Learning*,Addison Wesley Publishing Company, 1989: 1-82.
http://www2.fiit.stuba.sk/~kvasnicka/Free%20books/Goldberg_Genetic_Algorithms_in_Search.pdf,

[4] D. Hermawanto, *Genetic Algorithm for Solving Simple Mathematical Equality Problem*, (LIPI)https://arxiv.org/ftp/arxiv/papers/1308/1308.4675.pdf

[5] *Algoritmi Genetici. Studiu de caz: Optimizarea traficului intr-o retea*:
http://www.scritub.com/stiinta/informatica/retele/Algoritmi-Genetici-Studiu-de-c24232.php.

[6] Brian R. HUNT, Ronald L. LIPSMAN şi Jonathan M. ROSENBERG *A Guide to MATLAB for Beginners and Experienced Users*, Cambridge University Pres, 2001
https://kkpatel7.files.wordpress.com/2014/10/a-guide-to-matlab.pdf