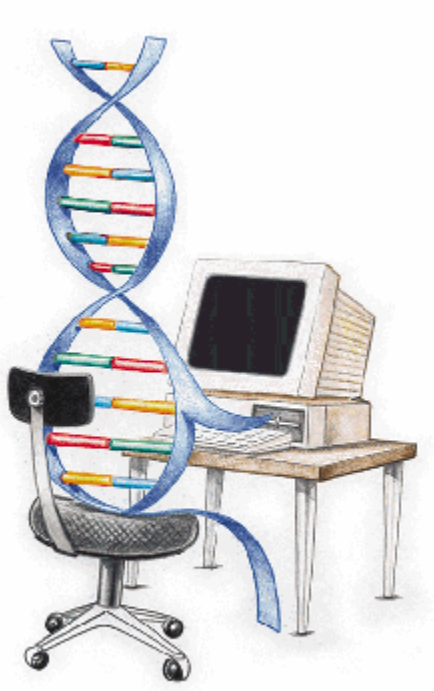
Genetikus algoritmusok megvalósítása MATLAB segítségével



Werner Ágnes

A Matlab genetikus algoritmusokat használó eszköztára

Kétféle módon használhatjuk fel az eszköztár lehetőségeit:

1. **Parancssorból**
2. **Genetic Algorithm Tool segítségével.**

Parancssori lehetőségek

Fő függvénye a ga függvény, amely F(X) minimumát próbálja meg meghatározni, megadott feltételeket figyelembe véve.

F a fitnesz függvény, X egy tetszőleges egyed

Ekkor az optimális megoldás egy olyan X, ahol

1. $A_{eq}*X=b_{eq}$ **(lineáris egyenletek)**
2. $A*X \leq b$ **(lineáris egyenlőtlenségek)**
3. $C_{eq}(X)=0$ **(nemlineáris egyenletek)**
4. $C(X) \leq 0$ **(nemlineáris egyenlőtlenségek)**
5. $LB \leq X \leq UB$ **, azaz X egy adott intervallumban keresendő**

Feltételek teljesülése mellett F(X) értéke minimális.

Ezen jelölések mellett a ga függvény általános alakja:

```
>>[X, FVAL, REASON, OUTPUT, POPULATION,
  SCORES]=GA(F, NVARS, A, b, Aeq, beq, LB, UB, NONLCON,
  Options)
```

A fent nem definiált jelölések jelentése:

- **NVARS: F függvény függvényváltozóinak száma**
- **NONCLON: C(X) és Ceq(X) függvényeket megvalósító Matlab függvény (ezeket ált. magunknak kell implementálnunk)**
- **FVAL: F(X), a kimenő megoldásegyedre**
- **REASON: a kilépés okának leírása**
- **OUTPUT: a futás körülményeiről ad néhány információt ez a struktúra**
- **POPULATION: a kilépéskor meglévő populáció**
- **SCORES: a kilépéskor meglévő populáció fitnessz értékei**
- **options: az algoritmus paramétereit tartalmazó struktúra**

Egyszerű példa:

A sikeres futtatáshoz elegendő F és NVARS megadása is:

>>ga(@(x) x*x, 1)

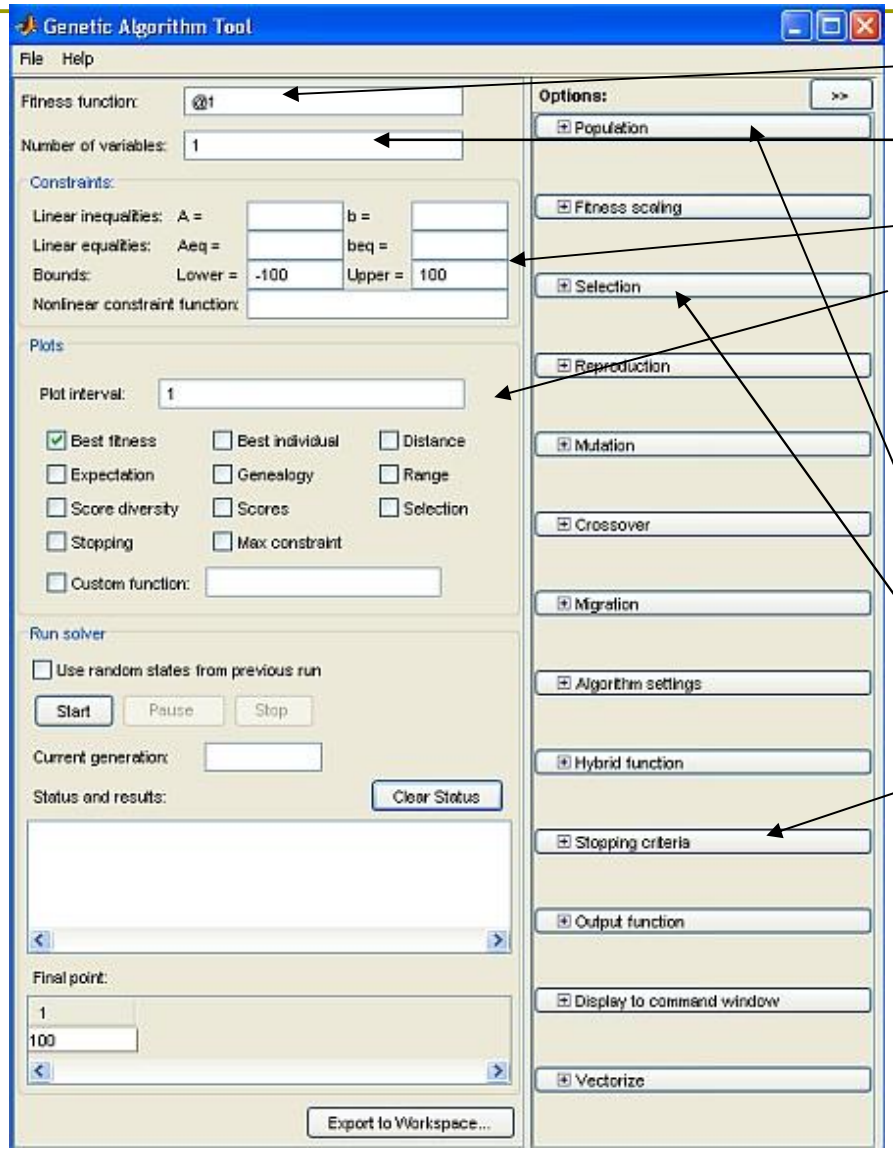
Ez az $f(x)=x^2$ függvény abszolút minimumát keresi, ami x=0 helyen található.

```
>> ga(@(x) x*x, 1)
Optimization terminated: maximum number of generations exceeded.

ans =

 -4.7080e-005
```

A futás egészen közeli eredményt talált.

A leállás a generációszám maximális értékének túllépése miatt történt.

Genetic Algorithm Tool



Fitnessz függvény megadása

Függvényváltozók száma

Feltételek

Megjelenítő funkciók (Plots):
Ezen beállítások szabályozásával az algoritmus egyes lépéseinek főbb paramétereiről kaphatunk grafikusan ábrázolt információt, már futás közben.

Populáció Tulajdonságai

Szelekciós Tulajdonságok

Kilépési feltételek megadása

Bonyolultabb példa

Matlabék (és talán mások) kedvence a **Rastrigin-függvény**, ha a genetikus algoritmusokról van szó. Ez egy n-változós valós függvény, amely *n* függvényében a következő formulával adható meg:

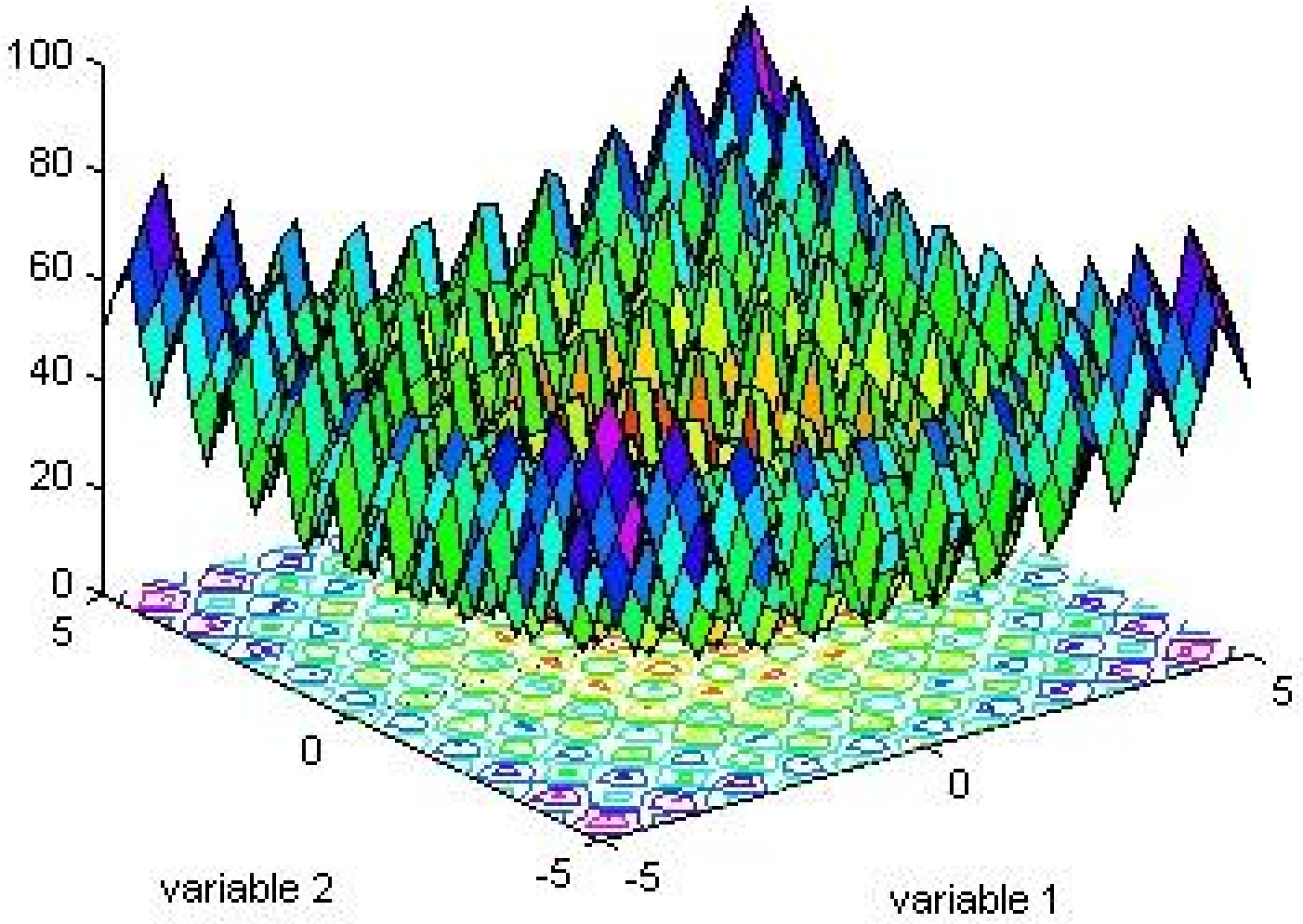
$$f(x) = 10 \cdot n + \sum_{i=1}^{n} (x_i^2 - 10 \cdot \cos(2 \cdot p \cdot x_i))$$

Ez a függvény folytonos, differenciálható, és könnyen észrevehető, hogy minimuma $\underline{x} = 0$ pontban van, értéke $f(x) = 0$.

Ezen kívül hála a cos függvény periodicitásának, a függvény tele van lokális minimumhelyekkel, ami igen rossz hatással lehet a kereső algoritmusokra.

A függvény megtalálható **rastriginsfcn** néven a Matlab tárházában.

A Rastrigin-függvény képe (n=2, -5 ≤ x1,x2 ≤ 5).



Használjuk a genetikus algoritmust

```
>> x  =  ga(@rastriginsfcn,  2)
Optimization terminated: maximum number of generations exceeded.

x =

    0.0009   -0.0039
```

Az eredmény elfogadható első neki futásra.
Érdemes megnézni egy konkurens kereső algoritmust. Vegyük a Matlab fminsearch függvényét, nézzük mit tud kezdeni vele:

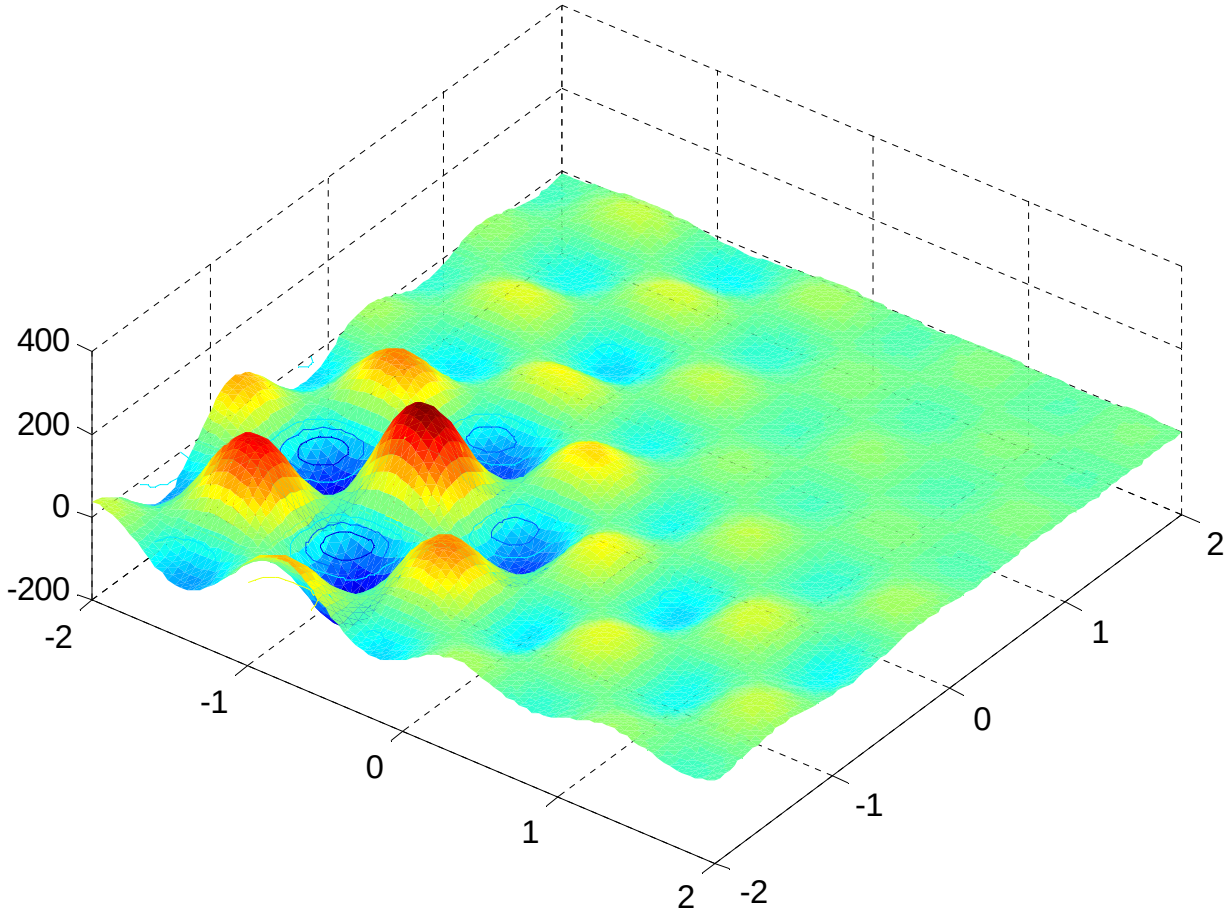
```
>> fminsearch(@rastriginsfcn,  rand(1,2))

ans =

    0.9950    0.9950
```

A keresés beleesett egy lokális minimum körüli gödörbe.

Egy kétváltozós valós fg. - Shufcn fg.

```
function f = shufcn(y)
%SHU objective function.
%
%   L.C.W. Dixon and G.P. Szegö (eds.), Towards Global Optimisation 2,
%   North-Holland, Amsterdam 1978.

%   Copyright 2004 The MathWorks, Inc.
%   $Revision: 1.1 $  $Date: 2004/01/14 15:35:06 $

for j = 1: size(y,1)
   f(j) = 0.0;
   x = y(j,:);
   temp1 = 0;
   temp2 = 0;
   x1 = x(1);
   x2 = x(2);
   for i = 1:5
      temp1 = temp1 + i.*cos((i+1).*x1+i);
      temp2 = temp2 + i.*cos((i+1).*x2+i);
   end
   f(j) = temp1.*temp2;
end
```

```
>> plotobjective(@shufcn,[-2 2;-2 2]);
>>
```



Eredmények

```
>> FitnessFunction=@shufcn;
>> numerOfVariables=2;
>> [x,Fval,exitFlag,Output]=ga(FitnessFunction,numerOfVariables);
Optimization terminated: stall generations limit exceeded.
```

```
>>  fprintf('The number of generations was: %d\n', Output.generations);
The number of generations was: 82
```

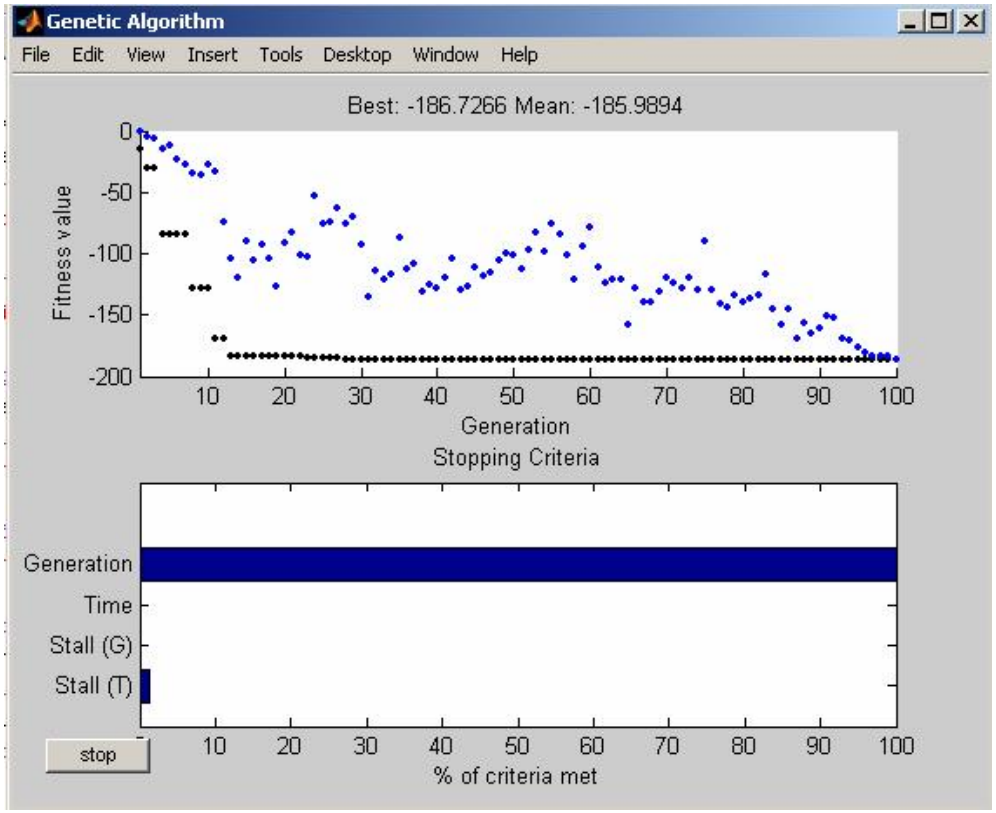
```
>> fprintf('The number of function evaluations was : %d\n', Output.funccount);
The number of function evaluations was : 1640
```

```
>> fprintf('The best function value found was: %d\n', Fval);
The best function value found was: -1.854801e+002
```

Paraméterek módosítási lehetőségei

```
>> opts = gaoptimset('PlotFcns',{@gaplotbestf,@gaplotstopping});
```

```
>> [x,Fval,exitFlag,Output] = ga(FitnessFunction,numberOfVariables,opts);
Optimization terminated: maximum number of generations exceeded.
```



Költség és hatékonyságelemzés

- Nehéz egy olyan módszer hatékonyságát elemezni, ami nemdeterminisztikus lépéseket használ működése közben.
- Egy adott GA implementáció futása a véletlen elemek miatt mindig változik, nincs két egyforma kimenetű futás (hacsak nem nagyon triviális az algoritmus, vagy nagyon rossz a véletlenszám-generátor adott megvalósításnál).
- Így nem sok értelme lenne összehasonlító futási eredményeket nézegetni, főleg ilyen példák esetében ahol az abszolút futási idő legdurvább esetben is csak néhány másodperc.

Nézzük meg milyen tényezők befolyásolják a GA költségét!

Populáció mérete

A populáció mérete nyilvánvalóan alapvetően befolyásolja mind a futás, mind a tárhely igényét.

Generációk száma

A generációk számával egyenesen arányos a futási idő, mivel minden új generáció egy új főciklus lefuttatását igényli. Mivel a kilépési feltétel nemcsak a maximális generációszámtól függ, előbb is terminálhat az algoritmus, nem feltétlenül igaz, hogy egy *10x* nagyobb generációszámmal paraméterezett algoritmus *10x* annyi ideig fut.

Szelekció

A szelekció GA esetében a populáció méretének függvénye, mivel pont ennyi szülőt kell kiválasztanunk (elitista módszer esetén ennek egy részét). Ez a megvalósításoktól függően általában lineáris költségű, de bizonyos esetekben (pl. pár-verseny szelekció) lehet nagyobb is.

Rekombináció, mutáció

Hasonlóan a szelekcióhoz, ez is a populáció méretének a függvénye. Viszont itt már bizonyos esetekben a futási költségbe beleszólhat az egyedek reprezentációja. Ugyanis több esetben a gének számával arányos az egyedek keresztezése ill. mutációja. Természetesen több gén esetén tovább tart ezek elvégzése.

Visszahelyezés

A visszahelyezés művelete $\sigma(1)$ általában, mivel egyszerűen csak a régi populációt megfeleltetjük az újonnan kialakítottal. Olyan esetekben viszont, amikor az algoritmus nem generációs, lehetséges nagyobb költségű megvalósítás is.

Fitnesz kiértékelés

A fitnesz kiértékelés nagyban függ a kiértékelő függvénytől. Vizsgált eseteinkben ez $\sigma(1)$, de könnyen elképzelhető igen bonyolult fitnesz függvény is.

Nézzünk meg egy nagyon egyszerű összehasonlítást

Globális minimum

keresése az első példánkban vett $f(x) = x^2$ függvénynél.

Versenyeztessük meg a ga és az fminsearch algoritmusokat!

A feladat komplexitása miatt azt várjuk, hogy az utóbbi lesz a nyertes.

Lefuttatva alapbeállításokkal, 100 futás átlagára azt kapjuk, hogy kb. 12-13-szor gyorsabb, mint a GA megvalósítás.

Ráadásul az eredmény átlag 15-16 jegyre pontos, míg a genetikus metódus 4-7 jegyig egyező eredményeket tud csak produkálni.

Javítva a GA paraméterezésén, némi kísérletezés után ezt sikerül annyira lefaragni, hogy azonos pontosság mellett mindössze 5-ször annyi időbe telik lefuttatni a ga parancsot, mint társát.

Mit mutat ez meg?

Nem többet és nem is kevesebbet, mint amire számítottunk.

Látszik, hogy ilyen típusú feladatnál érdemesebb a hagyományos kereső technikákat alkalmazni, főleg hogy ezeknek igen jó megvalósítása adva van Matlab környezetben. A probléma túl egyszerű, túl költséges egy párhuzamos számításokat alkalmazó eljárás használatához.

Érdemes viszont azt megfigyelni, hogy a paraméterek helyes beállítása milyen nagymértékben tudja befolyásolni a futás költségét és kimenetelét.

További következtetések

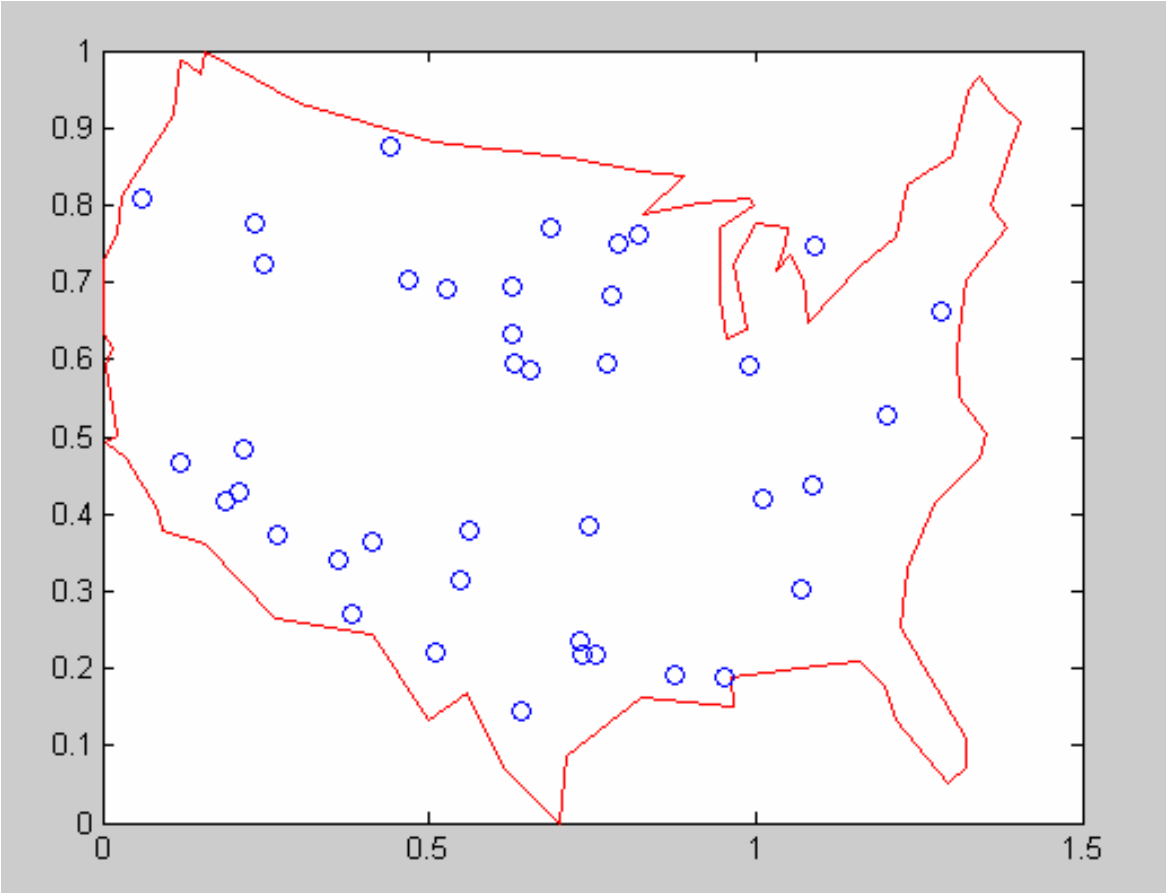
További futási idő teszteken azt kapjuk, hogy Rastrigin-függvény esetében megmarad az fminsearch 10-15-szörös sebességkülönbsége, alapbeállítások mellett.

Cserébe nem ad helyes eredményt egyikre sem…

Ebből olyan következtetést lehet levonni, hogy GA alkalmazása olyan esetekben célszerű ahol jobb eredményt várunk tőle, mint a hagyományos kereső algoritmusoktól, vagy ahol azok nem használhatóak.

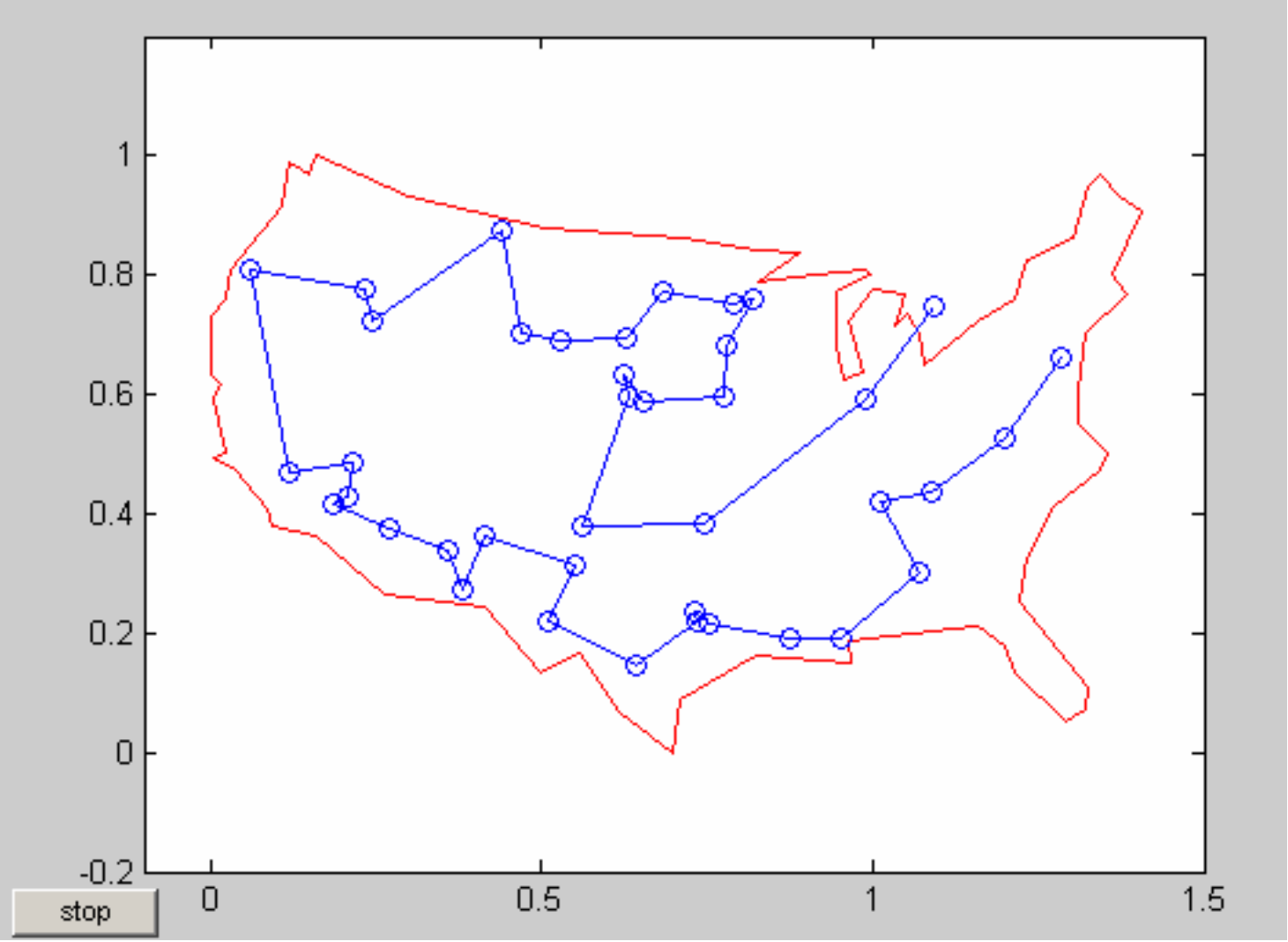
Bizonyos esetekben jó megoldás lehet az algoritmusok keverése: egy alacsonyabb költségű kereső eljárás segítségével információt szerzünk a problémáról, amit felhasználhatunk a későbbi GA paraméterezésénél.

Utazó ügynök probléma




```
>> load('usborder.mat','x','y','xx','yy');
>> plot(x,y,'Color','red'); hold on;
```

Megoldás GA-val felhasználva a Matlab-ot



Készítsük el a „Hello World!” genetikus algoritmusát!

Megpróbáljuk genetikusan „kitenyészteni” sztringek egy sorozatából ezt az üdvözlést.

A populáció egyedei azonos hosszúságú sztringek, amiket genotípusosan ábrázolunk. Minden sztringhez hozzárendelünk egy sorvektort, amelynek elemei (azaz a gének) a sztring egyes karaktereinek ASCII kódjai.

A fitnesz érték az egyed és a keresett sztring távolsága, azaz az egyes betűk közötti távolságok összege.

Minél kisebb a fitnesz érték, annál rátermettebb az adott egyed. Nyilvánvaló, hogy ha ez az érték nulla, akkor elértük a keresett sztringet.

Algoritmus

- **Az algoritmus fő ciklusában sorba rendezzük a populáció egyedeit növekvő fitnesz érték alapján.**
- **A legjobbakat automatikusan beválogatjuk a következő populációba (elitizmus).**
- **A maradék helyek feltöltéséhez szelekció során kiválogatjuk a szülőket. A szelekció ebben az esetben egyszerű véletlen kiválasztás, ami nem foglalkozik az önreprodukcióval (mindkét szülő ugyanaz az egyed).**
- **A kiválasztott szülőket egypontos keresztezéssel rekombináljuk, a keletkező egyedet behelyezzük az új populációba.**
- **Legvégül egy előre beállított mutációs ráta szerint módosítjuk a sztringeket. A mutáció megvalósítása véletlen génmutáció, egyetlen génre.**

Algoritmus

- A kapott új populációra kiértékeljük a rátermettségi függvényt, az egyedeket behelyettesítjük a régi populáció helyére, növeljük a generációszámot és ezzel le is zárul a fő ciklus.
- Kérdés még, hogy milyen kilépési feltételt lehet alkalmazni? Két dolgot veszünk figyelembe: egy előre megadott generációszám ill. az optimális megoldás elérését (fitnesz érték nulla).
- Az alapbeállítás 1000 példányos populációkkal dolgozik.
- A populáció legjobb 10%-át válogatjuk az elitek közé minden lépésben.
- A mutáció esélye 0,25 egyedenként. Ha elérjük a
- 100 generációt optimális eredmény nélkül, a függvény leáll, és az adott pillanatban legjobb egyedet adja vissza közelítő megoldásként.

A függvény definiálása, alapértékek beállítása

```
function y = helloga(str)
% HELLOGA(STR)
%
% STR-ben megadott sztring kitenyésztése genetikus algoritmussal.
% Paraméter nélkül hívva a 'Hello World!' üzenetet próbálja elérni.
%
%Stratégiai paraméterek beállítása
GA_TARGET = 'Hello World!';    %default célstring
GA_POPSIZE = 1000;             %populáció egyedszáma
GA_MAXITER = 100;              %maximális iterációszám
GA_ELITRATE = 0.1;             %elitráta
GA_MUTATION = 0.25;            %mutációs ráta
PRINTOUT = 10;                 %eredmény kiíratás periódusa
 if nargin>0
GA_TARGET=str;
end
```

Kezdeti populáció

```
% t := 0, populációk számának inicializálása
t = 0;
% P kezdeti populáció létrehozása
P = floor (rand(GA_POPSIZE, length(GA_TARGET)) * 96 + 32);
F = sum(abs(P-ones(GA_POPSIZE,1)*GA_TARGET), 2);
% WHILE NOT Kilépési_feltétel(P)
while t < GA_MAXITER & F(1) ~= 0          % kilépési feltétel: iterációk
                                          száma vagy megoldás
% Aktuális populáció sorbarendezése
[F, I] = sort(F);
for i = 1 : length(I)
TMP(i, :) = P(I(i), :);
end
P = TMP;
y = char( P(1, :));          % legjobb egyed
% Az algoritmus futásáról tájékoztatjuk a felhasználót
if rem (t, PRINTOUT) == 0
str = sprintf('%d. generáció legjobb egyede: %s      fitness: %d', t, y, F(1));
disp (str);
end
```

Elitizmus, szelekció, rekombináció, mutáció

```
% Elitek beválogatása
elitek = floor (GA_ELITRATE * GA_POPSIZE); Puj(1:elitek,:) =
    P(1:elitek,:);
% Szelekció (a fennmaradó helyekre)
for i = elitek+1 : GA_POPSIZE
e1 = floor( rand * GA_POPSIZE + 1);   % anya
e2 = floor( rand * GA_POPSIZE + 1);   % apa
crp = floor( rand * length(GA_TARGET) + 1);   % keresztezési pont
% Rekombináció
Puj(i, :) = [P(e1, 1:crp), P(e2, (crp+1):length(GA_TARGET))];
end
% Mutáció (várható érték alapján)
for i = 1 : GA_POPSIZE*GA_MUTATION
Puj (floor(rand*GA_POPSIZE)+1, floor(rand*length(GA_TARGET))+1) =
    floor(rand * 96 + 32);
end
% Visszahelyezés
P = Puj;
```

Kiértékelés, eredmény kiíratása

```
% P egyedeinek kiértékelése az F fitnesz függvény alapján
F = sum(abs(P-ones(GA_POPSIZE,1)*GA_TARGET), 2);
% Populációszám (iterációszám) növelése
t = t + 1;
end
% WHILE ciklus vége
% Legjobb fitnesz értékkel rendelkező egyed kiválasztása
[F, I] = sort(F);
y = char(P(I(1), :));
if F(1) == 0
disp('Optimális érték elérve.');
else
disp('A program elérte a maximális generációszámot optimum
    nélkül.');
end
% end of helloga
```

Első próbálkozás

```
>> helloga
0. generáció legjobb egyede: Kc<|o#^U_yn!   fitness: 147
10. generáció legjobb egyede: Mfsde/Pokgj-  fitness: 83
20. generáció legjobb egyede: Kmlpg!Opqrc!  fitness: 41
30. generáció legjobb egyede: Kflsp!Opqrc!  fitness: 30
40. generáció legjobb egyede: Eelpp ]oqid!  fitness: 18
50. generáció legjobb egyede: Hclnn!Yoqkc!  fitness: 11
60. generáció legjobb egyede: Helmn!Vorkc!  fitness: 6
70. generáció legjobb egyede: Helln Voqkc!  fitness: 5
80. generáció legjobb egyede: Helln Xorkd!  fitness: 3
90. generáció legjobb egyede: Helln Workd!  fitness: 2
A program elérte a maximális generációszámot optimum nélkül.

ans =

Helln Workd!
```

Második próbálkozás

```
>> helloga
0. generáció legjobb egyede: AXgjf3>xukx,   fitness: 124
10. generáció legjobb egyede: AXgjd Zkmu]   fitness: 67
20. generáció legjobb egyede: Hfllk Zkmuc+  fitness: 37
30. generáció legjobb egyede: Hflln Xnsh`'  fitness: 19
40. generáció legjobb egyede: Hflin Xnskc   fitness: 11
50. generáció legjobb egyede: Hflln Wnskc   fitness: 7
60. generáció legjobb egyede: Hflln Xnsld   fitness: 6
70. generáció legjobb egyede: Hfllo Wnskc   fitness: 6
80. generáció legjobb egyede: Hfllo Wnskc!  fitness: 5
90. generáció legjobb egyede: Hfllo Wnrlc!  fitness: 3
A program elérte a maximális generációszámot optimum nélkül.

ans =

Hello Worlc!
```