Bevezetés a lágy számítás módszereibe

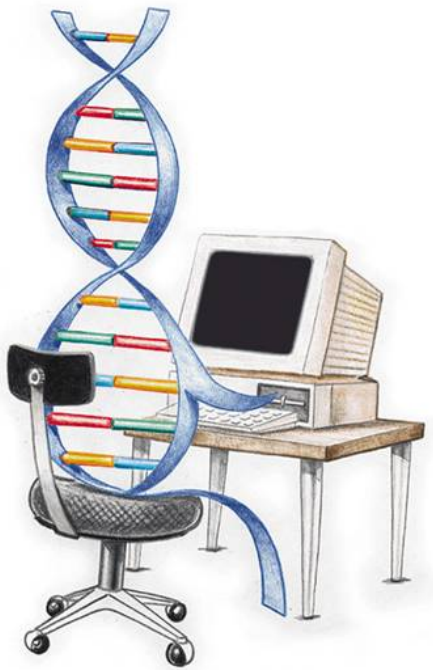
Werner Ágnes

Pannon Egyetem
Villamosmérnöki és Információs Rendszerek Tanszék

Genetikus algoritmusok - Alkalmazási területek

Genetikus algoritmusok megvalósítása MATLAB segítségével

1. Parancssorból
2. Genetic Algorithm Toolbox segítségével



Parancssori lehetőségek 1.

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.

Parancssori lehetőségek 2.

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 $C_{eq}(X)$ 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 fitnesz értékei
- **options**: az algoritmus paramétereit tartalmazó struktúra

Egyszerű példa

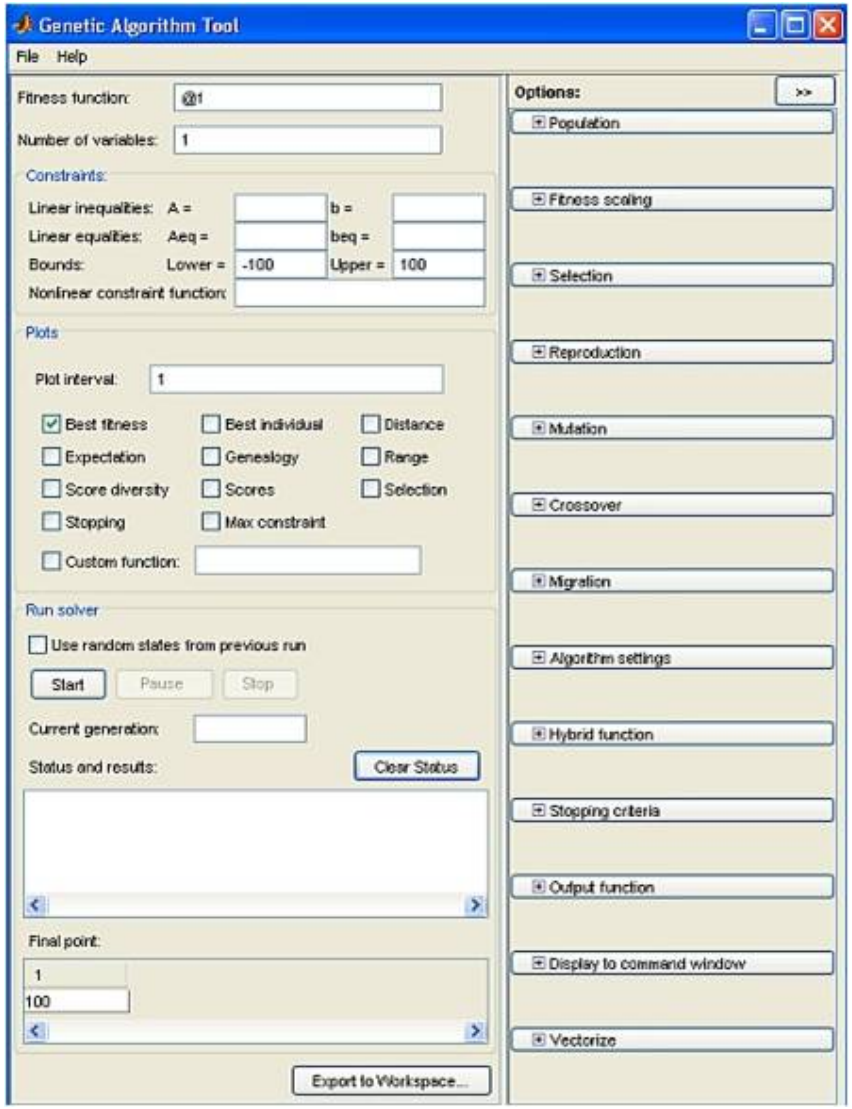
A sikeres futtatáshoz elegendő F és $NVARS$ megadása is:
$>> ga(@(x)x * x, 1)$
Ez az $f(x) = x^2$ függvény abszolút minimumát keresi, ami $x = 0$ helyen található.

```
>> 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

Indítás: gatool

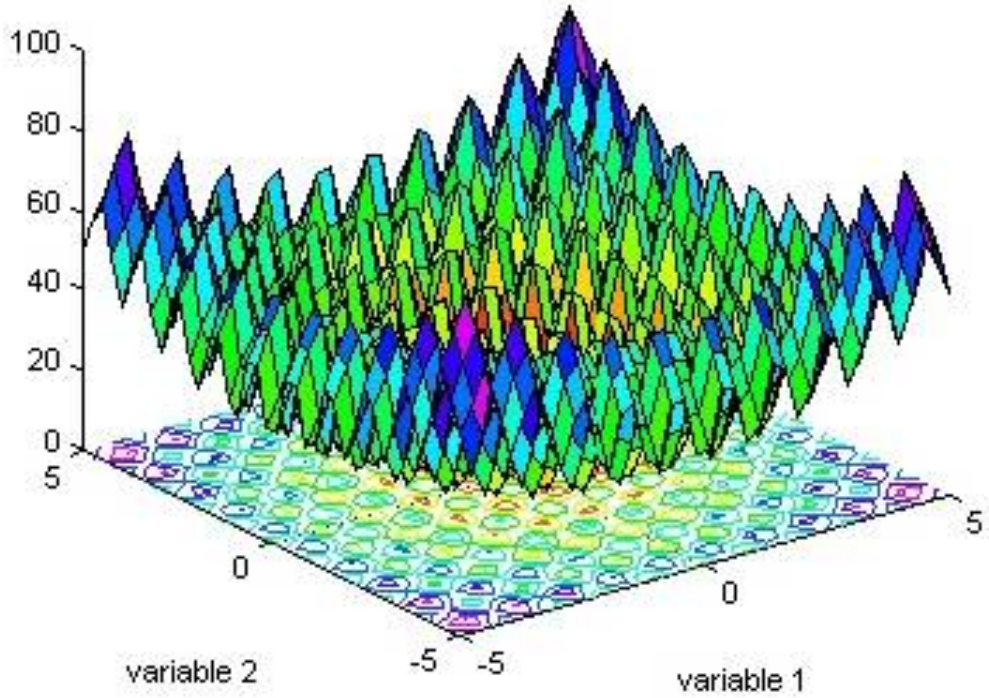


Bonyolultabb példa

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:
$f(x) = 10n + \sum_{i=1}^{n}(x_i^2 - 10\cos(2\pi x_i))$
Ez a függvény folytonos, differenciálható, és könnyen észrevehető, hogy minimuma $x = 0$ 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 ($n = 2, -5 \leq x1, x2 \leq 5$)

plotobjective(@rastriginsfcn,[-5 5;-5 5]);



Az eredmény elfogadható első neki futásra.

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

x =

    0.0009   -0.0039
```

Mondatfelismerő genetikus algoritmus

- Megpróbálunk genetikusan kitenyészteni sztringek egy sorozatából egy mondatot.
- 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 jellemzői

- 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 keresztezés**sel rekombináljuk, a keletkező egyedet behelyezzük az új populációba.
- Legvégül egy előre beállított **mutáció**s ráta szerint módosítjuk a sztringeket. A mutáció megvalósítása véletlen génmutáció, egyetlen génre.

Algoritmus jellemzői

- 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 feltétel**t lehet alkalmazni? Két dolgot veszünk figyelembe: egy előre megadott generációszám ill. az optimális megoldás elérését (fitnesz érték nulla).
- Az alapbeállítás 1000 **példány**os populációkkal dolgozik.
- A populáció legjobb 10%-át válogatjuk az **elitek** közé minden lépésben.
- A mutáció esélye $0,25$ egyedenként. Ha elérjük a 1000 generációt optimális eredmény nélkül, a függvény leáll, és az adott pillanatban legjobb egyedet adja vissza közelítő megoldásként.

Algoritmus

```
function y = mondat(str)
% MONDAT(STR)
%       STR-ben megadott sztring kitenyésztése genetikus algoritmussal.
%       Paraméter nélkül hívva a 'Agent technology is a new field in software
%       development.' mondatot próbálja meg elérni.
%Stratégiai paraméterek beállítása
GA_TARGET = 'Agent technology is a new field in software development.';
GA_POPSIZE = 1000; %populáció egyedszáma
GA_MAXITER = 1000; %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
```

Algoritmus

```
% 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 sorba rendezése
[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
% Elitek beválogatása
elitek = floor (GA_ELITRATE * GA_POPSIZE); Puj(1:elitek,:) = P(1:elitek,:);
```

Algoritmus

```
% 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;
% P egyedeinek kiértékelése az F fitnessz függvény alapján
F = sum(abs(P-ones(GA_POPSIZE,1)*GA_TARGET), 2);
% Populációszám (iterációszám) növelése
t = t + 1;
end
```

Eredmény1

```
>> mondat
0. generáció legjobb egyede: 4-gOJTqcvwtvVXg*)[SBOXJ_X(mXYw"/OQlTo>4t<rNI] )ndl$`]\^9   fitness: 1398
10. generáció legjobb egyede: [SsLB!|U7M>oYc/@9uY6U_hKi"b/iz\"iv((hUzgcLSBwonY\L|n4_wP   fitness: 1106
20. generáció legjobb egyede: @rRhwC(wd^lcpt□h$TJ/c+)Ki"b/iz\"iv((hUzgcLSBwonY\Ygkgoa=   fitness: 713
30. generáció legjobb egyede: Bkbhw$|gZf[pdt□□$TJ/c+)Oi"a/iz\"iv((hUzgcLN%fonY\z|n4rwP   fitness: 643
40. generáció legjobb egyede: Bkbhw$|gZSbqeqoe1Up/l?hKi"bQ]z\"iv((hdmRop^%fonY\Ygkgoa4   fitness: 531
50. generáció legjobb egyede: Bkdhw$|bhppieqoe1Un&t!hKi"bm]z\"iv(vhUzRop^%fUfidmgkgoo=   fitness: 440
60. generáció legjobb egyede: Bkbhw$|bhppieqoy$Pn&t!hcx-bZ]z\"iv(vhUzgip^%fonid^|ngoo=   fitness: 375
70. generáció legjobb egyede: Bkdhw$|gZppifriy(uj&b!hcq"dZiz\"iv((hdmgcp^&fonYdmgkgoo=   fitness: 312
80. generáció legjobb egyede: @rbhw$tgZppifriy(uj&b!hcq"dZif\"iv((hgugcp^%fow^dmgogoo=   fitness: 285
90. generáció legjobb egyede: @kbow$|bhfmqeqoy(un&b!hdi"bmiz\"il )h_zncp^%fonYhlgkgos0   fitness: 259
100. generáció legjobb egyede: Bkdhw$|gafmqfqoy$an&b!hci"bc^n\"iv (hdmgcp^%fowYhlgkgos0   fitness: 243
110. generáció legjobb egyede: Bkbhw$sgZfmqrqiy$uj&b!hd|"bmij\"iv vodmgcp^&fonahlgkgos0   fitness: 220
120. generáció legjobb egyede: Bkbhw$tgafmqeqoy$jn&b!hd|"bmij\"iv (h_zncp^%fmnihqtkgos0   fitness: 204
130. generáció legjobb egyede: Bkbow$sbafmqrriy$jn&b!hd|"bmij\"il (hgzncp^%fmnihmpogow4   fitness: 185
140. generáció legjobb egyede: Bkdhw$tgafmqrqby$jn&b!id|"bmij\"iv vodsncp^%ffnihmpogos0   fitness: 164
150. generáció legjobb egyede: Bkdnw$tgffmqrqiy$jn&b!hd|"bmin\"iq vodsncp^%ffnihmplgos0   fitness: 151
160. generáció legjobb egyede: Bkdnw$tgafmqrney$jn&b!idv"bmin\"iq vodsncp^"ffxihmpkgrs0   fitness: 139
170. generáció legjobb egyede: Bkdnw$sbafmqrney$jn&b!md|"bmij\"iq vodsncpf%ffvihmpkgos0   fitness: 133
180. generáció legjobb egyede: Bkdnw$tgafmqrqiy$jn&b!ifv!bmijc"iq vodswcp^"ffxihmpogos0   fitness: 120
190. generáció legjobb egyede: Bkdnw$tbafmqrney$jn&b!idv"bmijc"iq vogswcpf ffvihmpkgos0   fitness: 110
200. generáció legjobb egyede: Bkdnw$tbafmqlniy jn&b!idv"bmijc"iq vodswcpf ffvilmpogos0   fitness: 97
210. generáció legjobb egyede: Bkdnw!tgafmqlniy jn&b!md|"bmijc"iq vodswcpf ffvilmpogos0   fitness: 93
220. generáció legjobb egyede: @kdnw!tgafmqlney jn&b!mdv"bmijc"iq vodswcpf ffvilmpogos0   fitness: 89
```

Eredmény2

```
540. generáció legjobb egyede: Agdnt tecinolngy is b mew field in sogtware development. fitness: 6
550. generáció legjobb egyede: Agent tecinolngy is b mew field in sogtware development. fitness: 5
560. generáció legjobb egyede: Agent tecinolngy is a mew field in software development. fitness: 3
570. generáció legjobb egyede: Agent tecinolngy is a mew field in software development. fitness: 3
580. generáció legjobb egyede: Agent tecinolngy is a mew field in software development. fitness: 3
590. generáció legjobb egyede: Agent tecinolngy is a mew field in software development. fitness: 3
600. generáció legjobb egyede: Agent tecinolpgy is a new field in software development. fitness: 2
610. generáció legjobb egyede: Agent technolngy is a mew field in software development. fitness: 2
620. generáció legjobb egyede: Agent technolngy is a new field in software development. fitness: 1
630. generáció legjobb egyede: Agent technolpgy is a new field in software development. fitness: 1
640. generáció legjobb egyede: Agent technolngy is a new field in software development. fitness: 1
650. generáció legjobb egyede: Agent technolpgy is a new field in software development. fitness: 1
Optimális érték elérve.

ans =

Agent technology is a new field in software development.

>>
```

Órarend készítési probléma

Sok feltételt kell figyelembe venni:

- Minden tanárnak fix számú órája van bizonyos osztályokban.
- Nincs olyan osztály és tanár, akinek egyszerre két órája van.
- Minden tanárnak egy héten lehetőleg legyen egy szabad napja.
- Az osztályoknak csak a nap elején vagy végén lehet lyukas órájuk.
- Ha egy tanárnak több órája van egy nap egy osztállyal, akkor lehetőleg azok egymás után legyenek.
- Egy tanárnak lehetőleg ne legyen sok lyukas órája.
- stb.

Hagyományos órarend reprezentáció

A kétdimenziós mátrix (i, j) eleme azokat a tanárokat tartalmazza, akik a j-edik órában az i-edik osztályban tartanak órát.

	1. nap				…	n. nap			…
	1. óra	2. óra	…	*n*. óra	…	…	*j*. óra	…	…
1. oszt	Tóth L.	Biró I.							
2. oszt	Szabó L.	Kovács G.							
.									
.									
.									
i. oszt							Kiss I.		
.									
.									
.									

Ebben könnyű megállapítani, hogy egy tanár elfoglalt-e egy adott időben vagy ki tart órát egy bizonyos időpontban és osztályban.

Nem támogatja annak ábrázolását, amikor több osztálynak egyidejűleg több tanár tart órát. Például bontott nyelvóráknál 2 osztályt 4 tanár is taníthat.

Halmazos reprezentáció

A legkisebb **adategység a halmaz**, egy olyan struktúra, amely tetszőleges számú osztályt, tanárt és tantermet tartalmaz.
Pl. két osztályhoz egy tanárt rendelünk (például összevont testnevelés óra esetén) a két osztályt és az egy tanárt felvesszük a halmazba.
Csak **egy dimenzió**ban dolgozunk, amely nem más mint az idő.
Az időtengelyen lévő időrésekbe, melyek a lehetséges órákat jelentik-kell halmazainkat elhelyezni.
Egy időrésbe több halmaz is kerülhet, itt ügyelni kell arra, hogy ne legyen ütközés, ne kerüljön egy időrésbe két olyan halmaz, melyben közös tanár vagy közös osztály szerepel.

Halmazos reprezentáció

Az $o_{i,j}$, $t_{i,j}$, $te_{i,j}$ értékekkel az i-edik halmazban szereplő osztályok, tanárok, termek sorszámát jelöljük, j pedig a halmazon belüli sorszámok indexe.

1. nap			2. nap			...
1. óra	2. óra	...	...	...	...	...
Halmaz_1: $osztály_{0_{1,1}}, osztály_{0_{1,2}}, \ldots$ $\tan ár_{t_{1,1}}, \tan ár_{t_{1,2}}, \ldots$ $terem_{te_{1,1}}, terem_{te_{1,2}}, \ldots$	Halmaz_{N+1}: $osztály_{0_{N+1,1}}, osztály_{0_{N+1,2}}, \ldots$ $\tan ár_{t_{N+1,1}}, \tan ár_{t_{N+1,2}}, \ldots$ $terem_{te_{N+,1}}, terem_{te_{N+1,2}}, \ldots$					
Halmaz_2	Halmaz_{N+2}					
...	...	...				
Halmaz_N	Halmaz_{2N}					

Hagyományos órarend reprezentáció

Függőleges és vízszintes linearizálás

1. időrés	2. időrés	...	k. időrés
Halmaz 1	Halmaz N+1		Halmaz (k-1)*N+1
Halmaz 2	Halmaz N+2		Halmaz (k-1*N+2
...	...		
Halmaz N	Halmaz 2N		Halmaz k*N

Függőleges:

Halmaz 1	Halmaz 2	...	Halmaz N	Halmaz N+1	...	Halmaz k*N

Vízszintes:

Halmaz 1	Halmaz N+1	...	Halmaz (k-1)*N+1	Halmaz 2	...	Halmaz k*N

Egy órarend jóságának elemzése

Kemény kötések

- Tanárütközésről akkor beszélünk, amikor egy tanárnak egyidejűleg több osztály számára kéne órát tartania.
- Teremhiány jelentkezhet akkor, ha egy osztály számára nem jut tanterem, ahol az órát megtudnák tartani. (korlátozott erőforrás)
- Tanárok kemény ráérése. A tanároknak iskolán kívüli feladatai is lehetnek, melyeket kötelező ellátni. Így például előfordulhat, hogy egy tanárnak nem lehet órája a pénteki napon.

Lágy kötések

- Lyukasóra. Általános iskolában nem megengedhető, középiskolában is nehezen tolerálható, ha valamelyik osztály órarendjében lyukasóra van.
- Nulladik óra. A tanulók számára megterhelő a nulladik vagy esetleg a 7. 8. óra.
- Többszörös óra. Egy osztálynak, ne legyen egy tárgyból egy nap több órája. Például heti két fizika óra esetén ne essen ez a két óra egy napra.
- Héten belüli egyenletes óraeloszlás. A napi óraszámok között lehetőleg kis eltérés legyen az egyenletes terhelés érdekében.
- Tanárok lágy ráérése. Előfordulhat, hogy egy tanár, más intézményben is tanít. Ilyenkor kérést fogalmaz meg, hogy a hét melyik napján, milyen időben ne osszák be.

Egy lehetséges büntetőpontozás

Osztályütközés: 1000
Tanárütközés: 1000
Teremhiány: 1000
Kemény tanárráérés: 500
Többszörös órák: 100
Napközi lyukasóra: 100
Első nap eleji lyukasóra: 50
Napi óraszám egyenetlensége: 50
Heti egyenletesség: 20
Napon belüli szakadozottság: 20
Lágy tanárráérés: 10

Fitnesz függvény lehet pl.: $\frac{1}{1+x^2}$ (biztosítja a jó konvergenciát)
Cél a fitnesz függvény értékének **minimalizálás**a.

Menügenerálási feladat

Követelmények:

- Legyen tekintettel a páciens **energia (kalória) igén**yére,
- A táplálkozási normák szerinti **helyes táplálék összetétel**ére,
- Biztosítsa a **változatosság**ot,
- Tegye lehetővé az **interaktív korrekció**kat,
- Legyen tekintettel a páciens egyes **speciális megkötés**ére (a nem kedvelt és/vagy problémát okozó komponensek minimalizálására),
- Legyen tekintettel **orvosi megkötések**re (pl. diabetes vagy más orvosilag megkülönböztetendő – a kardiovaszkuláris problémához gyakran társuló – betegségek esetén),
- Esetleg a **szezonális beszerzés**i lehetőségekre és a beszerzési költségekre.

Feladatok lehetnek

- Egy webes felületű rendszer a felhasználó által kitöltött kérdések alapján **diagnózis**t állít fel
- A szerver paraméterként megkapja a **felhasználó** táplálkozással kapcsolatos **igénye**it:
  - allergiás a tejtermékekre,
  - nem szereti a sült burgonyát,
  - a sertéshúst és az uborkát,
  - továbbá diétás javaslatot szeretne készíttetni.
- Vele egy időben egy **orvos** konfigurálhatja a weben keresztül a **javaslatai**t. Például egy célcsoport számára kér javaslatot, melyben a szénhidrát tartalmát alulról és felülről, a zsírtartalom értékét csak felülről korlátozza.
- A szerver ezeket az adatokat feldolgozza és a diagnózis illetve a korlátok segítségével, a **dietetikus szakértők** által feltöltött adatbázisból pontos **korlátokat ad a tápanyag-összetevőkre**, és meghatározza azoknak az **optimális mennyiség**ét.

Mire lehet szükségünk

- a megoldást meghatározó **paraméterek** halmaza,
- a paraméterek **lehetséges értékei**nek a halmaza,
- a megoldás **jóság**át a paraméter-értékek által hordozott információ, és a paraméter-értékek egymással való összefüggése határozza meg (az általános és a felhasználó specifikus értékek tükrében ítéljük meg) Pl. ebéd esetén: leves, feltét, köret, kiegészítő és ivólé attribútumok részhalmaza. E véges számú attribútumnak kell értéket adnunk. Ezen értékek tápanyag-tanácsadás esetén lehetnek a receptkönyveinkben megtalálható receptjeink.

Javaslatok a genetikus algoritmus használatához

A kromoszómában (egyedben) az információt **direkt érték kódolás**sal tárolhatjuk.
Étkezéseket generáló genetikus algoritmusnál a gyakorlatban ez azt jelenti, hogy az egyes gének sztring típusú változók, melyek egyértelműen meghatározzák az adatbázisban tárolt receptet.
Ebéd esetén a kromoszóma 5 sztringből áll, melyek mindegyike egy ételt (receptet) reprezentál.
Pl.

- Gyümölcsleves
- Párolt rizs
- Párizsi szelet
- Káposzta saláta
- Rostos almalé

A direkt érték kódoláshoz tartozó keresztezési operátor működése:

Gyümölcsleves	Húsleves	Gyümölcsleves	Húsleves
Párolt rízs	Nokedli	Párolt rizs	Nokedli
Párizsi szelet	Sertéspörkölt	Párizsi szelet	Sertéspörkölt
Káposzta saláta	Savanyú uborka	**Savanyú uborka**	Káposzta saláta
Rostos almalé	Rostos baracklé	**Rostos barackl**é	Rostos ivólé

A keresztezési folyamat a keresztezési pont megválasztásával kezdődik.
A keresztezési pontot a legegyszerűbb esetben véletlenszerűen választjuk meg.
A kromoszómák (egyedek) paraméter-értékeit felcseréljük a keresztezési ponttól kezdődően.

Mutáció, fitneszfüggvény

A **mutáció** egy paraméter értékét változtatja meg véletlenszerűen. Ki lehet próbálni, hogy a keresztezés és a mutáció valószínűségét mekkorának érdemes választani! (80%, illetve 10-40% között várható a legjobb eredmény)

A megoldáshoz olyan **fitnesz függvén**yt kell használnunk, mely lehetőséget ad a korlátokon kívül eső tápanyag-összetevő értékkel rendelkező javaslatok elvetésére, a helyes értékkel rendelkező egyedekhez pedig jó osztályzatot rendel, továbbá a nem harmonizáló étkezéseket kiszűri.

A GA minden egyes iterációjában az osztályozó függvény jósági értéket rendel a kromoszómákhoz.

Pl. Korlátozhatjuk az abban megtalálható húsmennyiséget, vagy a só mennyiségét.

Heti étkezés tervezése esetén a fitnesz függvény osztályozhat a heti terv sótartalmának, a levesek vagy a hideg vacsorák számának függvényében.

Szabálybázis

A fitnesz függvény táplálkozás-szakértői szempontból **szabályalapú pontozás**t használhat.
Egy adatbázisban tárolt szabálytörzs szabályait az osztályozó függvény minden értékelésnél illeszti a kvantitatív szempontból már értékelt egyedre, és végrehajtja a szabályban tárolt utasítást.

Az étkezésekre vonatkozó szabálybázis egy sora pl.:

Köret	Feltét	Ivólé	Leves	Kiegészítő	Utasítás	Paraméter
Tetszőleges	Fehér hús	Paradicsomos ivólé	Paradicsomleves	Tetszőleges	Ront	50%

Amennyiben a szabálybázisban ez a sor szerepel és a fitnesz függvény olyan egyedet osztályoz, amelyre igaz, hogy paradicsomos ivólét és paradicsomlevest tartalmaz és a feltét a lehetséges értékek halmazszerű tárolásában a fehér hús halmazba esik, akkor végrehajtja az utasítást, jelen esetben felére rontja az egyed pontszámát.
A szabálybázis a táplálkozási szakértők által meghatározott **statikus szabályok**ból és a generálás során létrejövő **dinamikus szabályok**ból áll.

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).
- **Populáció mérete** A populáció mérete nyilvánvalóan alapvetően befolyásolja mind a futás, mind a tárhely igényét.
- **Generációk száma** A generációk számával egyenesen arányos a futási idõ, 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.

Következtetések

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