

Mesterséges Intelligencia

Metaheurisztikus algoritmusok
alapjai

FONTOS

- Az alábbi anyag munkavázlat, hibákat tartalmazhat. Amennyiben hibát találnak, kérem, a portálon keresztül üzenetben jelezzék, hogy melyik heti előadás, vagy jegyzet melyik részében, milyen hibát véltek felfedezni!
- Az anyagok kizárólag a Széchenyi István Egyetem 2021-2022 tavaszi félévében Mesterséges Intelligencia kurzust felvett hallgatói számára készültek, kizárólag az adott félév kurzusaihoz használható fel!
- Az alábbi hivatkozásokon megnyitott minden fájl automatikusan begyűjti a hallgató különböző egyedi azonosítóit, mely alapján beazonosítható lehet. Ennek megfelelően a hivatkozásokat ne osszák meg egymással (különösen a kurzust nem hallgatókkal), mert abból az egyedi azonosítók visszakereshetők és a személyazonosság meghatározható!
- Az alábbi anyagra vonatkozóan minden jog fenntartva!
- Az anyagok bármely részének vagy egészének nyomtatása, másolása, megosztása, sokszorosítása, terjesztése, értékesítése módosítással vagy módosítás nélkül egyaránt szigorúan tilos!

A lecke főbb témakörei

- Keresés, optimalizálás
- Metaheurisztikus algoritmusok
 - Általános leírás
 - Tesztelés
 - Csoportosítás
- Konkrét módszerek

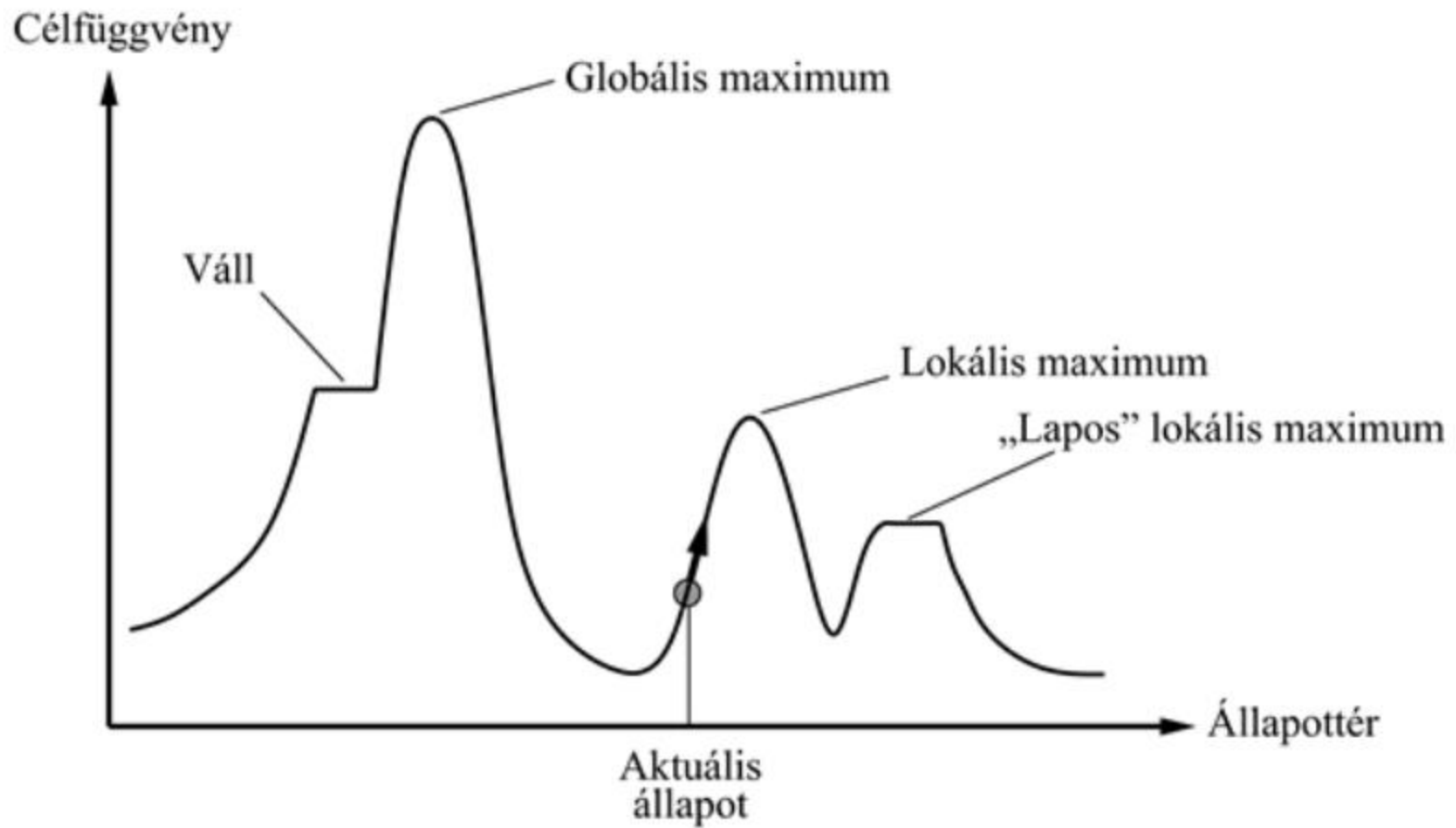
Metaheurisztikus algoritmusok alapjai

Optimalizálás

Optimalizálás

- Cél több alternatív lehetséges megoldás közül valamilyen szempont, vagy szempontok alapján a legjobb (optimális) megoldás kiválasztása
- A keresési tér
 - az egydimenzióstól szinte a végtelen dimenziósig terjedhet
 - lehet diszkrét és folytonos is
- Globális optimum
 - ha a teljes keresési térre vonatkozó legjobb megoldásról van szó
 - Lehet belőle több is
 - jósága teljesen megegyezik, de elhelyezkedésük eltérő
- Lokális optimum
 - a keresési tér egy kisebb részére vonatkozó optimum
 - a megoldás (jelölt) szűk környezetére vonatkozó legjobb megoldás
 - Természetesen a globális optimum bizonyos szempontból lokális optimum is

Optimalizálás



Optimalizálás

- Egyes esetekben a keresési tér bonyolultsága miatt nem, vagy nagyon nehezen, sok idő alatt található meg az optimális megoldás
 - Pl. a különböző NP-nehéz feladatok köre
 - utazóügynök probléma
 - ilyen esetekben a garantált optimális megoldást csak az összes megoldás összehasonlítása révén lehet előállítani
 - egyáltalán nem hatékony
 - a valós problémák esetében a rendelkezésünkre álló erőforrások korlátosak
 - akár számítási teljesítményben, akár időben
 - előfordul, hogy kompromisszumként nem a globális optimumot keressük, hanem egy úgynevezett kvázi optimális megoldást
 - A kvázi optimum lehet az optimumot adott mértékben megközelítő megoldás, vagy az adott idő, számítási lépések alatt elérhető addigi legjobb.

Optimalizálás

Néhány híres NP-nehéz feladat:

- Hozzárendelési feladat (Assignment Problem)
- Utazó Ügynök Probléma (Traveling Salesman Problem)
- Ládapakolás (Bin Packing)
- Hátizsák probléma (Knapsack Problem)
- Halmazlefedési feladat (Set Cover Problem)

Optimalizálás

- A keresés során megkülönböztethetjük az informált és a nem informált keresést
 - Előbbi esetén az egyes megoldás jelöltekről rendelkezünk valamilyen minőségi leírással
 - A nem informált keresés esetén csak annyi információval rendelkezünk, hogy a kiválasztott megoldás megegyezik-e a keresett céllal

Metaheurisztikus algoritmusok alapjai

Metaheurisztikus módszerek

Determinisztikus algoritmusok

- Determinisztikus algoritmusok
 - konvencionális optimalizáló algoritmusok
 - hegymászó algoritmus, lineáris és nemlineáris programozás
 - két (vagy több) lefutás során azonos bemenetet feltételezve mindig ugyanazt az eredményt hozzák, a lépések nem térnek el

Determinisztikus algoritmusok

- Sztochasztikus algoritmusok
 - működésében nagy szerepet játszanak a véletlen számok
 - Heurisztikus
 - nem garantálják, hogy megtalálják az optimumot (persze ez nincs kizárva)
 - a probléma (keresési tér) valamilyen előzetes ismerete alapján felállított stratégia mentén kutatja át azt
 - kvázi optimális megoldást gyakran nagyságrendekkel gyorsabban talál meg a determinisztikus megoldásokhoz képest
 - metaheurisztikus
 - nincs egységes meghatározása
 - felsőbb heurisztikát jelent
 - iteratív folyamatok irányítására szolgáló stratégiák, melyek célja (kvázi)optimum megoldásokat találni a keresési tér hatékony felderítése révén

Metaheurisztikus algoritmusok alapjai

Metaheurisztikus
keresőalgoritmusok tesztelése

Metaheurisztikus keresőalgoritmusok tesztelése

- sztochasztikus jelleg
 - rendkívül nehéz objektíven összehasonlítani
 - egyetlen metaheurisztikus módszer sem jelent univerzálisan jó megoldást minden problémára
 - az egyedi jellegzetességeik, „viselkedésük” viszont egy-egy típusú probléma esetén előnnyel szokott járni
- empirikus tesztek végzik
 - az elért eredmények alapján hasonlítják össze a különböző paramétereket, mint a futási idő, a számítási igény, vagy a keresési lépések száma, vagy a megtalált legjobb megoldás
- A legalapvetőbb megoldás a különböző, kimondottan optimalizációs algoritmusok tesztelésére szolgáló (benchmark) függvények
 - (és egyéb adathalmazok, mint például gráfok)
 - különböző matematikai jellemzőkkel bírnak
- Például a COCO (COMparing Continuous Optimisers)
 - Folytonos Optimalizáló Összehasonlító keretrendszer
 - célja, hogy egységes tesztkörnyezetet biztosítson

Metaheurisztikus keresőalgoritmusok tesztelése

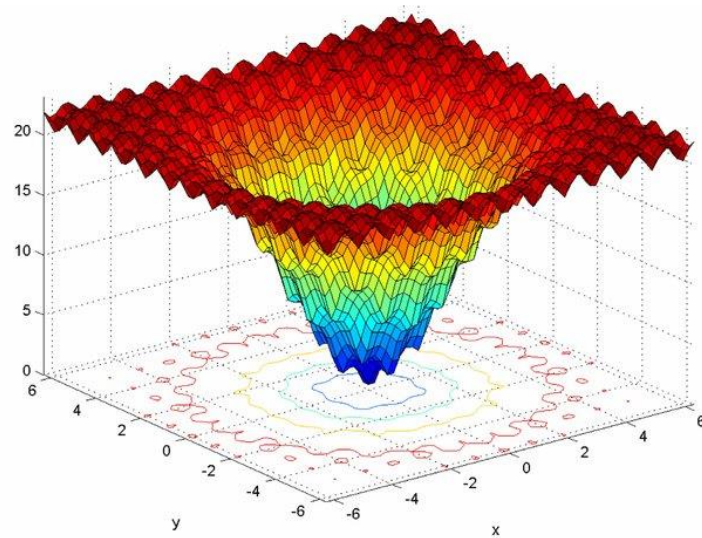
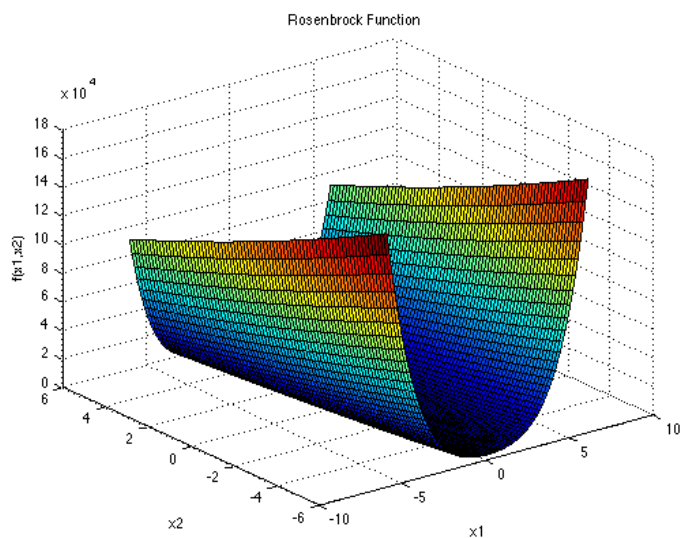
- Benchmark függvények csoportosításának főbb szempontjai
 - Modalitás
 - a félrevezető csúcsok száma utal a függvény modalitására, aminek lényegében a célja, hogy minél több „lehetőséget” adjon arra (a nem kívánatos helyzetre), hogy lokális optimumba „ragadjon” az algoritmus a viselkedésétől függően;
 - tálak (basin)
 - egy nagyobb területet körbezáró nagy meredekségű alakra utal (melyből több is lehet egy függvényben), ami nagyon „vonzó” egyes algoritmusok számára, hiszen egyes heurisztikák a hirtelen javulás irányába indulnak;
 - völgyek (valley)
 - akkor beszélünk völgyről, ha kisebb változatosságú szűk területet meredek „falak” vesznek körül, hasonlóan a tálakhoz, ez is különösen vonzó az olyan megoldások számára, amely két megoldás jósága közötti eltérés nagysága alapján dönt a következő lépésről;

Metaheurisztikus keresőalgoritmusok tesztelése

Modalitás

Unimodális

multimodális



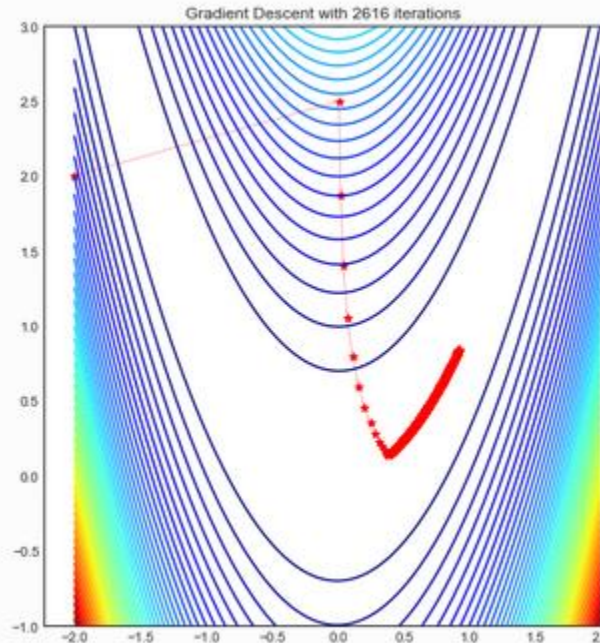
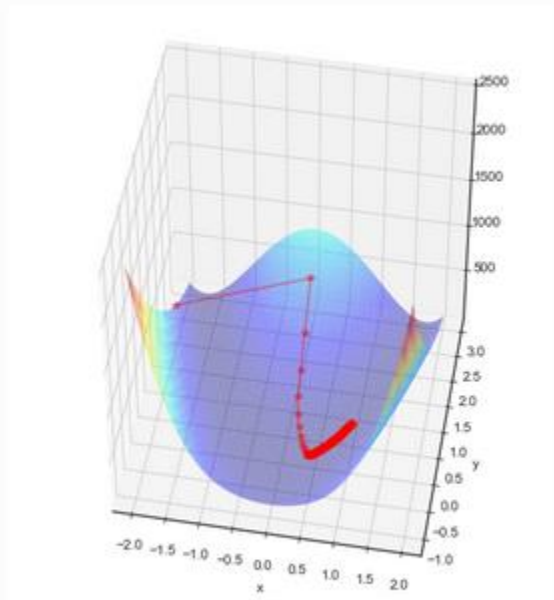
Metaheurisztikus keresőalgoritmusok tesztelése

Völgyek

Rosenbrock-függvény megoldása a Legmeredekebb lejtő (Gradient descent) módszerrel

$$p_{n+1} = p_n - \eta \nabla f(p_n)$$

ahol ∇f gradiens vektor



Metaheurisztikus keresőalgoritmusok tesztelése

- szeparálhatóság (separability)
 - a különböző benchmark függvények bonyolultságára utal, a szeparálható problémákat jellemzően könnyű megoldani, hiszen minden függvényváltozó független a többi változótól, melynek köszönhetően független optimalizációs folyamatok végezhetőek el, vagyis minden változó (vagy azok egy csoportja) külön optimalizálható;
- dimenziók (dimensionality)
 - általánosságban elmondható, hogy egy probléma (függvény) dimenzióinak számával együtt nő annak bonyolultsága is, mivel a paraméterekkel együtt a keresési tér mérete exponenciálisan nő.

Metaheurisztikus keresőalgoritmusok tesztelése

Szeeparálhatóság

Például $z=x^2+y^2$

Kérdés: $z=(x^2+y^2)^4$ esetén függetlenek –e a változók egymástól?

Válasz: igen, mert a következő feltétel teljesül mindegyik változóra

$$\frac{\partial f(\vec{x})}{\partial x_i} = g(x_i)h(\vec{x})$$

Metaheurisztikus algoritmusok alapjai

Metaheurisztikus optimalizáló
algoritmusok csoportosítása

- az algoritmusok e családja nagyszámú
 - tagjai eltérő jellemzőkkel rendelkezhetnek
- számos csoportosítási lehetőség is fellelhető a
 - Az egyik szerint 5 fő csoportba sorolhatók a különböző jellemzőik alapján

Koncepció

- természet inspirálta módszerek
 - bakteriális evolúciós algoritmus, az ősrobbanás – nagy reccs (Big Bang – Big Crunch) algoritmus, a hangyakolónia algoritmus, vagy a mesterséges immunrendszer módszer
- nem természet inspirálta módszerek
 - tabu keresés, vagy az iterált lokális keresés

Iterált lokális keresés

```
algorithm ILS ( $x, f(\cdot), \text{history}$ )  
1    $t := \text{LocalSearch}(x, f(\cdot)); x_b := t;$   
2   while (stopping criterion not satisfied)  $\rightarrow$   
3        $s := \text{perturbation}(t, \text{history});$   
4        $\hat{s} := \text{LocalSearch}(s, f(\cdot));$   
5        $t := \text{AcceptanceCriterion}(t, \hat{s}, \text{history});$   
6       if ( $f(t) < f(x_b)$ ) then  $x_b := t;$   
7       endif  
8   endwhile  
9   return ( $x_b$ );  
end ils
```

Tabu keresés

- minden lépésnél elfogadhatók a rosszabbodó lépések, ha nem áll rendelkezésre javító lépés (például amikor a keresés megakad egy lokális minimumon).
- Ezenkívül tiltásokat vezetnek be (a továbbiakban a tabu kifejezés), hogy megakadályozzák a keresést, hogy visszatérjen a korábban látogatott megoldásokhoz.
- A tabu keresés megvalósítása memóriastruktúrákat használ, amelyek leírják a felkeresett megoldásokat vagy a felhasználó által megadott szabálykészleteket. Ha egy lehetséges megoldást egy bizonyos rövid távú időszakon belül korábban meglátogattak, vagy szabályt sértett, akkor " tabu "-ként (tiltott) jelölik, hogy az algoritmus ezt a lehetőséget ne vegye figyelembe ismételten.

Egyidőben tárolt megoldások

- Populáció alapú
 - egyszerre több megoldásjelölttel rendelkeznek a
 - ezek fejlődés (evolúciója) során történik a megoldások optimalizálása
 - Lehetséges több párhuzamos populáció is
- Egy pontos
 - egyetlen megoldással dolgoznak
 - trajektória módszerek (trajectory methods)
 - lényegében egy pont fejlődése során egy utat (trajektóriát) jár be
 - Például: tabu keresés, változó szomszéd kereső eljárás (Variable Neighborhood Search).

Célfüggvény időbeni változása

- Dinamikus célfüggvényű módszer
 - például az irányított lokális keresés (Guided Local Search)
 - az optimalizáció során változtatja
 - célja az, hogy elkerülje az esetleges lokális optimumokba való ragadást
- Statikus célfüggvényű módszer
 - A leggyakoribb

Szomszédsági struktúra

- Szomszédsági struktúrájú
 - amennyiben a fitness topológia nem változik a keresés során
- változó szomszédsági struktúrájú
 - például a változó szomszéd kereső eljárás eltérő szomszédsági struktúrákat alkalmaz
 - lehetővé teszi, hogy váltson a különböző fitness terek között

Memória

- Memóriát használó
 - keresési folyamat lépéseit tárolja
 - rövid távú memóriával rendelkező
 - például az előző lépések (vagy egy részük), bizonyos döntések, az addigi megoldások közül kiválasztottak
 - hosszú memóriával rendelkező
 - valamilyen akkumulált szintetikus paramétereket használnak
- Memória mentes
 - nem tárolnak visszamenőleges adatokat

Metaheurisztikus algoritmusok alapjai

Genetikus Algoritmus

Genetikus Algoritmus

- John Holland mutatta be az 1960-as években
- különösen nagy népszerűségnek örvendett az 1970-es évek elejétől kezdve
- és mára szinte megszámlálhatatlan módosított változata létezik
- alapelve Darwin evolúciós elméletén alapul, pontosabban a természetes szelekción
 - a környezetükhöz legjobban alkalmazkodott egyedek tovább élnek és több (kisebb változással rendelkező) utódot hoznak létre

Genetikus Algoritmus

- az egyes megoldásokat (jelölteket) úgynevezett kromoszómákban kódolja
 - bináris vektorként reprezentálnak (ezeket hívják még egyedeknek is)
 - általánosították tetszőleges n elemű vektorra
 - állhat karakterekből, egész számokból, vagy valós számokból
 - ez reprezentálja a természetes genomot
- A populáció adott számú kromoszómából áll
 - különböző megoldásjelöltek
 - az együtt élő élőlényeknek feleltethető meg
- Az egy időben (ciklusban) „élő” megoldások alkotják a generációt
- Az egyes megoldások jóságának leírására a fitness függvény szolgál.

Genetikus Algoritmus

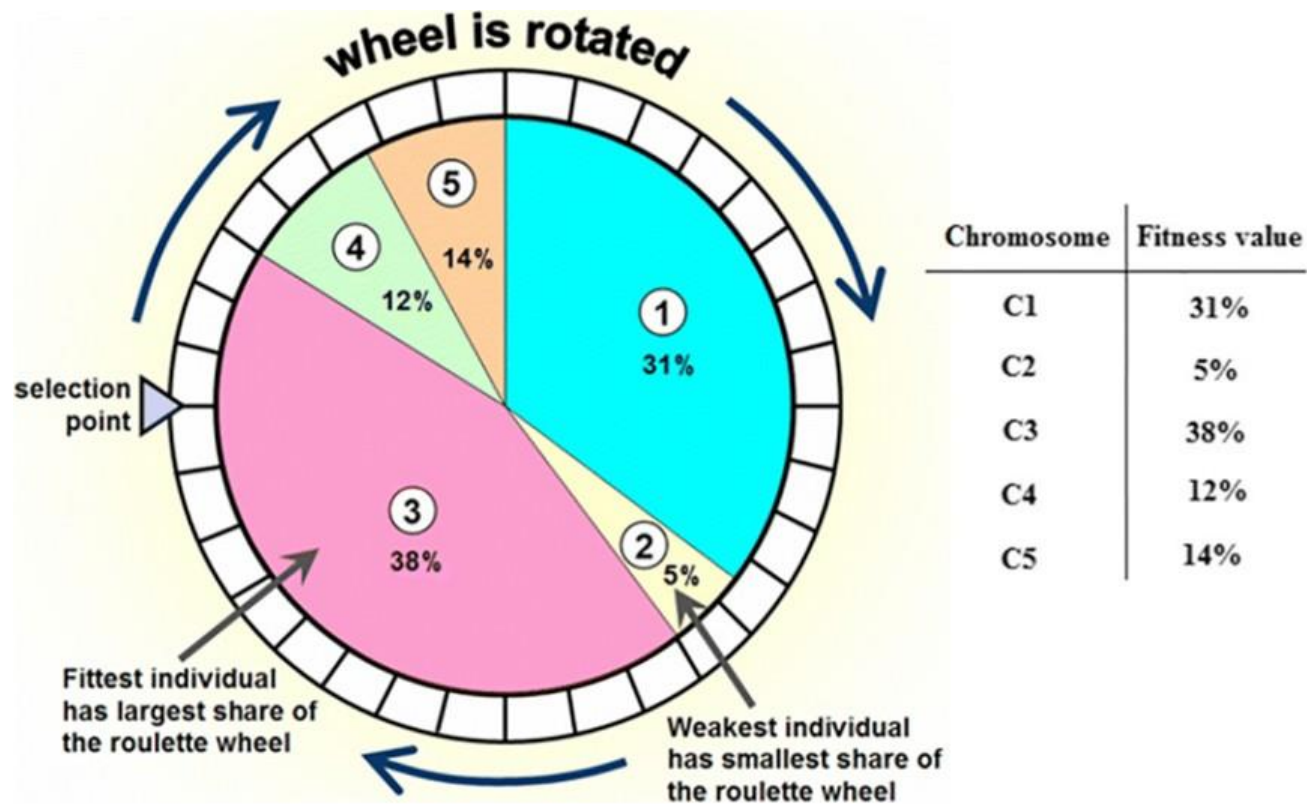
- három alapvető genetikus operátor
 - a szelekció
 - kiválasztja a továbbjutó, keresztezésre jelölt egyedeket
 - a keresztezés
 - Két kromoszómából állít elő egy (vagy több) új kromoszómát
 - lényegében két megoldást kombinálva új megoldást alkot
 - a mutáció
 - Adott valószínűséggel bármelyik egyedben végbe mehet
 - véletlenszerűen módosítja a kromoszóma egy részét
- Az egyes operátorokra több alternatív módszer is létezik.

Metaheurisztikus algoritmusok alapjai

Genetikus Algoritmus: szelekció

Fitnesz arányos kiválasztás

- Másnéven rulettkerék módszer
- A kromoszómák a jóságukkal arányos méretű szeletet kapnak a „rulettkerékből”
 - A kis jóságú kromoszómák is kiválasztásra kerülhetnek
 - Csak kisebb valószínűséggel
 - A jobb egyedek nagyobb valószínűséggel kerülnek kiválasztásra



Forrás: Shirani Faradonbeh, Roohollah & Hasanipناه, Mahdi & Bakhshandeh Amnieh, Hassan & Jahed Armaghani, Danial & Monjezi, Masoud. (2018). Development of GP and GEP models to estimate an environmental issue induced by blasting operation. *Environmental Monitoring and Assessment*. 190. 10.1007/s10661-018-6719-y.

További módszerek


- Random szelekció
- Levágó szelekció (truncation selection)
- Verseny szelekció (tournament selection)
 - Pár-verseny szelekció (binary tournament selection)
- Jutalom alapú szelekció (reward-based selection)
- Sztochasztikus univerzális szelekció
- Stb.

Verseny szelekció

Chromosome #	C_1	C_2	C_3	C_4	C_5	C_6	C_7
Fitness value	10	1	8	6	9	4	7


Tournament size= 3

Randomly 3 chromosomes are selected



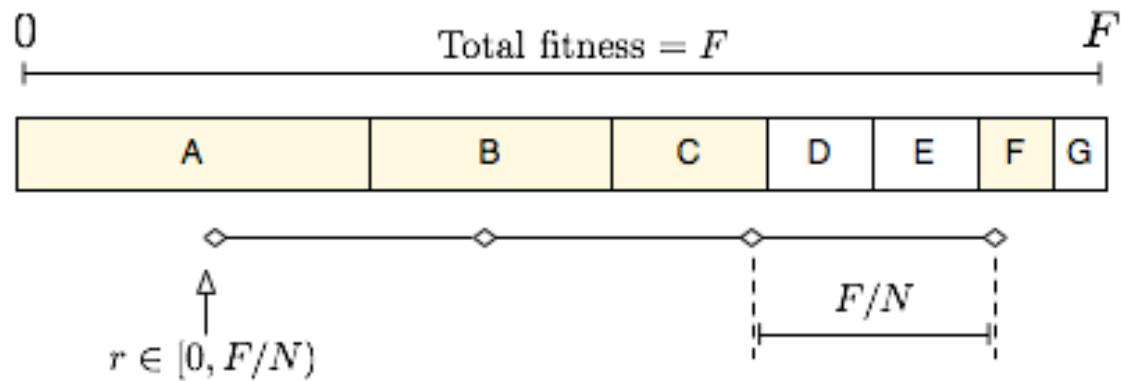
Chromosome #	C_2	C_6	C_7
Fitness value	1	4	7

Chromosome with best Fitness is selected



Winner Chromosome #	C_7
Fitness value	7

Sztochasztikus univerzális szelekció



Metaheurisztikus algoritmusok alapjai

Genetikus Algoritmus: keresztezés

Keresztezés operátor

- Számos alternatív megoldás
- Nagyan függ a kódolástól
 - A kromoszómákat leíró adatszerkezet és az arra vonatkozó megkötések
 - Fák, gráfok, vektorok stb.
- Tipikus keresztező megoldások a vektor-jellegű kromoszómák esetén
 - Egy/többpontos keresztezés
 - Uniform keresztezés
 - Aritmetikai keresztezés

Egypontos keresztezés

- A két szülő kromoszómát egy véletlenszerűen kiválasztott pontban elvágjuk és a levágott részeket felcserélve megkapjuk az utódokat

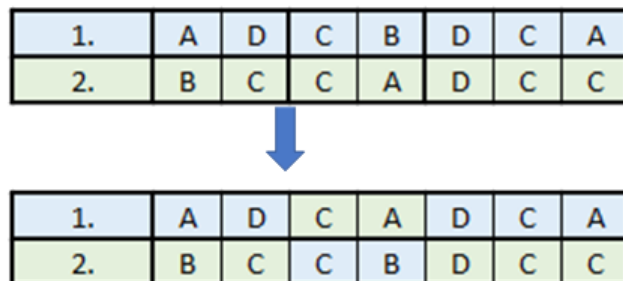
1.	A	D	C	B	D	C	A
2.	B	C	C	A	D	C	C



1.	A	D	C	B	D	C	C
2.	B	C	C	A	D	C	A

Többpontos keresztezés


- A két szülő kromoszómát több (>2) véletlenszerűen kiválasztott pontban elvágjuk és a levágott részeket felcserélve megkapjuk az utódokat



Uniform keresztezés

- A két szülő kromoszómát minden második pontban (felváltva) felcserélve megkapjuk az utódokat
 - Lényegében egy „minden pontos keresztezés”

1.	A	D	C	B	D	C	A
2.	B	C	C	A	D	C	C



1.	A	C	C	A	D	C	A
2.	B	D	C	B	D	C	C

aritmetikai keresztezés

- A két szülő kromoszómát egy tetszőleges f függvény segítségével kombináljuk
 - Tipikus függvény: $z_i = f(x_i, y_i)$

$$f(x_i, y_i) = \alpha * x_i + (1 - \alpha) * y_i \mid \alpha \in [0; 1]$$

	1	2	3	4	5	6	7
x	A	D	C	B	D	C	A
y	B	C	C	A	D	C	C

↓

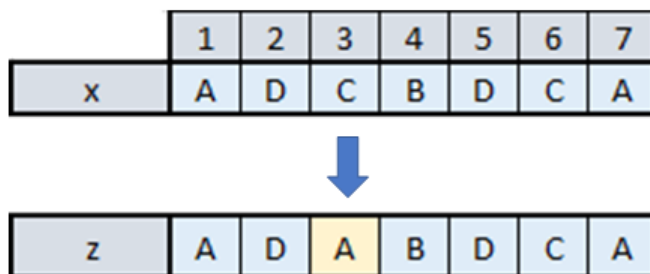
z	$f(x_1, y_1)$	$f(x_2, y_2)$	$f(x_3, y_3)$	$f(x_4, y_4)$	$f(x_5, y_5)$	$f(x_6, y_6)$	$f(x_7, y_7)$
---	---------------	---------------	---------------	---------------	---------------	---------------	---------------

Metaheurisztikus algoritmusok alapjai

Genetikus Algoritmus: mutáció

Mutációs operátor

- Kromoszómák véletlenszerű megváltoztatása
 - Több változata is van
 - Eredetileg
 - A populáció elemeit adott valószínűséggel kiválasztjuk és egy véletlenszerűen kiválasztott gént módosítunk



Metaheurisztikus algoritmusok alapjai

Genetikus Algoritmus: pszeudokód

Genetikus Algoritmus

```
kezdeti populáció létrehozása
generáció=0
amíg generáció < max. generáció
{
    egyedek rangsorolása az alkalmassági érték alapján
    szelekció
    keresztezés
    mutáció
    visszahelyettesítés
    generáció = generáció + 1
}
```


Visszahelyettesítés

Populáció mérete jellemzően fix, de a keresztezés és a mutáció során úgy egyedeket is létrehoztunk

Kérdés: Melyik egyedeket tartsuk meg a populációban a következő generációban?

Két nagy csoport:

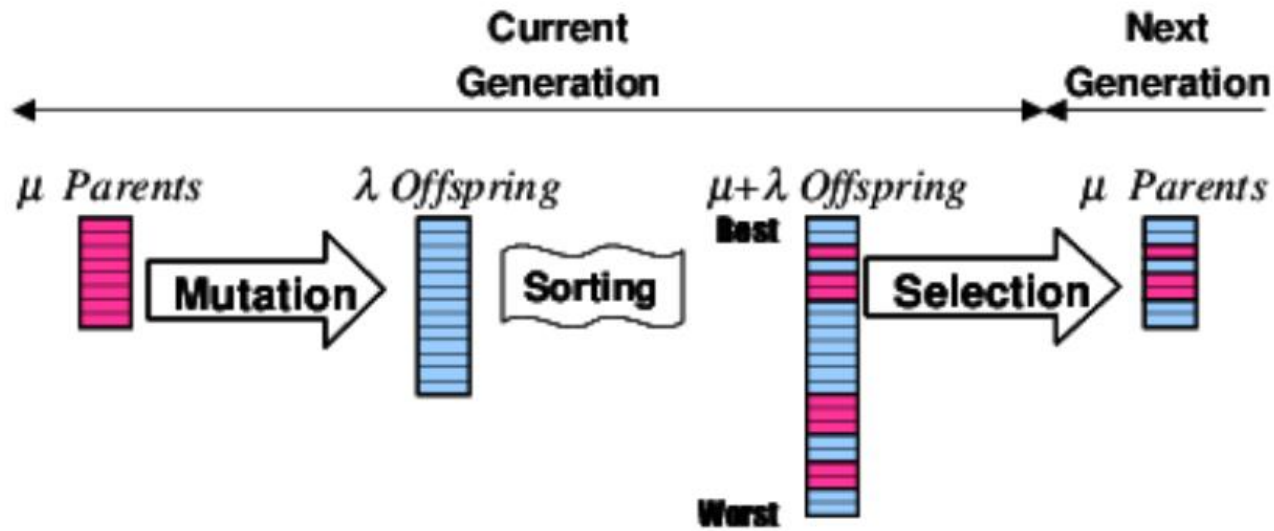
- Fitness alapú módszerek
- Életkor alapú módszerek: adott esetben segíthet egy lokális optimumból kimászni

Életkor alapú módszerek

- **Generációs modell:** ugyanannyi egyedet (λ) hozunk létre a keresztezés és a mutáció során összesen amennyi a populáció mérete (μ)
Az újonnan létrehozott egyedek fogják a populációt alkotni
- $\lambda < \mu$: legrégebbi (FIFO) stratégia

Fitness alapú módszerek

- **Legrosszabb egyedek helyettesítése:** μ egyedszámú populáció legrosszabb λ egyedét helyettesítjük a létrehozott λ utóddal
- **$(\mu + \lambda)$ kiválasztás:** μ elemű populáció és λ utód együttes rangsorolása, majd legjobb μ db egyed kiválasztása



Fitness alapú módszerek

- **(μ, λ) kiválasztás:** tipikusan $\lambda > \mu$ több utódot hozunk létre mint a populáció mérete
 λ db utód rangsorolása, és a μ db legjobb fogja a populáció alkotni

Metaheurisztikus algoritmusok alapjai

Bakteriális Evolúciós Algoritmus

Bakteriális Evolúciós Algoritmus

- közvetlen elődjének az 1997–ben bemutatott pszeudo-bakteriális genetikus algoritmus
 - fuzzy logikai vezérlők generálására alkottak meg
 - A genetikus algoritmusokhoz képest annyiban volt új a módszer, hogy bakteriális mutációt használt
- A bakteriális evolúciós algoritmust 1999–ben N. E. Nawa és T. Furuhashi publikálták
 - céljuk fuzzy rendszerek optimális paramétereinek keresése volt
 - a baktériumok fejlődésének koncepcióját alkalmazták
 - A bakteriális mutáción felül a génátadás, vagy más néven a géntranszfer operátort alkalmazza.

Bakteriális Evolúciós Algoritmus

- baktériumok, vagy egyedek
 - kódolt (megoldás) jelöltek az adott problémára.
- Az egyedek valamely tulajdonságait az úgynevezett gének tárolják
 - azok értékei az allélok
- Az egyes baktériumok jóságának, vagy alkalmassági mértékének meghatározására nincs szükség külön fitness függvényre
 - az adott probléma kiértékelése használatos erre a célra
- Az egyedek összessége alkotja a baktérium populációt
 - az azonos időben létező egyedek képezik az egyes generációkat

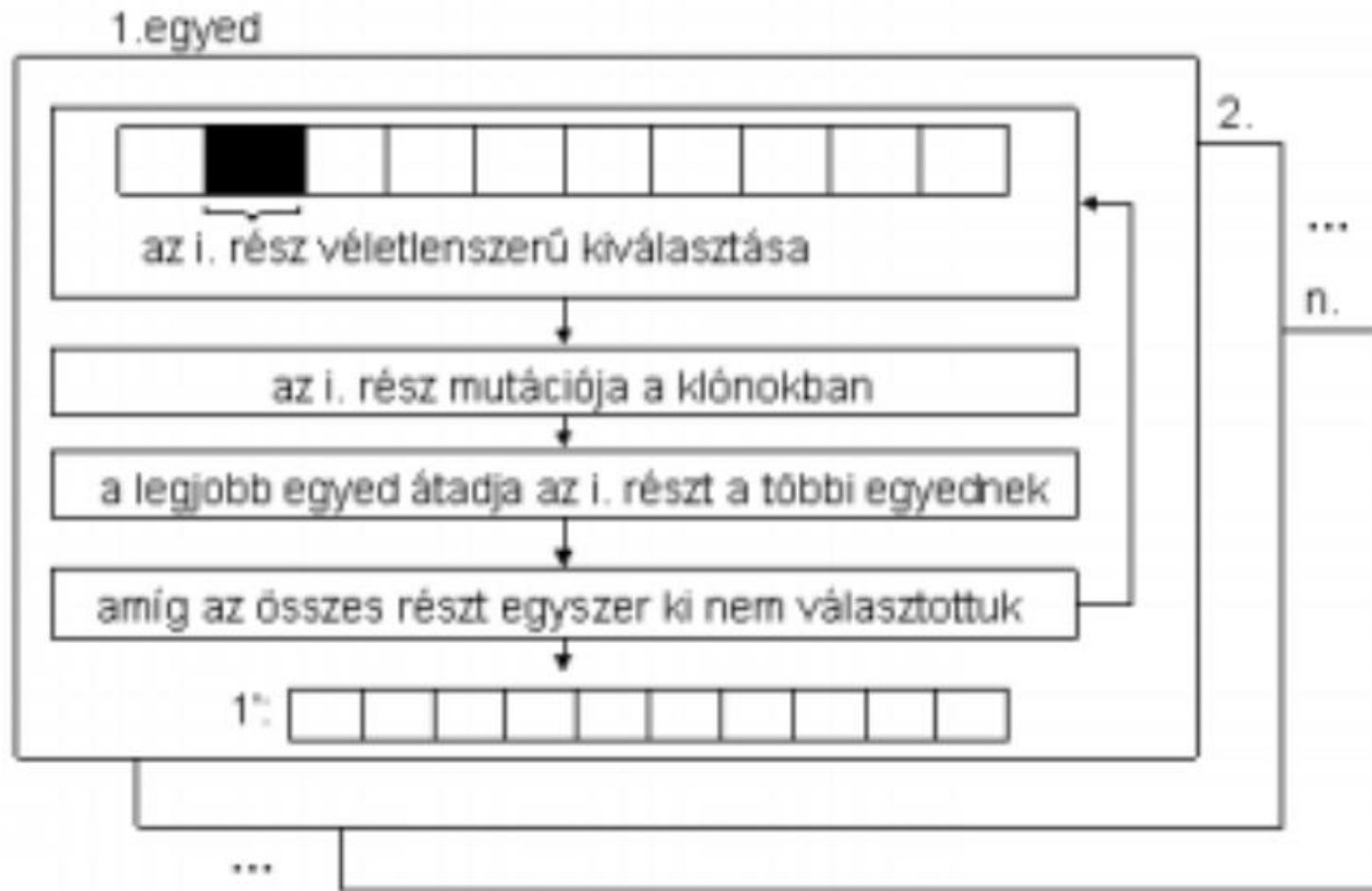
Metaheurisztikus algoritmusok alapjai

Bakteriális Evolúciós Algoritmus:
bakteriális mutáció

Bakteriális mutáció

- A populáció minden egyedén (baktériumon) külön-külön végrehajtott operátor
- A baktérium jóságának „önálló”, más egyedtől származó információátadás nélküli javítása a cél
- A módszer garantálja, hogy az eredeti baktérium jósága nem romolhat
- Lépései egy kiválasztott baktériumra vonatkozóan
 - A kiválasztott baktériumot n_k számban klónozzuk (identikus másolatokat hozunk létre)
 - Egy véletlenszerűen kiválasztott allélt véletlenszerűen megváltoztatunk az összes klónban (de az eredeti baktériumban nem)
 - Kiértékeljük a módosított klónokat és kiválasztjuk a legjobb egyedet a klónok és az eredeti baktérium közül
 - A legjobb egyed átadja a korábbi lépésben kiválasztott allélját minden másik egyednek (klónoknak és/vagy eredeti baktériumnak)
 - Ezt a lépéssorozatot addig ismételjük, amíg minden allél kiválasztásra nem került
 - A lépések végén a módosított baktériumot visszahelyezzük a populációba
 - *legrosszabb esetben a kiindulási paraméterekkel fog rendelkezni*

Bakteriális mutáció



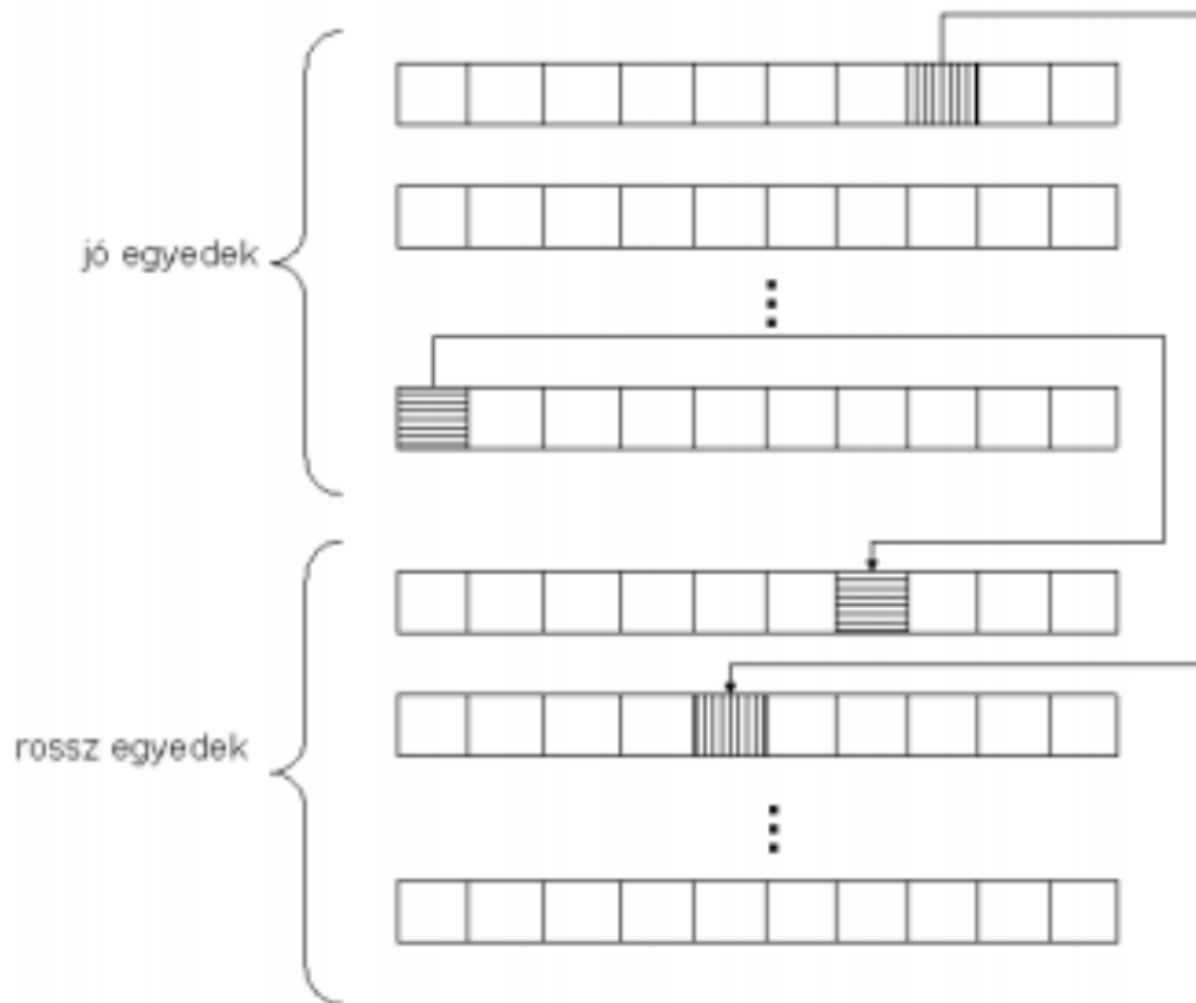
Metaheurisztikus algoritmusok alapjai

Bakteriális Evolúciós Algoritmus:
génátadás

Génátadás

- A célja, hogy a populáció egyedei közötti információcserével javítsa a baktériumok jóságát
- A lépéseit n_i (infekciók száma) alkalommal ismételjük meg
- Lépései
 - A populációban található baktériumokat jóság szerint sorbarendezzük
 - Egy előre definiált n_c pontban elvágjuk a populációt
 - Ezzel a jó és rossz baktériumok csoportjára osztjuk azt
 - A jó és a rossz baktériumok csoportjából véletlenszerűen kiválasztunk egy-egy egyedet
 - A kiválasztott jó (forrás) baktérium átadja egy véletlenszerű allélját a kiválasztott rossz (cél) baktériumnak
 - Amennyiben rögzített hosszúságúak a baktériumok, akkor a célbaktérium felülírja az allélt
 - Eltérő hosszúságú baktériumok esetén hozzá is fűzheti az allélt

Génátadás



Metaheurisztikus algoritmusok alapjai

Bakteriális Evolúciós Algoritmus:
pseudokód

Bakteriális Evolúciós Algoritmus

- **Kezdeti populáció létrehozása** /*Baktériumok véletlenszerű generálása*/
- **Generáció = 0**
- **Amíg a terminálási feltétel nem teljesül** /*Az aktuális generáció szám kisebb, mint a megengedett maximális generációk száma*/
 - **Minden egyedre**
 - **Bakteriális mutáció**
 - **Amíg el nem érjük a maximális génátadások számát**
 - **Génátadás**
 - **Generáció += 1**

Metaheurisztikus algoritmusok alapjai

Részecske-sereg Optimalizáció

Részecskesereg Optimalizáció

- particle swarm optimization
- R. Eberhart és J. Kennedy publikálták 1995–ben
 - céljuk nemlineáris függvények optimalizációja volt
 - két paradigmát is javasoltak, illetve vizsgáltak
 - Az egyik globálisan, míg a másik lokálisan orientált részecskesereg módszer

Részecskesereg Optimalizáció

- Az algoritmus részecskék mozgásának az analógiáját alkalmazza
 - a füst gomolygásához hasonló az egyes egyedek (részecskék) mozgása
 - hasonlít például a madarak és a halak összehangolt mozgására is
- A részecskék a megoldás jelölteket jelképezik, amelyek a keresési térben mozognak.

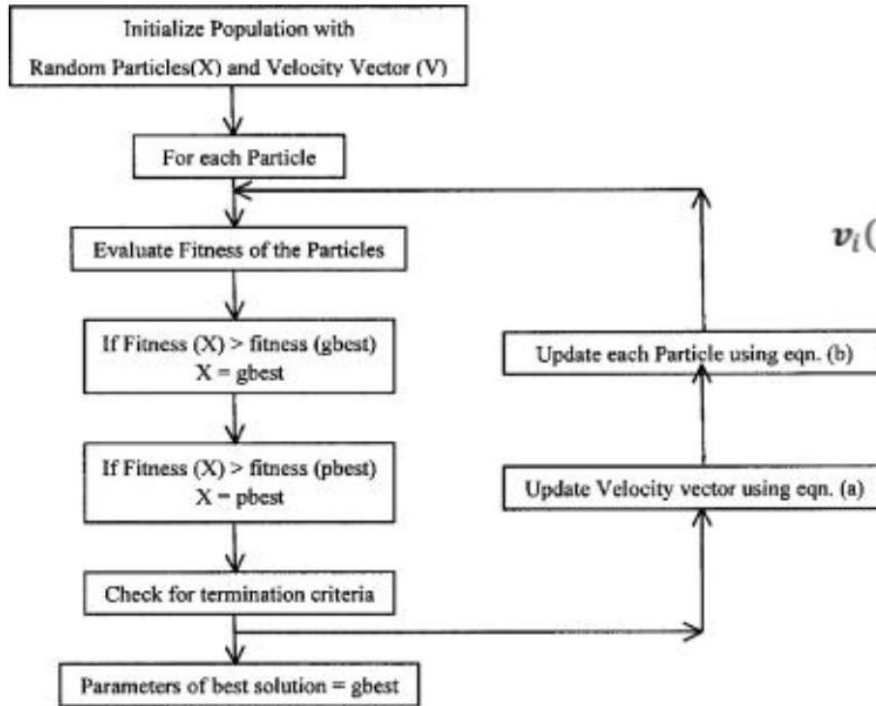
Metaheurisztikus algoritmusok alapjai

Részecske-sereg Optimalizáció:
pseudokód

Részecskesereg Optimalizáció

- **Kezdeti részecskék létrehozása a D dimenzió** /*Részecskék pozícióinak és sebességeinek véletlenszerű generálása*/
- **Amíg a terminálási feltétel nem teljesül, addig *minden részecskére***
- ***A kiválasztott minimalizálási függvény kiértékelése a D változókra***
- **Ha a részecske aktuális értéke jobb, mint a részecske legjobb értéke, akkor**
- ***A legjobb értéket az aktuális értékre cseréljük***
- **Ha a részecske legjobb értéke jobb, mint a globális legjobb érték, akkor**
- ***A legjobb globális értéket az aktuális részecske értékre cseréljük***
- **Részecske sebességének frissítése**
- **Részecske mozgása az új pozícióba**

Részecskecsereg optimalizáció



$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1)$$

Local search

Global search

$$\mathbf{v}_i(t+1) = w\mathbf{v}_i(t) + r_1C_1(P_i^l(t) - \mathbf{x}_i) + r_2C_2(P^g(t) - \mathbf{x}_i)$$

Metaheurisztikus algoritmusok alapjai

Ősrobbanás – nagy reccs

Ősrobbanás – nagy reccs

- Big Bang – Big Crunch
- O. K. Erol és I. Eksin publikálták 2005-ben
- céljuk egy általuk fejlesztett újszerű optimalizációs módszer bemutatása volt
- lényege az univerzum fejlődési analógiájára épül
 - Ősrobbanás
 - egyetlen pontból tágulással jött létre az ismert univerzum
 - nagy reccs
 - ahol az univerzum egyetlen pontba roskad
- Az univerzum maga a keresési tér
- Az univerzum pontjai (az égitestek) az egyes egyedek, vagyis a megoldás jelöltek.

Ősrobbanás – nagy reccs

- A jószág kiszámítsa valamilyen előre megadott fitness függvény alapján történik
 - az univerzum keletkezésének analógiájára ez jelképezi a világegyetemben (keresési térben) található égitestek (egyedek) tömegét
- benchmark jellegű tesztben összehasonlították algoritmusukat egy genetikus algoritmussal
 - az eredményeik alapján az új módszer több esetben felülmúlta a genetikus algoritmus által hozott eredményeket

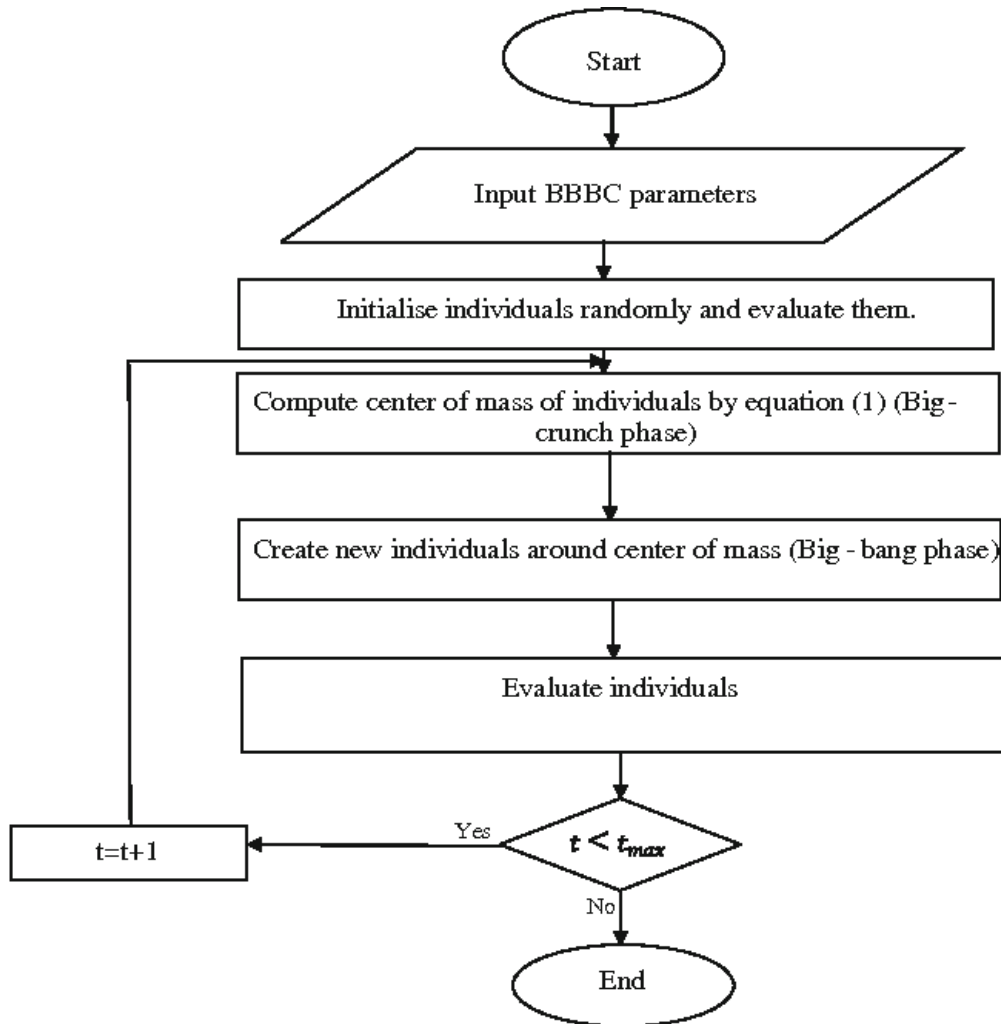
Metaheurisztikus algoritmusok alapjai

Ősrobbanás – nagy reccs:
pseudokód

Ősrobbanás – nagy reccs

- ***Kezdeti populáció véletlenszerű létrehozása***
/*Egyedek generálása a probléma keresési terében*/
- **Amíg a terminálási feltétel nem teljesül, addig**
- ***Minden egyedre***
- ***Jóság* kiszámítása**
- **Súlypont meghatározása az egyedek alapján**
- **Új egyedek létrehozása a súlypont körül** /*Az iterációnként módosított keresési térben*/

Ősrobbanás – nagy reccs



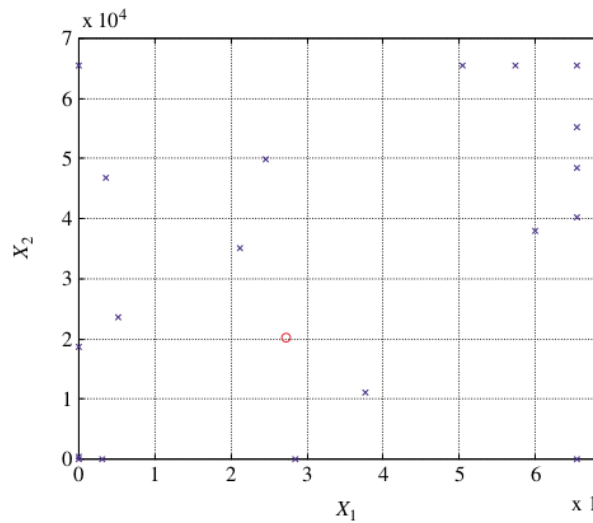
$$\vec{x}^c = \frac{\sum_{i=1}^N \frac{\vec{x}^i}{f_i}}{\sum_{i=1}^N \frac{1}{f_i}}$$

$$x^{new} = x^c + \frac{L\gamma}{k}$$

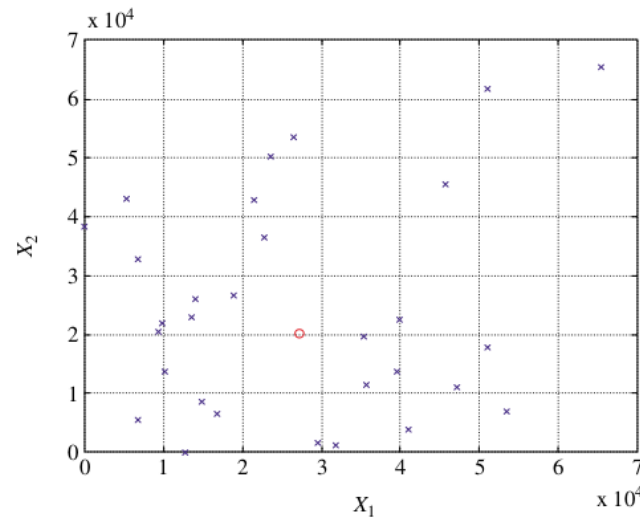
k iterációszám

Ősrobbanás – nagy reccs példa

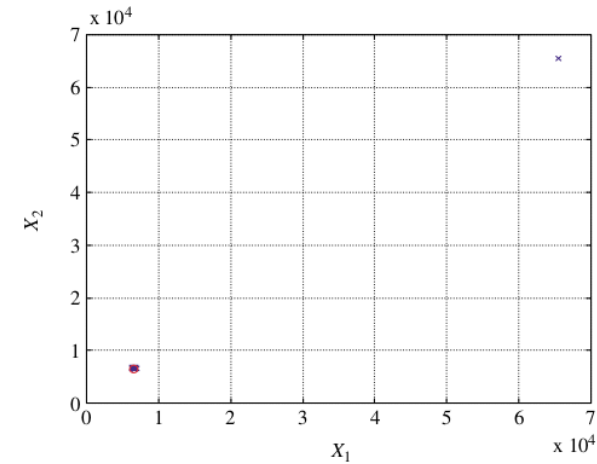
Rosenbrock-függvény minimumának keresése



Kezdetben



4. iterációban



500. iterációban

Metaheurisztikus algoritmusok alapjai

Imperialista Kompetitív Algoritmus

Imperialista Kompetitív Algoritmus

- Imperialist Competitive Algorithm
- E. Atashpaz-Gargari és C. Lucas publikálta 2007-ben
- céljuk egy a birodalmak versengésének mintájára épített optimalizációs algoritmus létrehozása volt
- az egyes országok jelképezik a probléma egy-egy megoldását a keresési térben, vagyis az egyedeket
 - két csoportra oszthatóak, az erősebbek a birodalmak (imperialists), míg a gyengébbek a gyarmatok (colonies)
 - A gyarmatok a különböző birodalmak között szétosztásra kerülnek.

Imperialista Kompetitív Algoritmus

- Az országok erősségének meghatározására a költség függvény szolgál
 - a fitness függvényhez hasonló
 - működése eltér
 - alacsony költségfüggvény = erős ország
 - nem csak az egyedek önálló költségfüggvényét határozza meg
 - A birodalmak és gyarmataik együttes költségfüggvényét is felhasználja
- A gyarmatok közelítenek a felettük álló birodalmakhoz
 - A mozgás következtében azok költségfüggvény-értéke változhat és jobb lehet a hozzájuk tartozó birodaloménál
 - az egyes gyarmatok erősebbé válhatnak, mint a följük rendelt birodalom
 - Az ilyen esetekben az adott gyarmat „átveszi a hatalmat”
- A versengés során a gyengébb birodalmak elveszíthetik a gyengébb gyarmataikat
 - erősebb birodalomhoz kerülhetnek
- Az erőtlen birodalmak össze is omolhatnak
 - Eltűnnek, gyarmatai szétszétásra

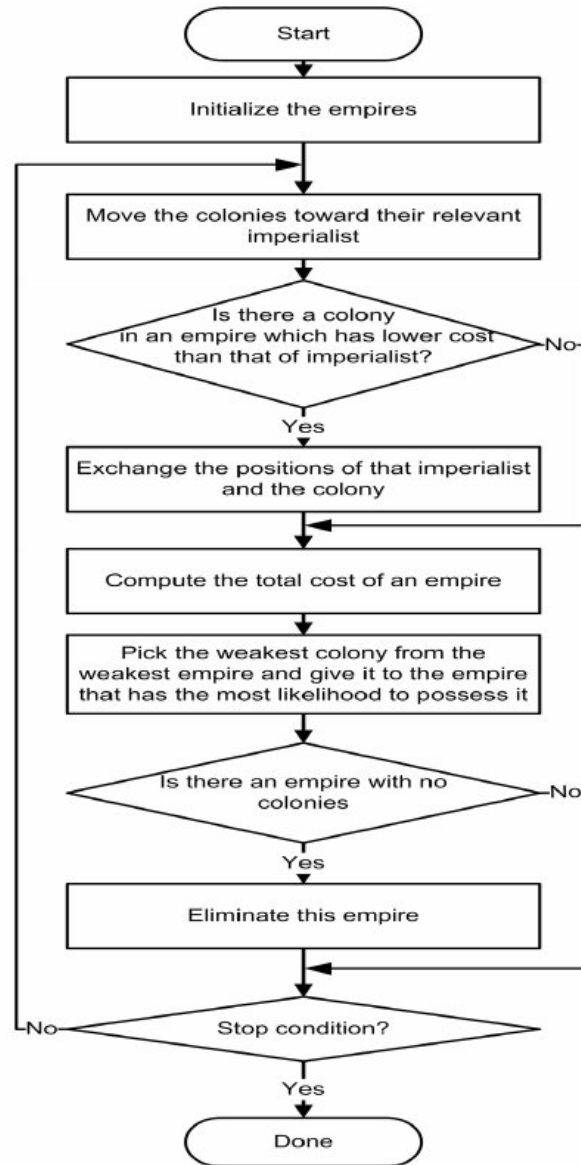
Metaheurisztikus algoritmusok alapjai

Imperialista Kompetitív Algoritmus:
pseudokód

Imperialista Kompetitív Algoritmus

- *Kezdeti országok véletlenszerű létrehozása* /*Egyedek generálása a probléma keresési terében*/
- *Minden országra*
 - *Költség függvény* kiszámítása
- A birodalmak és gyarmatok kiválasztása
- Amíg a terminálási feltétel nem teljesül, addig
 - *Minden gyarmatra*
 - *Mozgatás a birodalma irányába*
 - *Minden gyarmatra*
 - Ha a gyarmat erősebb, mint a birodalma, akkor
 - Gyarmat és birodalom felcserélése
 - *Minden birodalomra*
 - *A birodalom teljes erejének meghatározása* /*A birodalom és a gyarmatai együttes, aggregált ereje*/
 - *A leggyengébb birodalom vagy birodalmak leggyengébb gyarmatának, vagy gyarmatainak kiválasztása*
 - *A kiválasztott gyarmat(ok) véletlenszerű átadása valamely birodalomnak, vagy birodalmaknak* /*A birodalmak ereje alapján eltérő valószínűséggel kaphatják meg a gyarmato(ka)t*/
- *Az erőtlen birodalmak eliminálása, gyarmataik szétosztása*

Imperialista Kompetitív algoritmus



Felhasznált források

- Botzheim János Dr. - Kóczy T. László Dr. - Tikk Domonkos: *Intelligens rendszerek*. Győr : Széchenyi István Egyetem, 2008. 287 p.
- S. Forrest, and M. Mitchell, Relative building-block fitness and the building-block hypothesis, in L. D. Whitley (Eds.): *Foundations of Genetic Algorithms 2*, Morgan Kaufman, San Mateo, CA, 1993.
- X.-S. Yang, *Nature-Inspired Metaheuristic Algorithms*, Luniver Press, Cambridge, UK, 2010.
- C. Blum and A. Roli, *Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison*, *ACM Computing Surveys*, 35(3), pp. 268–308, 2003.
- N. E. Nawa, T. Hashiyama, T. Furuhashi, and Y. Uchikawa, Fuzzy logic controllers generated by pseudo-bacterial genetic algorithm, In *Proceedings of the IEEE International Conference on Neural Networks 1997*, pp. 2408–2413, 1997.
- J. H. Holland, *Adaptation in Natural and Artificial Systems*, The MIT Press, Cambridge, Massachusetts, 1992.
- N. E. Nawa and T. Furuhashi, Fuzzy system parameters discovery by bacterial evolutionary algorithm, *IEEE Transactions on Fuzzy Systems*, 7(5), pp. 608–616, 1999.
- J. Kennedy and R. Eberhart, Particle Swarm Optimization, In *Proceedings of IEEE International Conference on Neural Networks 1995*, pp. 1942–1948, 1995.
- R. Eberhart and J. Kennedy, A New Optimizer Using Particle Swarm Theory, In *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pp. 39–43, 1995.
- O. K. Erol and I. Eksin, A new optimization method: Big Bang - Big Crunch, *Advances in Engineering Software*, vol. 37, Elsevier, pp. 106–111, 2005.
- H. Tang, J. Zhou, S. Xue and L. Xie, Big Bang–Big Crunch optimization for parameter estimation in structural systems, *Mechanical Systems and Signal Processing*, vol. 24, Elsevier, pp. 2888–2897, 2010.
- A. Kaveh and S. Talatahari, Size optimization of space trusses using Big Bang–Big Crunch algorithm, *Computers and Structures*, vol. 87, Elsevier, pp. 1129–1140, 2009.
- E. Atashpaz-Gargari and C. Lucas, Imperialist Competitive Algorithm: An Algorithm for Optimization Inspired by Imperialist Competition, In *Proceedings of IEEE Congress on Evolutionary Computation 2007*, pp. 4661–4667, 2007.
- H. Duan, C. Xu, S. Liu and S. Shao, Template matching using chaotic imperialist competitive algorithm, *Pattern Recognition Letters*, vol. 31, Elsevier, pp. 1868–1875, 2010.
- T. Niknam, E. T. Fard, N. Pourjafarian and A. Roustaei, An efficient hybrid algorithm based on modified imperialist competitive algorithm and K-means for data clustering, *Engineering Applications of Artificial Intelligence*, vol. 24, Elsevier, pp. 306–317, 2011.
- S. Talatahari, B. Farahmand Azar, R. Sheikholeslami and A. H. Gandomi, Imperialist competitive algorithm combined with chaos for global optimization, *Communications in Nonlinear Science and Numerical Simulation*, vol. 17, Elsevier, pp. 1312–1319, 2012.
- http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO.html
- <http://www.sfu.ca/~ssurjano/optimization.html>
- <http://coco.gforge.inria.fr/>
- https://www.toshiba.co.jp/rdc/rd/detail_e/e1904_01.html
- <https://aishsearch.github.io/#/>
- <https://github.com/fcampelo/EC-Bestary>