Haladó informatikai algoritmusok

„Nézzünk körül a környezetünkben!” algoritmusok

Bacterial Foraging (BFOA)

- Rajintelligencia (Swarm Intelligence) elven működő eljárás
- A rajintelligencia (kollektív intelligencia) módszerek közös tulajdonsága, hogy nagyszámú homogén egyed viselkedésmintáin alapulnak.
- Az alapelv szerint lehetséges, hogy egy individuális egyed nem képes megoldani adott feladatot, azonban ha nagyszámú egyed csoportot alkot, akkor a csoport kollektív intelligenciája már elég lehet a feladat sikeres megoldásához.
- Az E. coli baktérium kolóniák táplálék kereső és reprodukciós viselkedés mintáin alapul működése.
- Az Escherichia Coli az ostoros baktériumok családjába tartozik. Ostorszerű végtagjai, az úgynevezett flagellumok segítségével képes az önálló úszásra. Ez a mozgás a kemotaxis, mely során a fő cél a tápanyagban gazdag helyek elérése, továbbá a valamilyen okból veszélyesnek ítélt helyek elkerülése.
- A baktérium táplálkozás (Bacterial Foraging) az E. Coli baktérium kolónia egyedei által végzett kemotaxist másolja a keresés során. Az egyedek hol tudatosan, hol véletlenszerűen mozognak az egyre jobb megoldást adó pontok felé.

Bemenő paraméterek

- *Problemsize*: A döntési változók száma.
- *Cellsnum*: A baktériumok kezdeti populációjának nagysága. Döntési változónként 10-50 ajánlott. Fontos, hogy 2-vel oszthatónak kell lennie!
- *Ned, Nre, Nc, Ns*: Dispersal, Reproduction, Chemotactic, Maximum úszás lépések száma. A keresést három egymásba ágyazott ciklus valósítja meg.
  - A **legbelső ciklus** maga a kemotaxis, a baktériumok mozgását megvalósító szakasz. Evolúciós algoritmusról van szó, az egyedek meghatározott élettartammal rendelkeznek. Minél jobb pozícióban vannak az adott probléma szempontjából, annál kevésbé csökken az élettartamuk.
  - A **középső ciklus** a reprodukció, minden iteráció során az algoritmus rendezi az egyedeket élettartam szempontjából, majd a halmazt általában középen kettébontva az idősebb baktériumok helyére átmásolja a fiatalabb példányokat, ezzel szimulálva az osztódást.
  - A **külső ciklus** az elimináció, amikor véletlenszerű egyedek meghatározott valószínűséggel likvidálásra kerülnek. A paraméterek értéke nagymértékben befolyásolja a keresés közbeni viselkedést. Általános szabály, hogy a kemotaxis lépesek száma nagy legyen, a többi viszonylag kicsi, pl.: 4, 8, 20, 5 .
- *Stepsize*: Az alaplépés nagysága a problématéren belül.
- *Ped*: Az elimináció valószínűségi változója, érdemes viszonylag nagy érteket választani a dinamikus kereséshez, például 0,25-0,5.

Pszeudokód:

```
    Input: Problemsize, Cellsnum, Ned, Nre, Nc, Ns, Stepsize, Ped
    Output: Cellbest
1.  Population ← InitializePopulation(Cellsnum, 1 Problemsize);
2.  for l = 0 to Ned do
3.      for k = 0 to Nre do
4.          for j = 0 to Nc do
5.              ChemotaxisAndSwim(Population, Problemsize, Cellsnum, Ns, Stepsize);
6.              foreach Cell ∈ Population do
7.                  if Cost(Cell) ≤ Cost(Cellbest) then
8.                      Cellbest ← Cell;
9.                  end
10.             end
11.         end
12.         SortByCellHealth(Population);
13.         Selected ← SelectByCellHealth(Population, Cellsnum/2);
14.         Population ← Selected;
15.         Population ← Selected;
16.     end
17.     foreach Cell ∈ Population do
18.         if Rand() ≤ Ped then
19.            Cell ← CreateCellAtRandomLocation();
20.         end
21.     end
22. end
23. return Cellbest;

25. ChemotaxisAndSwim(Population, Problemsize, Cellsnum, Ns, Stepsize)
26.     foreach Cell ∈ Population do
27.         Cellfitness ← Cost(Cell) + Interaction(Cell, Population);
28.         Cellhealth ← Cellfitness;
29.         Cell' ← ∅;
30.         for i = 0 to Ns do
31.             RandomStepDirection ← CreateStep(Problemsize);
32.             Cell' ← TakeStep(RandomStepDirection, Stepsize);
33.             Cell'fitness ← Cost(Cell') + Interaction(Cell'8 , Population);
34.             if Cell'fitness > Cellfitness then
35.                 i ← Ns;
36.             else
37.                 Cell ← Cell';
38.                 Cellhealth ← Cellhealth + Cell'fitness;
39.             end
40.         end
41.     end
42. end
```

Bat Algorithm (BATA)

- A Bat vagy denevér algoritmust különböző mérnöki problémák megoldására fejlesztették ki.
- Működési elve szerint szintén rajintelligencia eljárás, a denevérek visszhang alapján való tájékozódását másolja.
- A denevérek teljes sötétségben is képesek elejteni zsákmányukat, az általuk kibocsátott hang környezetről való visszaverődése alapján.

Bemenő paraméterek:

- *PopulationSize*: A denevérek populációjának nagysága, 10 és 40 között érdemes tartani a populációszámot.
- *ProblemSize*: A döntési változók száma.
- *Loudness*: A denevérek által kibocsátott hang ereje, az érteket 0 es 1 között érdemes változtatni, 0,5 az ajánlott.
- *PulseRate*: A denevérek által kibocsátott hang kibocsátási gyakorisága, az érteket 0 és 1 között érdemes változtatni, 0,5 az ajánlott.
- *Frequency*: A denevérek által kibocsátott hang hullámhosszának minimális és maximális határértéke. Az algoritmus működése során azt befolyásolja, mekkora lépésközzel dolgozhat az eljárás a problématéren belül. Érdemes úgy megadni, hogy a denevérek a teljes problémateret bejárhassák.

Pszeudokód:

```
    Input: PopulationSize, ProblemSize, Loudness, PulseRate, Frequency
    Output: best
1.  Initialize the bat population xi(i = 1,2, ..., populationSize) and vi
2.  Define pulse Frequency fi at xi
3.  Initialize PulseRates ri and the Loudness Ai
4.  while (t < Max number of iterations)
5.  Generate new solutions by adjusting frequency,
6.  and updating velocities and locations/solutions
7.      if (rand > ri)
8.      Select a solution among the best solutions
9.      Generate a local solution around the selected best solution
10.     end if
11.     Generate a new solution by flying randomly
12.     if (rand < Ai & f(xi) < f(x*))
13.     Accept the new solutions
14.     Increase ri and reduce Ai
15.     end if
16.  Rank the bats and find the current best x*
17.  end while
18.  return best
```

Bees Algorithm (BA)

- A Bees Algorithm (BA) eljárást folytonos matematikai függvények szélső-érték keresésére dolgozták ki.
- Az algoritmus a rajintelligencia eljárások osztályába tartozik.
- A méhek táplálék kereső viselkedése inspirálta.
- A méhkaptárakból először felderítő méhek indulnak nektár után kutatni. A kaptárba visszatérve tudatják a többiekkel a nektár helyét és mennyiséget, melyek függvényében adott számú munkás méh tér vissza velük a nektárhoz.
- Az algoritmus működése során a felderítők folyamatosan keresik az ígéretes pontokat. A pontok fitnesz értéke alapján további egyedek csatlakoznak hozzájuk, és lokális keresést hajtanak végre. A lokális optimumok elkerülése érdekében az algoritmus folyamatosan hoz létre felderítőket véletlenszerű pozíciókban.

Bemenő paraméterek:

- *Beesnum*: A méhek kezdeti populációjának nagysaga. Döntési változónként 10-50 ajánlott.
- *Problemsize*: A döntési változók száma.
- *Sitesnum, EliteSitesnum*: Az ígéretes pontok száma, ahol az algoritmus lokális keresést hajt végre. A második paraméter a kiemelten kezelt ígéretes pontok száma, ahol a lokális keresés alaposabb. EliteSitesnum<Sitesnum, pl. 3 és 1.
- *OtherBeesnum, EliteBeesnum*: A munkás méhek száma, akik a felderítők által talált ígéretes pontokban a lokális keresést végrehajtják. A második paraméterben megadott számú munkás méhek csak a kiemelten kezelt ígéretes pontokban keresnek. EliteBeesnum> Beesnum. Pl. 7 és 3.
- *PatchSizeinit*: Az ígéretes pontokban, lokális keresésnél a keresési tér nagyságának kiinduló értéke, mely a továbbiakban az alábbi képlet szerint alakul: $x_i = x_i \pm rand(1) \times PatchSize$

Az optimumhoz való konvergencia biztosítása érdekében a PatchSize mérete iterációnként egy pozitív konstans értekkel szorzódik, ami kisebb mint 1, pl. 0,95.

```
    Input: Problemsize, Beesnum, Sitesnum, EliteSitesnum, PatchSizeinit, EliteBeesnum,
OtherBeesnum
    Output: Beebest
1.  Population ← InitializePopulation(Beesnum, 1 Problemsize);
2.  while ¬StopCondition() do
3.      EvaluatePopulation(Population);
4.      Beebest ← GetBestSolution(Population);
5.      NextGeneration ← ∅;
6.      Patchsize ← ( PatchSizeinit × PatchDecreasefactor);
7.      Sitesbest ← SelectBestSites(Population, Sitesnum);
8.      foreach Sitei ∈ Sitesbest do
9.          RecruitedBeesnum ← ∅;
10.         if i < EliteSitesnum then
11.             RecruitedBeesnum ← EliteBeesnum;
12.         else
13.             RecruitedBeesnum ← OtherBeesnum;
14.         end
15.         Neighborhood ← ∅;
16.         for j to RecruitedBeesnum do
17.             Neighborhood ← CreateNeighborhoodBee(Sitei, Patchsize);
18.         end
19.         NextGeneration ← GetBestSolution(Neighborhood);
20.     end
21.     RemainingBeesnum ← (Beesnum- Sitesnum);
22.     for j to RemainingBeesnum do
23.         NextGeneration ← CreateRandomBee();
24.     end
25.     Population ← NextGeneration;
26. end
27. return Beebest;
```

Pszeudokód

Cuckoo Search Algorithm (CS)

- A kakukk keresés algoritmust különböző mérnöki problémák megoldására fejlesztették ki.
- Működési elve a különböző kakukkféléktől származik, amelyek idegen, más fajhoz tartozó madarak feszkébe raknak tojást, majd az utódok rendszerint hamarabb kelnek ki, mint a gazdamadár fiókái. A frissen kikelt fióka a többi tojást kilöki a fészekből, így több élelem jut neki a mostohaszülők egyetlen utódjaként.
- Az algoritmus működése során minden tojás egy megoldásnak felel meg, a kakukktojás pedig egy új, potenciálisan jobb megoldás. Ha a kakukktojás valóban jobb megoldás, akkor kilök a feszekből egy sima tojást.

Bemenő paraméterek:

- *NestSize*: A fészkek száma, dimenziónként 10 és 40 között érdemes tartani.
- *ProblemSize*: A döntési változók száma.
- *DiscoveryRate*: Valószínűségi változó, a kakukktojás felfedezésének valószínűsége, értékét 0 és 1 között érdemes változtatni, 0,25 az ajánlott.

Pszeudokód:

```
    Input: Problemsize, NestSize, DiscoveryRate
    Output: best
1.  begin
2.      Generate initial population of NestSize nests xi (i = 1, 2, ..., NestSize)
3.      while (t < MaxGeneration) or (stop criterion)
4.          Get a cuckoo randomly by L´evy flights
5.          evaluate its quality/fitness Fi
6.          Choose a nest among n (say, j) randomly
7.          if (Fi > Fj),
8.              replace j by the new solution;
9.          end
10.         Worse nests < DiscoveryRate are abandoned and new ones are built;
11.         Keep the best solutions (or nests with quality solutions);
12.         Rank the solutions and find the current best
13.     end while
14. end
15. return best
```

Cultural Algorithm (CA)

- Az algoritmus az evolúciós eljárások osztályába tartozik.
- Működése a társadalom kulturális evolúcióján alapul, ami egy generációkon átívelő jelenség. Egy társadalom kultúrájához tartoznak például különböző népszokások, hitek, viselkedési normák, tudományos ismeretek stb.
- Hogy mi marad tartósan a kultúra része, az egyének pozitív és negatív visszajelzései alapján dől el. (A kulturális környezet és az ember közötti kapcsolat tanulmányozásával a kulturális ökológia foglalkozik.)
- Az egyik legegyszerűbb példa egy klasszikus zenei mű, például Mozart Varázsfuvola című műve. Mivel az adott művészeti alkotás tetszett az embereknek, a kultúra része lett és generációkon át fennmaradt.
- A Cultural Algorithm ezen az elven működik, ahol a generációk egyedei egyre jobb megoldásokkal állnak elő. Az újdonságot az jelenti, hogy az egyedek megosztják egymás között a keresési információkat, és a jobb megoldásokat eltárolják egy generációkon átívelő kulturális szinten.
- A kulturális tudásbázis (Knowledge base) a keresés során az egyedek pozitív és negatív visszajelzései alapján változik, egyre jobb megoldásokat adva.

Bemenő paraméterek:

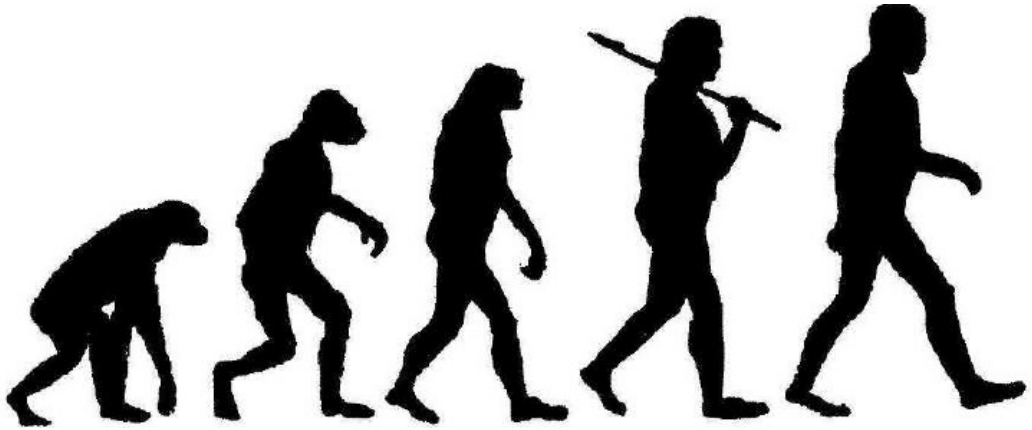
- *Populationnum*: A populáció nagysága. Döntési változónként 50-100 ajánlott.
- *Problemsize*: A döntési változók száma.

Pszeudokód:

```
    Input: Problemsize, Populationnum
    Output: KnowledgeBase
1.  Population ← InitializePopulation(Problemsize, Populationnum);
2.  KnowledgeBase ← InitializeKnowledgebase(Problemsize, Populationnum);
3.  while ¬StopCondition() do
4.      Evaluate(Population);
5.      SituationalKnowledgecandidate ← AcceptSituationalKnowledge(Population);
6.      UpdateSituationalKnowledge(KnowledgeBase, SituationalKnowledgecandidate);
7.      Children ← ReproduceWithInfluence(Population, KnowledgeBase);
8.      Population ← Select(Children, Population);
9.      NormativeKnowledgecandidate ← AcceptNormativeKnowledge(Population);
10.     UpdateNormativeKnowledge(KnowledgeBase, NormativeKnowledgecandidate);
11. end
12. return KnowledgeBase;
```

Differential Evolution (DE)

- Az algoritmus az evolúciós algoritmusok osztályába tartozik, mely eljárások közös tulajdonsága, hogy Darwin evolúciós elméleten alapul működésük.
- Ennek megfelelően központi eleme a természetes kiválasztódás, tehát a problémára jobb megoldást adó egyedek hozhatnak létre új generációt.
- Az evolúció során számos faj esetében megfigyelhető, hogy a generáció váltásokkal az adott környezet kihívásainak egyre inkább megfelelő egyedek jöttek létre.
- A leszármazott egyed új tulajdonságait a szülők tulajdonságainak keresztezéséből kapta.



- Először adott számú (N) kiinduló egyedet hoz létre véletlenszerű pozíciókban a problématéren belül.
- Az egyedszám az algoritmus futása során nem változik.
- Az iterációk során adott generáció (G index) minden tagjának egy új leszármazott egyedet (G+1 index) hoz létre keresztezéssel az alábbi differenciáló képlet (innen ered az algoritmus neve) alapján:

$$\underline{x}_{j,G+1} = \underline{x}_{r_1,G} + F(\underline{x}_{r_3,G} - \underline{x}_{r_2,G})$$

$\underline{x}_{r1}, \underline{x}_{r2}, \underline{x}_{r3}$ Véletlenszerűen kiválasztott egyedek

F Súlyozási tényező

- Ezután összehasonlítja az eredeti és a leszármazott egyed rátermettségét az adott probléma szempontjából.
- A rátermettség fitnesz függvény alapján kerül meghatározásra.
- A kedvezőbb megoldást adó egyed lesz tagja a kivetkező generációnak.

Bemenő paraméterek

- *Populationsize*: A populáció nagysága. Döntési változónként 10-50 ajánlott.
- *Problemsize*: A döntési változók száma.
- *Weightingfactor*: A differenciáló képletben szereplő súlyozási tényező, melynek értéke 0 és 2 között változhat. A szakirodalom 0,8-as értéket ajánl.
- *Crossoverrate*: Valószínűségi változó, mely azt adja meg, hogy az aktuális egyed legyen-e tagja a következő generációnak, vagy a differenciáló képlet alapján keresztezéssel jöjjön létre egy leszármazott egyed. Az értéke valószínűségi változó lévén 0 es 1 között változhat. A szakirodalom szerint 0,9 az ajánlott érték.

Pszeudokód:

```
    Input: Populationsize, Problemsize, Weightingfactor, Crossoverrate
    Output: Sbest
1.  Population ← InitializePopulation(Populationsize, Problemsize);
2.  EvaluatePopulation(Population);
3.  Sbest ← GetBestSolution(Population);
4.  while ¬ StopCondition() do
5.      NewPopulation ← ∅;
6.      foreach Pi ∈ Population do
7.          Si ← NewSample(Pi, Population, Problemsize, Wf, Cr);
8.          if Cost(Si) ≤ Cost(Pi) then
9.              NewPopulation ← Si;
10.         else
11.             NewPopulation ← Pi;
12.         end
13.     end
14.     Population ← NewPopulation;
15.     EvaluatePopulation(Population);
16.     Sbest ← GetBestSolution(Population);
17. end
18. return Sbest;

19. NewSample(Pi, Population, Problemsize, Weightingfactor, Crossoverrate)
20.     repeat
21.         P1 ← RandomMember(Population);
22.     until P1 ≠ P0 ;
23.     repeat
24.         P2 ← RandomMember(Population);
25.     until P2 ≠ P0 ∨ P2 ≠ P1;
26.     repeat
27.         P3 ← RandomMember(Population);
28.     until P3 ≠ P0 ∨ P3 ≠ P1 ∨ P3 ≠ P2 ;
29.     CutPoint ← RandomPosition(NP);
30.     S ← 0;
31.     for i to NP do
32.         if i ≡ CutPoint ∧ Rand() < CR then
33.             Si ← P3i + F × (P1i - P2i );
34.         else
35.             Si ← P0i;
36.         end
37.     end
38.     return S;
39. end
```

Firefly Algorithm (FF) - Szentjánosbogár algoritmus

- Különböző mérnöki problémák megoldására fejlesztették ki.
- Működési elve a szentjánosbogár-féléktől ered, mely rovarok speciális fénykibocsátásuk (bioluminescencia) segítségével találják meg egymást.
- Minél jobb megoldást talál az egyed, annál erősebb fényt bocsájt ki, ami az adott területre vonzza a csoport többi tagját.

Bemenő paraméterek:

- *PopulationSize*: A szentjánosbogarak száma, dimenziónként 10 és 40 között érdemes tartani.
- *ProblemSize*: A döntési változók száma.
- *Gamma*: A fényerősség csökkenésének mértékét definiálja, fontos hatással bír az algoritmus konvergencia sebességére. Az értékét tipikusan 0,01 és 100 között szokták változtatni a feladat függvényében.

Pszeudokód:

```
    Input: Problemsize, PopulationSize, γ
    Output: best
1.  Generate initial population of fireflies xi(i=1,2,...,PopulationSize)
2.  Light intensity Ii at xi is determined by f(xi)
3.  Define light absorption coefficient γ
4.  while (t<MaxGeneration)
5.      for i=1:n all n fireflies
6.          for j=1:i all n fireflies
7.              if (Ij>Ii)
8.                  Move firefly i towards j in d-dimension;
9.              end if
10.             Attractiveness varies with distance r via exp[-γr]
11.             Evaluate new solutions and update light intensity
12.         end for j
13.     end for i
14.     Rank the fireflies and find the current best
15. end while
16. return best
```

Harmony Search (HS)

- Működését a Jazz zenészek azon viselkedés mintája inspirálta, amikor közösen kezdenek el játszani valamilyen darabot, és saját játékukat fokozatosan a zenekarhoz igazítják, zenei harmóniát létrehozva.
- Fals hang esetén kisebb módosításokkal, improvizációval javítanak az előadáson.
- Az algoritmus a legjobb megoldásokat a Harmónia Memóriában (Harmony Memory) tárolja.
- Az új megoldásokat vagy ebből a harmónia memóriából közvetlenül, vagy a harmónia memóriából kisebb módosításokkal, vagy véletlenszerűen a problématéren belül hozza létre.
- A harmónia memória mindig úgy aktualizálódik, hogy a legjobb megoldást adó értékek szerepeljenek benne.
- A Harmony Search algoritmust több különböző jellegű optimalizálási probléma esetében használták sikerrel, például vízvezeték hálózat tervezésénél.

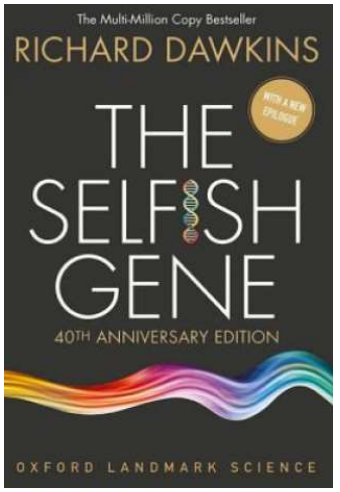
Bemenő paraméterek

- *Pitchnum*: A döntési változók száma.
- *Pitchbounds*: A döntési változó lehetséges minimum és maximum értéke.
- *Memorysize*: A harmónia memória mérete. Döntési változóként 10-50 ajánlott.
- *Consolidationrate*: Valószínűségi változó, mely azt adja meg, hogy az új megoldás a harmónia memória alapján, vagy véletlenszerűen jöjjön létre. Minél nagyobb az értéke, az algoritmus annál gyorsabban konvergál a harmónia memóriában található megoldások felé. A szakirodalom szerint 0,7 és 0,95 közötti értéket ajánl.
- *PitchAdjustrate*: Ha az új megoldást a harmónia memóriában levő érték kisebb módosításával hozzuk létre, a módosítás mértékét adja, érdemes 0,5 alatt tartani.
- *ImprovisationMax*: Az algoritmusban az iteráció számnak felel meg.

Pszeudokód

```
    Input: Pitchnum, Pitchbounds, Memorysize, Consolidationrate, PitchAdjustrate,
Improvisationmax
    Output: Harmonybest
1.  Harmonies ← InitializeHarmonyMemory(Pitchnum, 1 Pitchbounds, Memorysize);
2.  EvaluateHarmonies(Harmonies);
3.  for i to Improvisationmax do
4.      Harmony ← ∅;
5.      foreach Pitchi ∈ Pitchnum do
6.          if Rand() ≤ Consolidationrate then
7.              RandomHarmonyipitch ←
8.              SelectRandomHarmonyPitch(Harmonies, Pitchi);
9.              if Rand() ≤ PitchAdjustrate then
10.                 Harmonyipitch ← AdjustPitch(RandomHarmonyipitch);
11.             else
12.                 Harmonyipitch ← RandomHarmonyipitch ;
13.             end
14.         else
15.             Harmonyipitch ← RandomPitch(Pitchbounds);
16.         end
17.     end
18.     EvaluateHarmonies(Harmony);
19.     if Cost(Harmony) ≤ Cost(Worst(Harmonies)) then
20.         Worst(Harmonies) ← Harmony;
21.     end
22. end
23. return Harmonybest;
```

Memetic Algorithm (MA)



- A Memetics a kulturális információk cserélődését, átadását leíró teória, mely Richard Dawkins 1976-ban megjelent „The Selfish Gene” című művében jelent meg először.
- Lényege, hogy a kulturális információ áramlását az univerzális darwinizmus jegyében írja le. Az univerzális darwinizmus elmélete szerint minden komplex rendszer leírható a biológiai darwini evolúció analógiájára, ahol diszkrét információ egységek terjednek és öröklődnek az individuumok között.
- A meme (mém) a kulturális információ alapegysége (pl. egy ötlet, felfedezés, észrevétel stb.), aminek az elnevezése a biológiában jól ismert génből ered.
- A kulturális információt az egyének elméje tárolja, az egyének közötti kommunikáció révén pedig terjed, sokszorozódik, reprodukálja önmagát.
- Az információ terjedése során a befogadó individuum elméjének függvényében torzulhat, mutálódhat, ezzel gyengítve, vagy éppen erősítve eredeti jelentését.

- Az algoritmus alapvetően egy evolúciós eljárás, ahol az egyedek kommunikálnak egymással.
- A keresési információk mémként terjednek a populációban, az egyedek saját fitnesz értéküktől függően gyengítik, vagy erősítik a mémek jelentőségét.
- A mémeket az algoritmus egy generációkon átívelő kulturális szinten tárolja.

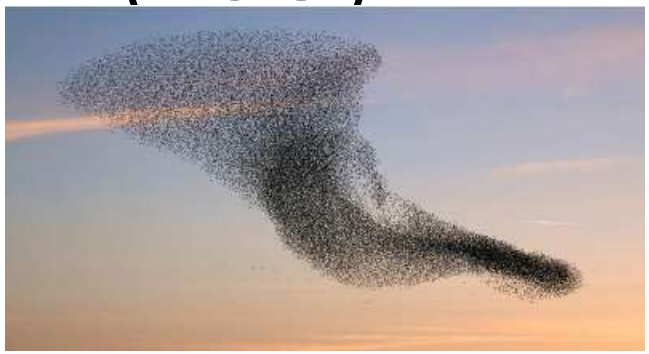
Bemenő paraméterek:

- *ProblemSize*: A döntési változók száma.
- *Populationsize*: A populáció nagysága. Döntési változónként 10-50 ajánlott.
- *MemePopsize*: A generációkon átívelő kulturális szinten eltárolt mémek száma. Döntési változónként 10-50 ajánlott.

Pszeudokód

```
    Input: ProblemSize, Popsize, MemePopsize
    Output: Sbest
1.  Population ← InitializePopulation(1 ProblemSize, Popsize);
2.  while ¬StopCondition() do
3.      foreach Si ∈ Population do
4.          Sicost ← Cost(Si);
5.      end
6.      Sbest ← GetBestSolution(Population);
7.      Population ← StochasticGlobalSearch(Population);
8.      MemeticPopulation ← SelectMemeticPopulation(Population, MemePopsize);
9.      foreach Si ∈ MemeticPopulation do
10.         Si ← LocalSearch(Si);
11.     end
12. end
13. return Sbest;
```

Particle Swarm Optimization (PSO)



- Szintén egy rajintelligencia módszer.
- Napjaink egyik legígéretesebb metaheurisztikus optimalizáló algoritmusa.
- Működését a madár és halrajok mozgása inspirálta.
- A nagyszámú egyed mozgásában egyfajta rendezettség figyelhető meg. Érzékelik egymás helyzetét, bizonyos mértékben pedig emlékeznek a korábbi pozíciókra.
- A keresés adott számú részecske létrehozásával kezdődik, amik véletlenszerű kiindulási pontokban helyezkednek el. A részecskék a problématérben az egyre jobb megoldást adó helyek felé mozognak, a csapatot a legjobb egyedek vezetik.
- Tisztában vannak aktuális és addigi legjobb pozíciójukkal, valamint a részecske csapat addigi legjobb pozíciójával, az új pozíció kiszámításához ezeket is figyelembe veszi az eljárás.
- Hogy az adott pozíció mennyire jó az adott keresés szempontjából, mindig egy fitnesz függvénnyel határozható meg.

- Az iterációk során az alábbi képletek alapján számítható ki a részecskék új pozíciója:

$$V_{id} = w \times V_{id} + c_1 \times rand() \times (p_{id} - x_{id}) + c_2 \times rand() \times (p_{gd} - x_{id})$$

$$x_{id} = x_{id} + V_{id} t$$

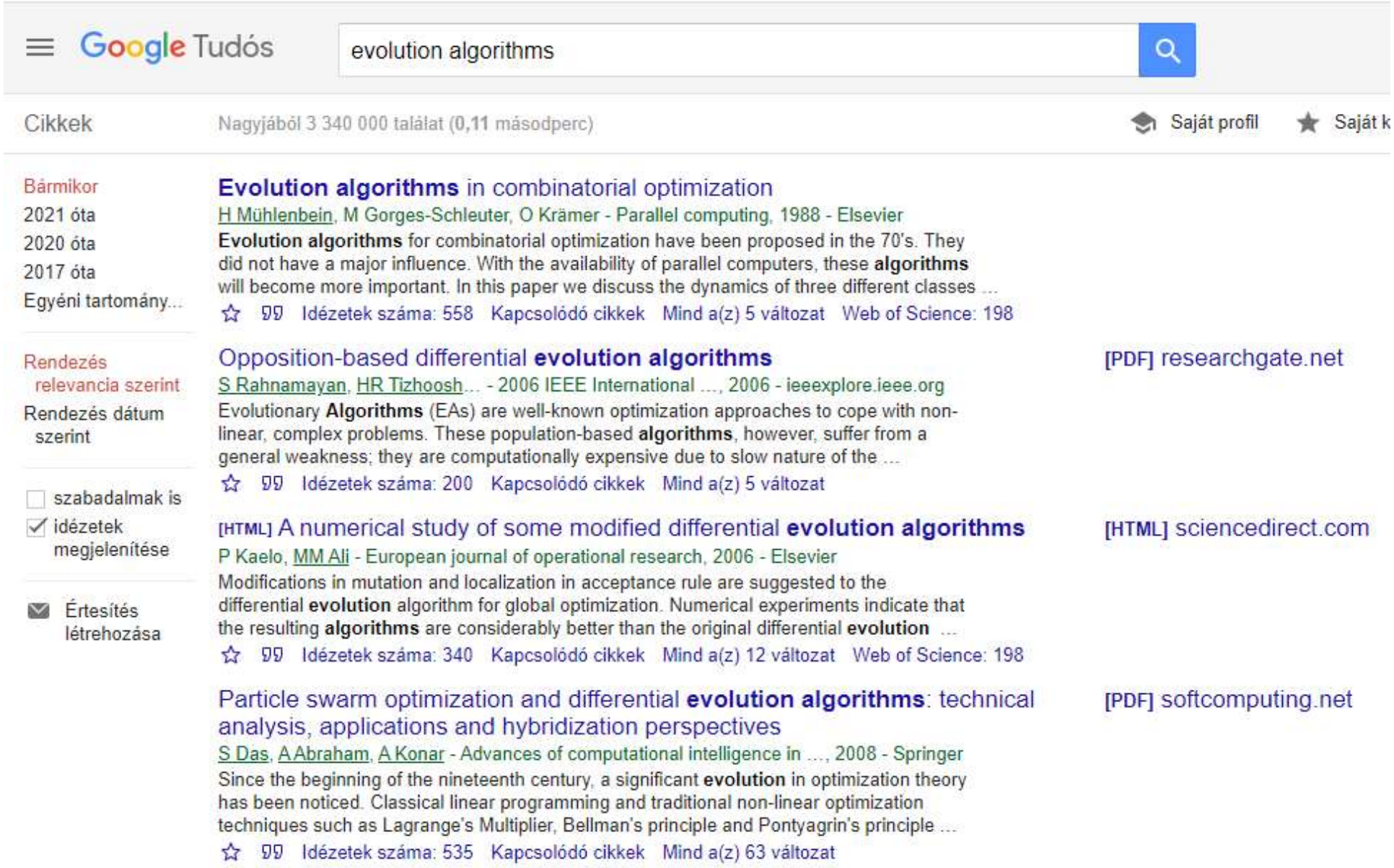
V_{id}	Részecske sebessége
w	Gyorsulás konstans
c_1	Egyéni súlyozás konstans
c_2	Szociális súlyozás konstans
p_{id}	Részecske eddigi legjobb pozíciója
p_{gd}	Csapat eddigi legjobb pozíciója
x_{id}	Részecske aktuális pozíciója
t	Időegység

- Az iteráció addig folytatódik, míg nem teljesül valamilyen leállási feltétel.
- Evolúciós algoritmusok esetében sokszor nem egyszerű egyértelműen jó leállási feltételt találni. Itt a feltétel többek között lehet megszabott számú iteráció végrehajtása, vagy ha a csapat legjobb pozíciója bizonyos számú iteráció után sem változott meg.

Tesztfüggvény	Leíró képlet	Határértékek
Ackley's function (F1)	$f(x) = -20 \times \exp\left(-0{,}2 \times \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i^2)}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + \exp(1)$	$-32{,}768 \le x_i \le 32{,}768$ $x_i = 0, i = 1, \dots, n$ f(x)=0
De Jong's function (F2)	$f(x) = \sum_{i=1}^{n} x_i^2$	$-5{,}12 \le x_i \le 5{,}12$ $x_i = 0, i = 1, \dots, n$ f(x)=0
Drop-Wave function (F3)	$f(x_1, x_2) = -\frac{1 + \cos(12 \times \sqrt{x_1^2 + x_2^2})}{\frac{1}{2}(x_1^2 + x_2^2) + 2} + 1$	$-5{,}12 \le x_i \le 5{,}12$ $x_i = 0, i = 1, \dots, n$ f(x)=0
Easom's function (F4)	$f(x_1, x_2) = -\cos(x_1) \times \cos(x_2) \times \exp(-(x_1 - \pi)^2 - (x_2 - \pi)^2) + 1$	$-10 \le x_i \le 10$ $x_i = \pi, i = 1, \dots, n$ f(x)=0
Griewangk's function (F5)	$f(x) = \frac{1}{4000} \times \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$-600 \le x_i \le 600$ $x_i = 0, i = 1, \dots, n$ f(x)=0
Matyas's function (F6)	$f(x_1, x_2) = 0{,}26 \times (x_1^2 + x_2^2) - 0{,}48 \times x_1 \times x_2$	$-10 \le x_i \le 10$ $x_i = 0, i = 1, \dots, n$ f(x)=0
Rastrigin's function (F7)	$f(x) = 10 \times n + \sum_{i=1}^{n}[x_i^2 - 10 \times \cos(2\pi x_i)]$	$-5{,}12 \le x_i \le 5{,}12$ $x_i = 0, i = 1, \dots, n$ f(x)=0

Irodalomjegyzék

- BFOA: LIU Y. es PASSINO K. M.: Biomimicry of social foraging bacteria for distributed optimization: Models, principles, and emergent behaviors, Journal of Optimization Theory and Applications, 2002. pp. 603–628
- BATA: YANG X.-S.: A New Metaheuristic Bat-Inspired Algorithm, Nature Inspired Cooperative Strategies for Optimization (NISCO 2010), (Eds. J. R. Gonzalez et al.), Studies in Computational Intelligence, Springer Berlin, Springer, 2010. pp. 65-74
- BA: PHAM D. T., GHANBARZADEH A., KOC E., OTRI S., RAHIM S., AND ZAIDI M.: The bees algorithm. Technical report, Manufacturing Engineering Centre, Cardiff University, 2005.
- CS: YANG X. S.,DEB S.: Cuckoo search via L´evy flights, Proc. of World Congress on Nature & Biologically Inspired Computing (NaBIC 2009), December 2009, India. IEEE Publications, 2009. pp. 210-214.
- CA: REYNOLDS R. G.: An introduction to cultural algorithms. In Proceedings of the 3rd Annual Conference on Evolutionary Programming, World Scienfific Publishing, 1994. pp. 131–139
- DE: PRICE K., STORN R. M. ES LAMPINEN J. A.: Differential evolution: A practical approach to global optimization, Springer, 2005.
- FF: YANG X. S.: Firefly algorithms for multimodal optimization. Stochastic Algorithms: Foundations and Applications, SAGA 2009. Lecture Notes in Computer Sciences 5792. pp. 169-178.
- HS: GEEM Z. W., KIM J. H., AND LOGANATHAN G. V.: A new heuristic optimization algorithm: Harmony search. Simulation, 76:60–68, 2001.
- MA: MOSCATO P.: On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical report, California Institute of Technology, 1989.
- PSO: KENNEDY J. es EBERHART R. C.:. Particle swarm optimization, In Proceedings IEEE int'l conf. on neural networks Vol. IV, 1995. pp 1942–1948


Google Tudós
evolution algorithms
Cikkek
Nagyjából 3 340 000 találat (0,11 másodperc)
Saját profil
Saját k
Bármikor
2021 óta
2020 óta
2017 óta
Egyéni tartomány...
Rendezés relevancia szerint
Rendezés dátum szerint
szabadalmak is
idézetek megjelenítése
Értesítés létrehozása
Evolution algorithms in combinatorial optimization
H Mühlenbein, M Gorges-Schleuter, O Krämer - Parallel computing, 1988 - Elsevier
Evolution algorithms for combinatorial optimization have been proposed in the 70's. They did not have a major influence. With the availability of parallel computers, these algorithms will become more important. In this paper we discuss the dynamics of three different classes ...
Idézetek száma: 558 Kapcsolódó cikkek Mind a(z) 5 változat Web of Science: 198
Opposition-based differential evolution algorithms
[PDF] researchgate.net
S Rahnamayan, HR Tizhoosh... - 2006 IEEE International ..., 2006 - ieeexplore.ieee.org
Evolutionary Algorithms (EAs) are well-known optimization approaches to cope with non-linear, complex problems. These population-based algorithms, however, suffer from a general weakness; they are computationally expensive due to slow nature of the ...
Idézetek száma: 200 Kapcsolódó cikkek Mind a(z) 5 változat
[HTML] A numerical study of some modified differential evolution algorithms
[HTML] sciencedirect.com
P Kaelo, MM Ali - European journal of operational research, 2006 - Elsevier
Modifications in mutation and localization in acceptance rule are suggested to the differential evolution algorithm for global optimization. Numerical experiments indicate that the resulting algorithms are considerably better than the original differential evolution ...
Idézetek száma: 340 Kapcsolódó cikkek Mind a(z) 12 változat Web of Science: 198
Particle swarm optimization and differential evolution algorithms: technical analysis, applications and hybridization perspectives
[PDF] softcomputing.net
S Das, A Abraham, A Konar - Advances of computational intelligence in ..., 2008 - Springer
Since the beginning of the nineteenth century, a significant evolution in optimization theory has been noticed. Classical linear programming and traditional non-linear optimization techniques such as Lagrange's Multiplier, Bellman's principle and Pontyagrin's principle ...
Idézetek száma: 535 Kapcsolódó cikkek Mind a(z) 63 változat