SAPIENTIA
ERDÉLYI MAGYAR
TUDOMÁNYEGYETEM

TUDOMÁNYOS DIÁKKÖRI DOLGOZAT

TERMÉSZET ÁLTAL INSPIRÁLT OPTIMALIZÁCIÓS MEGOLDÁSOK

Szerző: **Kiss Gergely**

Számítástechnika MSc. szak, I. évf.

Konzulens: **dr. Lendák Imre**

egyetemi tanár

Temesvár, 2022

SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM

TERMÉSZET ÁLTAL INSPIRÁLT OPTIMALIZÁCIÓS MEGOLDÁSOK

NATURE INSPIRED OPTIMIZATION SOLUTIONS

Kiss Gergely

Konzulens:
dr. Lendák Imre

Kézirat lezárva: 2022. március 26.

Abstract

The aim of this study is to present optimization algorithms which were inspired by nature and to analyze their accuracy. Accuracy plays an important role in optimization solutions, so the main goal of these algorithms is to ensure efficient and relatively accurate results. The study presents the genetic algorithm and the particle-swarm algorithm as two prominent members of the evolutionary algorithms.

One of the main advantages of evolutionary algorithms is that they can optimize any criterion function even if it's mathematically impossible to get a result. The evolutionary algorithms' common features are their parameters: population number and the number of iterations. They influence the accuracy and the speed of the algorithm aswell. Areas of application of these algorithms include machine learning, artificial neural networks, big data analysis.

I used various tools to perform the analysis: for the genetic algorithm I used *Matlab*'s *Optimization Toolbox* package, on the other hand I used my own *Python* and *Javascript* program to evaluate the particle swarm optimization algorithm. I chose the Rastrigin function as the criterion function. Each experiment was performed 100x which ultimately calculated mean, median and standard deviation. I got comparable results after a few sets of these experiments where every set of calculations were carried out with different parameters. These analyzes have shown that these algorithms do not always give accurate results and it is always the best to make compromises in terms of speed and accuracy depending on the optimization problem.

Keywords: optimization, evolutionary algorithms, genetic algorithm, particle swarm optimization.

Kivonat

A dolgozat célja bemutatni olyan optimalizációs algoritmusokat, melyek létrejöttét a természet inspirálta, valamint kivizsgálni ezen algoritmusok pontosságát. Optimalizációs megoldásoknál nagy szerepet kap a pontosság, így ezek az algoritmusoknak is fő céljuk a hatékony és relatív pontos eredmény biztosítása. A dolgozat a genetikus algoritmust valamint a részecske-raj algoritmust mutatja be, mint az evolúciós algoritmusok két jeles tagját.

Az evolúciós algoritmusok fő előnyei közé tartozik, hogy bármilyen kritériumfüggvényt optimalizálni tudnak, így a matematikai úton lehetetlen eredményt is ki tudják számolni. Közös vonásaik a paramétereikben húzódnak, miszerint a populációszámuk és a ciklusszámuk befolyásolja az algoritmusok pontosságát és gyorsaságát elsősorban. Az evolúciós algoritmusok felhasználási területe többek között a gépi tanulás, mesterséges neuronhálózatok, big data analízis.

A kísérletek lebonyolításához különböző segédeszközöket használtam: a genetikus algoritmushoz a *Matlab* szoftvercsomag *Optimization Toolbox* nevű modulját, míg a részecske-raj optimalizációnál saját kezűleg írt *Python* illetve *Javascript* szoftvert használtam. Kritériumfüggvényként a Rastrigin-féle függvényt választottam. Kísérletenként 100 számítást végeztem, ami végeredményül középértéket, mediánt és szórást számolt ki. A különböző kísérleteket különböző paraméterek mellett ismételtem meg, így összehasonlítható eredményekre jutottam. Az elemzések kimutatták, hogy ezek az algoritmusok – báris a kezdetleges változatuk – nem minden esetben szolgálnak pontos eredménnyel, így kompromisszumos megoldásokat lehet csak ajánlani az algoritmus gyorsasága és pontossága terén.

Kulcsszavak: optimalizáció, evolúciós algoritmusok, genetikus algoritmus, részecske-raj optimalizáció.

Tartalomjegyzék

Ábrák jegyzéke

Táblázatok jegyzéke

1. fejezet

Bevezető

A modern kori mérnöki tudományokban egyre nagyobb teret hódítanak azok a megoldások, amelyek a természet jól bevállt működési elvét, strukturális felépítését imitálják. A biomimetika/bionika egy egész tudományág, ami ezekkel a természetbeli megoldások mérnöki megvalósításával foglalkozik.

A műszaki tudományokban lépten-nyomon szembetaláljuk magunkat optimalizációs problémákkal. Alapvető fizikai törvényszerűségeket is fel lehet állítani, mint optimalizációs probléma: például az energia megmaradási törvénye. Teljesen jogos tehát azt állítani, hogy a természet végzi a legtöbb optimalizációt.

Ha már a természet ilyen jól tud optimalizálni, feltétlenül léteznie kell olyan megoldásoknak, amelyek követnek valamiféle természetes folyamatokat. Ebben a dolgozatban két ilyen megoldás kerül bemutatásra, amely egyszerűen és hatékonyan oldja meg az optimalizációs problémákat: a genetikus algoritmus és a részecske-raj optimalizáció. Ezek az algoritmusok sztochasztikus folyamatok[1] , amelyek az evolúciós algoritmusok családjába tartoznak.

Ha a természet olyan összetett folyamatot, mint az optimalizáció képes lebonyolítani, akkor az ember csakis természetes folyamatok által tudja ezt a feladatot leghatékonyabban elvégezni. A fent említett algoritmusok bemutatása és elemzése előtt viszont érdemes megismerkedni az optimalizáció matematikai definíciójával.

[1] A sztochasztikus folyamat, vagy más néven véletlenszerű folyamat, az a folyamat, melyet – részben vagy teljesen – valószínűségi változók jellemeznek. Ennek az ellentéte a determinisztikus folyamat, ahol a folyamatot leíró változók nem véletlenszerűek (Wikipedia – A szabad enciklopédia 2020)

2. fejezet

Matematikai optimalizáció

Ebben a fejezetben a matematikai optimalizáció elméleti háttere kerül bemutatásra.

2.1. Matematikai optimalizáció definíciója

Matematikai optimalizáció alatt olyan folyamatot értünk, amely egy előre meghatározott kritérium alapján keresi a legjobb megoldást (Vujanović 1980). Ezt az eredményt hívjuk optimális értéknek. Az optimalizáció kritériumfüggvényét[1] a 2.1 képlettel ábrázoljuk.

$$\begin{aligned} f &: X \to Y, Y \subseteq R \\ f &= f(x) \\ x &= x_1, x_2, ..., x_n \end{aligned} \tag{2.1}$$

Az optimalizáció valós esetben a 2.1 függvény minimális vagy maximális értékének a meghatározását jelenti, vagyis minimum esetében a 2.2 képlet, maximum esetében a 2.3 képlet érvényes.

$$f(x^*) \leq f(x), \forall x \in X \tag{2.2}$$

$$f(x^*) \geq f(x), \forall x \in X \tag{2.3}$$

[1] Kritériumfüggvény. Szinonímái: költségfüggvény, energiafüggvény, hasznossági függvény.

Mivel a fenti esetekben a függvény legkisebb (illetve legnagyobb) értékét az egész értelmezési tartomány szintjén az x^* adja meg, ezért ezt globális optimalizációnak[2] nevezzük.

A kritériumfüggvény mellett más egyenletek (2.4 képlet) is szerepelhetnek a probléma meghatározásában.

$$g_k(x) = 0 \tag{2.4}$$

Ilyenkor korlátos vagy megkötött matematikai optimalizációról beszélünk. A továbbiakban csak megkötés nélküli optimalizációról lesz szó.

2.2. Optimalizációs módszerek

Sokféle analitikus módszer segítségével lehet eljutni a kritériumfüggvény optimális értékéhez. Többségük a függvény többszörös differenciálhatóságát követeli meg, így összetettebb esetekben ezeket a módszereket nem tudjuk alkalmazni. Ahogy nő a probléma dimenzionalitása, úgy válik nehezebbé az analitikus eredmény kiszámítása (ha egyáltalán lehetséges az analitikus eredményig eljutni). Ilyenkor az egyedüli megoldást a numerikus módszerek jelentik.

Tegyük fel, hogy a számítógép segítségét kérjük ennek a problémának a megoldásában. Milyen módszerhez kell folyamodnunk, hogy a várt eredményt kapjuk meg? Legrosszabb esetben végigfutnánk az összes elemen, és így megtalálnánk a keresett legjobb értéket. Ez egyrészt nem kivitelezhető, mert a függvényeink általában folytonos térben mozognak, így végtelen számú elemet kellene leellenőrizni. Ha mégis valahogy felosztanánk a függvény dómenjét N részre, az rettenetesen hosszú számítási időt venne igénybe. Egyedüli megoldást a numerikus algoritmusok kínálnak (Rapaić 2019).

[2]Globális optimalizáció kiterjed az egész értelmezési tartomány szintjére. Lokális optimalizáció esetében csak az x^* közelében vizsgáljuk a függvényt: $\| x^* - x \| < \varepsilon$

3. fejezet

Evolúciós algoritmusok

A numerikus módszerek előnye, hogy viszonylag gyorsan, bármilyen függvény esetében tudnak eredménnyel szolgálni, nincsenek különleges követelményeik, mint az analitikus módszerek esetében.

A számítástechnikai irodalomban nagyszámú sztochasztikus optimalizációs algoritmus létezik, egyik csoportjuk ezeknek az evolúciós algoritmusok. Ezeknek az algoritmusoknak az a különlegessége, hogy véletlenszerű változókat tartalmaznak, így két egymás utáni, egyforma feltételek mellett lezajlott számítás nem fog pontosan egyforma eredményt adni.

Az evolúciós algoritmusok, mint ahogy az a nevükben is szerepel, a természetes evolúció működését próbálják imitálni. Közös vonásuk, hogy egyidőben több lehetséges eredménnyel számolnak, ezeknek a halmaza alkotja a *populációt* . Ez a populáció ciklusról-ciklusra változásokon megy keresztül úgy, hogy minden ciklusban egyre közelebb kerüljön az optimális értékhez.

Ebben a dolgozatban az evolúciós algoritmusok közül két képviselőjük kerül bemutatásra: a genetikus algoritmusok és a részecske-raj algoritmus.

4. fejezet

Genetikus algoritmus

Az élővilág egyik alappillére a szaporodás, ami biztosítja a fajok túlélését hosszú távon. Az evolúciós törvényeket betartva azok a fajok maradnak fenn legtovább, akik jobban tudtak alkalmazkodni a környezetükhöz; a vadonban a legnagyobb, legerősebb, legtalálékonyabb fajok vannak előnyben, így szaporodáskor is nagyobb esélyük van pont ezeket a géneket továbbadni. Minél jobb génekkel rendelkezik egy élőlény, annál életrevalóbb utódjai lesznek, és ezzel növelik az egész faj esélyét a túlélésre. R. A. Fisher[1] még az 1950-es években fejlesztette ki azt a matematikai modellt, amelyet a mai formába John Henry Holland[2] ültetett át, és manapság genetikus algoritmusnak hívunk (Holland 2012).

4.1. A genetikus algoritmus alapjai

A genetikus algoritmus populációját olyan egyedek (ún. kromoszómák) alkotják, amelyek géneket hordoznak magukban. A gének száma megegyezik az optimalizációs probléma dimenzionalitásával, vagyis általános esetben n darab génnel rendelkezik az egyed. Minden kromoszóma a keresési mezőben potenciális megoldás lehet, így mindre ki tudjuk számolni a kritériumfüggvény értékét, ami az egyed alkalmasságát jelenti. A genetikus algoritmus célja, hogy az egyedek génjeit úgy változtatja meg, hogy azok alkalmassági értékei a lehető legjobbak legyenek, így megtalálva az optimális értéket. A kromoszómák génjeit *genetikus műveletek* segítségével lehet változtatni.

4.2. A genetikus műveletek

Ebben a részben különféle genetikus műveleteket mutatunk be.

[1] Ronald Aylmer Fisher (1890–1962) brit statisztikus, genetikus

[2] John Henry Holland (1929–2015) amerikai számítástechnika és pszichológia professzor

4.2.1. Kiválasztás

Általános esetben a populációt N darab kromoszóma alkotja, melyeknek az alkalmassági értékein kell javítani. A genetikus elméletet követve a legjobb génekkel rendelkező egyedeknek kell utódokat létrehozniuk annak érdekében, hogy az egész populáció génállományát erősítsék. Tehát minden új ciklusban (generációban) meg kell ismételni a genetikus műveleteket. Egyszerre két egyedet kell kijelölni úgy, hogy a kiválasztás valószínűsége egyenesen arányos legyen az egyed alkalmasságával.

Egy efféle algoritmus a *rulettkerék kiválasztás* , ami minden egyednek biztosít egy mezőt a keréken, de a mező szélessége egyenesen arányos az egyedhez rendelt kiválasztási valószínűséggel. Első gondolatra ez nem is tűnik rossz megoldásnak, másrészt viszont olyan mellékhatás léphet fel, ami szuper-egyedeket generál: így nem tudjuk biztosítani a globálisan elérhető optimumot, ha a szuper-egyed egy lokális optimumban akad meg. Ez a szuper-egyed később magához vonza a többi egyedet is, így a globális optimumot mind elkerüli (Kanović, Jeličić és Rapaić 2017).

Ahhoz, hogy ne favorizáljunk túl bizonyos géneket, rang alapú rendszert kell bevezetni, ahol az egyedek az alkalmassági értékük alapján vannak sorrendbe állítva, és 1-től N-ig terjedő számmal vannak megjelölve. Tehát egy nagyon jó génekkel ellátott egyed és egy kevésbé jó géneket hordozó egyed között sokkal kisebb ez esetben a különbség.

4.2.2. Keresztezés

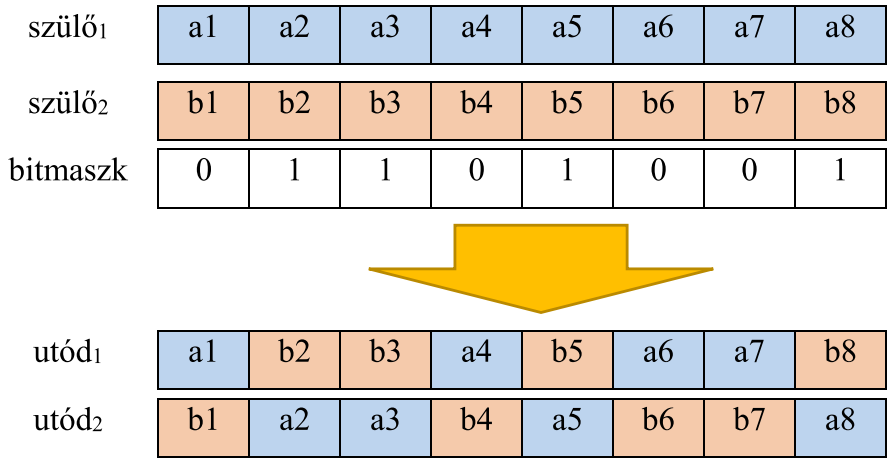
A kiválasztott egyedek képezik azt a halmazt, amiből majd kikerülnek a jövendőbeli szülők. Az utódok keresztezés útján jönnek létre, és így örökölnek bizonyos tulajdonságokat a felmenőiktől. Ez a művelet változtatja a populáció helyzetét, vagyis egyenes úton manipulálja a kromoszómák génszerkezetét.

A keresztezést többféle módon lehet kivitelezni:

- binárisan kódolt gének uniform keresztezése (4-1 ábra)

  Két szülőtől két utód jön létre olyan módon, hogy egy véletlenszerűen generált bit-maszkot használva keverjük össze a szülők génjeit. (Az első utód az első szülőtől kap egy gént, ha a bit-maszkban 0 szerepel, ha pedig 1-es, akkor a második szülőtől. A második utód pedig ellenkező logikával kapja meg a génjeit).

- reális számokkal kódolt gének aritmetikai keresztezése



4-1. ábra. Bináris uniform keresztezés (Kanović, Jeličić és Rapaić 2017)

A reális számoknál is hasonló keresztezés által születnek meg az utódok. A szülők (x_1 és x_2) génjeinek lineáris kombinációja alkotja az utódok génjeit (4.1 képlet).

$$\begin{aligned} x_1^0 &= a \cdot x_1 + (1-a) \cdot x_2 \\ x_2^0 &= a \cdot x_2 + (1-a) \cdot x_1 \end{aligned} \tag{4.1}$$

Ebben az esetben az a paraméter egy véletlenszerű szám 0 és 1 között, ami megfelel az előző példában szereplő bit-maszknak.

4.2.3. Mutáció

Mint ahogy az a természetes evolúcióban is jelentkezik, a gének időszakonként mutálódnak, így egy olyan populáció jön létre, amely nagyon kis mértékben eltér az előző generációtól. Ez azért fontos, mert ezek a kis mutációk jelentik azt a génbeli sokszínűséget, ami ellenállóbbá teszi az egyedeket (pl. kórokozók ellen).

A genetikus algoritmusban azért van szükség mutációra, hogy ne részesítsünk előnyben bizonyos egyedeket, amik így nem válnak szuper-egyedekké. Habár ezt a genetikus műveletet elenyésző számú egyeden végzik el, mégis gyorsabb konvergenciót eredményez, mintha mutáció nélkül zajlott volna az algoritmus (Kanović, Jeličić és Rapaić 2017).

Valós számok esetében a mutáció a következőképp végezhető el:

- a génhez tartozó valós számot egy véletlenszerű valós számmal váltjuk fel (legdrasztikusabb hatást vált ki),
- nullához közeli valós szám hozzáadása vagy kivonása, vagy

- egyhez közeli valós számmal való szorzás.

Miután befejeztük a genetikus műveleteket, el kell dönteni, hogy mely egyedek kerülnek be a következő generációba. A legegyszerűbb módszer az újonann kapott egyedeket átmenti a következő generációba, míg az összes szülőt kitörli a keresési mezőből. Ezzel az a gond, hogy egyes rendkívül jó génekkel rendelkező szülő le lesz cserélve egy estleges kevésbé jó utódra. Az ilyen probléma kiküszöbölésére alkalmazzák az *elitizmus* elvét, miszerint bizonyos kritériumoknak megfelelő szülő át lesz mentve a következő generációra és potenciálisan lehet még utódja. Ez segíti az algoritmus konvergencióját, de elővigyázatosan kell kiválasztani azt a bizonyos elit kritériumot; különben létrejön egy *szuper-egyed* , ami nem minden esetben a globális optimumot mutatja (Kanović, Jeličić és Rapaić 2017).

4.3. A genetikus algoritmus elemzése

A genetikus algoritmus empírikus analíziseként a szakirodalom tesztfüggvények elemzését ajánlja (Kanović, Jeličić és Rapaić 2017). Ebben a dolgozatban bemutató jelleggel csak egy ún. Rastrigin-féle tesztfüggvény elemzése kerül elemzésre.

A Rastrigin-féle függvény érdekessége, hogy sok lokális minimuma van, és a 4.2 képlettel van definiálva. A függvény háromdimenziós grafikonja a 4-2 ábrán megtekinthető, valamint a függvény x-y síkra tett vetülete (4-3 ábra).

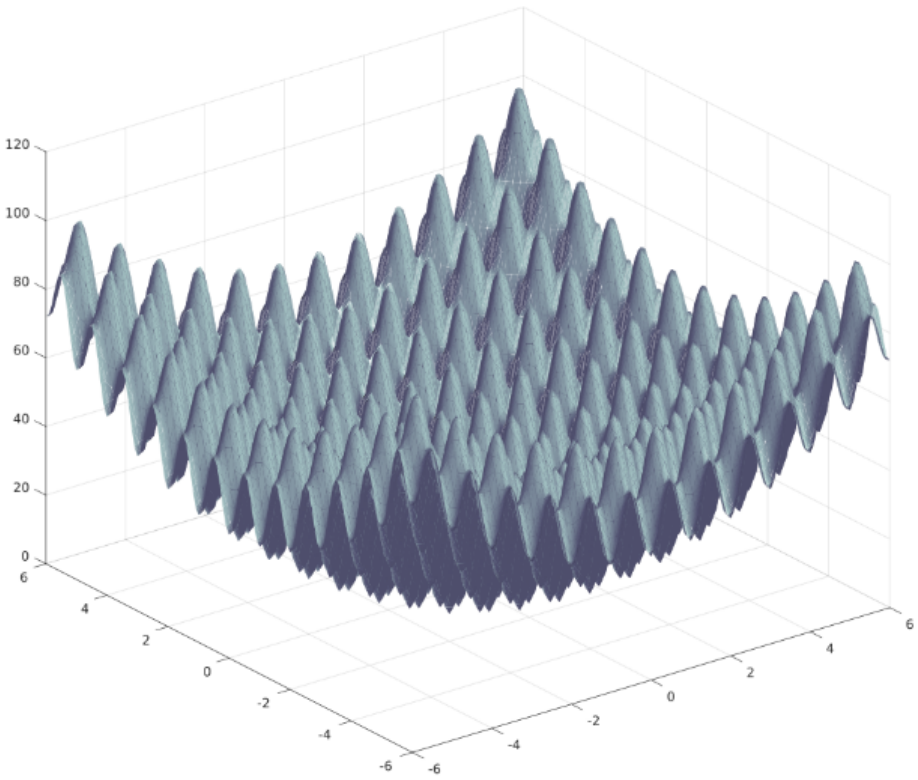
$$f(x) = 10n + \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i)) \tag{4.2}$$

Általános esetben a függvény optimális értéke: $f(x^*) = 0$, $x^* = [0, 0, ..., 0]$, míg kétdimenziós probléma esetében ($n = 2$): $f(x^*) = 0$, $x^* = [0, 0]$.

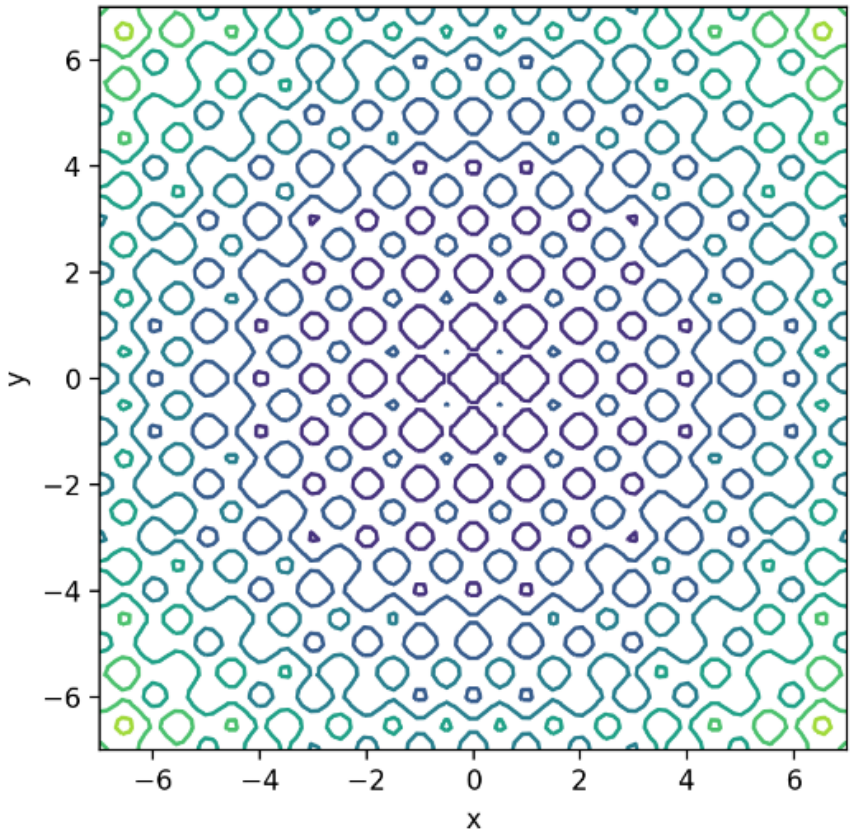
A számításokat a *Matlab* szoftvercsomag *Optimization Toolbox* (Mathworks 2016) moduljával végeztem. Négy különböző környezetben zajlott a számítás, más-más paraméterek mellett (generációszám és populációszám).

```
function v = rastrigin(x)
  A = 10;
  s = sum(x.*x - A * cos(2*pi*x));
  v = A * length(x) + s;
end
```

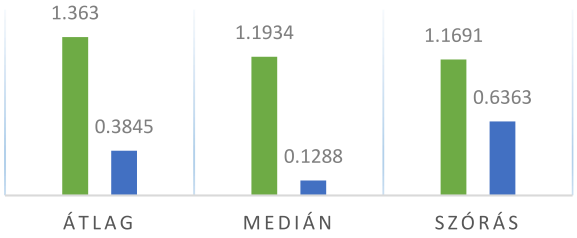
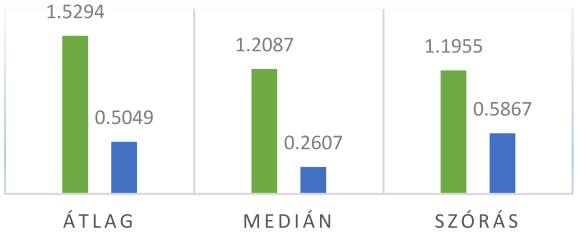
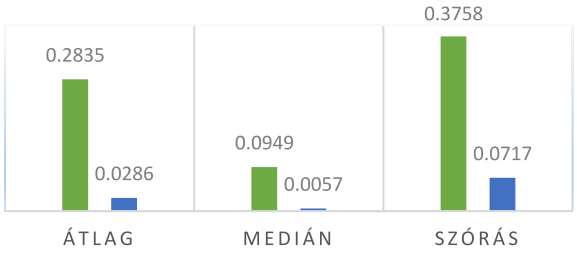
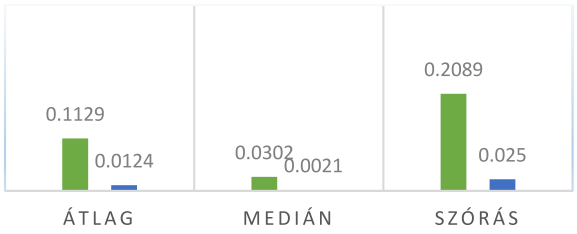
4.1. Listing. A Rastrigin-függvény kódja



4-2. ábra. Rastrigin-függvény grafikonja



4-3. ábra. Rastrigin-függvény x-y vetülete

#	populációszám	generációszám	eredmények
K1	25	200	
K2	25	400	
K3	100	200	
K4	200	200	

4-1. táblázat. A Rastrigin-függvény elemzésének az eredményei

A kísérletet százszoros megismétlése után születtek meg az eredmények, melyek a 4-1 táblázatban találhatók. Mindemellett az egyedek kezdőpozíciói elrendezettsége kétféle módon lett meghatározva: uniform elrendezettség (zöld), nem uniform elrendezettség (kék).

Az eredmények alapján megállapíthatjuk, hogy a generáció számának a növelésével nem javult az eredményünk pontossága, míg a populáció számának a növelése nagyban hozzájárult a jobb eredményhez. A genetikus algoritmusnál a populáció számának a növelése drasztikusan lassítja az optimalizációs folyamatot, így a probléma sajátosságaihoz képest kell eldönteni, hogy gyors és kevésbé pontos eredményt kapunk, vagy lassú de pontosabb eredményt.

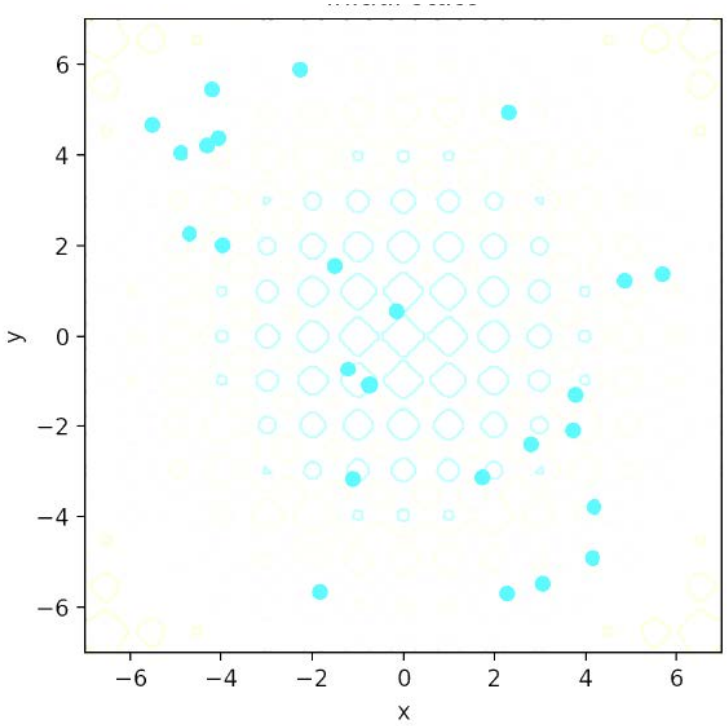
A legérdekesebb eredmények viszont az egyedek kezdőpozícióinak a változtatásakor

születtek. Minden esetben jobb eredményt ért el az algoritmus, amikor a keresési mezőn nem egyenletesen (4-4 ábra), hanem tömbben (4-5 ábra) helyezte el az egyedeket.

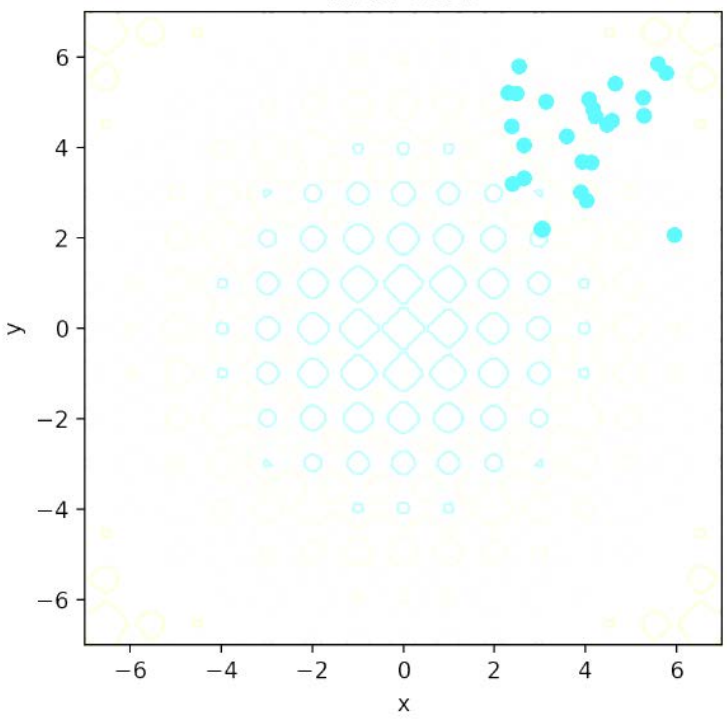
Az eredmények függvényében megállapítható, hogy a genetikai algoritmus jobban teljesít, ha kevesebb egyed nagyobb koncentrációban jelenik meg a keresési mezőn. A konkrét problémára levetítve ez jobb irányelvnek bizonyult, de más problémák esetében javasolt a paraméterek és a kezdőpozíció finomhangolása.

4.4. A genetikus algoritmus felhasználási területei

A genetikus algoritmus egy megbízható és állóképes algoritmus, mint amilyen a természetes evólúció is, amelyről modellezték. Rendkívűl széleskörű a használata: a matematikától kezdve, az egészségügyön át, a különböző mérnöki szakirányzatokig. A genetikus műveletek többféle módon való értelmezése különböző előnyöket ad a genetikus algoritmus felhasználásának más és más szakterületen: kombinatorikus problémák megoldása, képfeldolgozás, mesterséges neuronhálózatok és más gépi tanulási technikák (Kanović, Jeličić és Rapaić 2017).



4-4. ábra. Részecskék uniform kezdőpozíciói



4-5. ábra. Részecskék nem-uniform kezdőpozíciói

5. fejezet

Részecske-raj optimalizáció

Ebben a fejezetben a részecske-raj optimalizáció kerül bemutatásra.

5.1. Rajintelligencia

Az állatok alapvető túlélési ösztönei közé tartozik az élelem és élőhely utáni keresés. Ez minden állatfajra igaz, a hangyától a bálnáig, mégis a csoportokban élő állatfajok különösen egymásra vannak utalva ilyen téren. Az oka ennek, hogy külön-külön valószínűleg elpusztulnának, de csoportokban sokkal erősebbek és védettebbek lesznek. Ezt a jelenséget nevezik *rajintelligenciának* , miszerint a nem-intelligens egyedek intelligens viselkedésmintákat mutatnak csoportokban, ami végül sikeres túlélési képességet eredményez (Rapaić 2019).

1995-ben James Kennedy amerikai pszichológus és Russel C. Eberhart mérnök pontosan definiálták ezeket a viselkedésmintákat, és így született meg a részecske-raj optimalizáció (Dorigo és Birattari 2007).

5.2. A részecske-raj optimalizáció alapjai

A részecske-raj optimalizációs algoritmus populációja olyan egyedekből áll, amelyek állandó mozgásban tartják az egész rajt. Ez a mozgás nagyban függ az egyed saját tapasztalataitól, vagyis az eddigi bejárt úttól. Az így megszerzett információk alapján az egyedek egymással kommunikálnak, és a lehető legjobb eredmény felé mozdulnak el. Ez a mozgás ciklusonként frissül, ami a gyakorlatban annyit jelent, hogy lépésenként változik a raj helyzete. A genetikus algoritmustól eltérően a populáció tagjai nem cserélődnek, mindig ugyanazok az egyedek szerepelnek a rajban, csak a helyzetük változik.

Ha a sorrendben n-edik ciklus keresztmetszetét vesszük, minden részecske a következő információkat hordozza magával:

- a részecske jelenlegi helyzete – $x[n]$: a keresési tartomány egyik értéke, dimenzionalitása függ a kritériumfüggvény dimenziójától;
- jelenlegi sebesség – $v[n]$: a részecske előző lépésből szerzett sebessége (az előző helyzettől való elmozdulása);
- a részecske eddigi legjobb helyzete – $p[n]$: a részecskének az a pozíciója, amelyet eddigi útja során felvett, és a legjobb értéket hordozza.

A fent említett paraméterek mellett fontos még a raj szintjén történő információcsere, vagyis a raj szintjén elért eddigi legjobb helyzet – $g[n]$. Ez a paraméter a valaha elért legjobb részecske helyzetét jelenti.

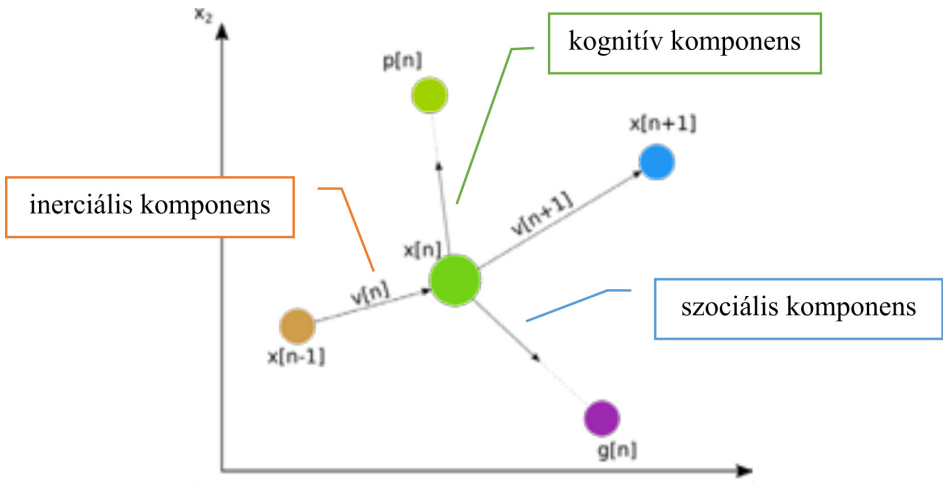
A részecskék térben és időben történő változását a 5.1 és a 5.2 egyenletek határozzák meg.

$$v[n+1] = w \cdot v[n] + cp \cdot rp[n] \cdot (p[n] - x[n]) + cg \cdot rg[n] \cdot (g[n] - x[n]) \tag{5.1}$$

$$x[n+1] = x[n] + v[n+1] \tag{5.2}$$

Az $rp[n]$ és az $rg[n]$ egymástól független véletlenszerű számok 0 és 1 között, míg a w, cp és cg együtthatók alkotják az algoritmus komponenseit:

- inerciális komponens – $w \cdot v[n]$: az előző lépésben kiszámolt sebességet befolyásoló w együttható egy részecske szabad mozgását szimulálja, vagyis nagyobb teret ad a részecske szabad mozgásának;
- kognitív komponens – $cp \cdot rp[n] \cdot (p[n] - x[n])$: cp gyorsulási együttható meghatározza, hogy egy részecske eddigi tapasztalatai mennyire befolyásolják a részecske következő helyzetét;
- szociális komponens – $cg \cdot rg[n] \cdot (g[n] - x[n])$: cg együttható a részecskék együttműködési hajlandóságát határozza meg, vagyis a raj szintjén jelentkező legjobb értékhez történő közelítést biztosítja.



5-1. ábra. A részecske-raj optimalizáció komponensei (Kanović, Jeličić és Rapaić 2017)

együttható	kezdő érték	végső érték
w	0.9	0.4
cp	2.5	0.5
cg	0.5	2.5

5-1. táblázat. Az időben változó együtthatók értéke (Kanović, Jeličić és Rapaić 2017)

Az algoritmus komponenseinek a kölcsönhatásait a 5-1 ábra vizualizálja. Az egyes komponensek súlyozott értékei (w, cp és cg) tetszőlegesen választhatóak, ebből eredendően több változata létezik a részecske-raj algoritmusnak. A dolgozatban bemutatott változatban ezek az együtthatók időben változó értékek, melyeknek az értékei a 5-1 táblázatban találhatóak meg.

5.3. A részecske-raj algoritmus

Első lépésként a raj nagyságát kell meghatározni, ami becslések alapján nagyjából tízszerese kell, hogy legyen a probléma dimenziójától (Rapaić 2019). Erre azért van szükség, mert túl kevés egyed képtelen bejárni az egész keresési mező összes dimenzióját és információ hiányában nem kerül felderítésre lehetséges megoldások.

Az algoritmus elején minden részecskéhez véletlenszerűen egy kezdő pozíciót kell rendelni ($x[0]$), ami egyben az egyedek eddigi legjobb pozícióját is jelenti ($p[0]$). A kritériumfüggvény alapján kiszámoljuk melyik pozíció a legjobb, ez lesz a globális legjobb pozíció ($g[0]$). Ezzel lezárult az inicializációs rész.

A fő részben minden részecskének ki kell számolni a következő sebességét – 5.1 képlet, és ez alapján a pozícióját – 5.2 képlet. A részecske eddigi legjobb helyzetét is meg kell változtatni, ha az újabb pozícióra jobb eredmény születik. Miután bejeződött a ciklus, a globális pozíciót kell frissíteni úgy, hogy a részecskék eddigi legjobb pozícióinak az értékét hasonlítjuk össze a globálisan legjobb értékkel.

```
begin pso-algoritmus
   raj inicializáció
   while (leállási feltétel nem teljesített)
   do
      for (i=1 to részecskék száma)
         sebesség és az új pozíció kiszámolása
         kritériumfüggvény értékének meghatározása
         részecske eddigi legjobb pozíciójának a frissítése
      end
      globálisan legjobb pozíció frissítése
   end do
end pso-algoritmus
```

5.1. Listing. A részecske-raj algoritmus pszeudokódja (Dorigo, A. Montes de Oca és Engelbrecht 2008)

Leállási feltételként leggyakrabban egy előre megadott maximális ciklusszámot használnak, de ez függ a probléma típusától. Egy másik megoldás lehet a globálisan legjobb pozícióra kapott érték változását követni, miszerint ha egy megadott tűréshatár alá esik az érték változása, akkor teljesül a leállási feltétel (Kanović, Jeličić és Rapaić 2017).

5.4. A részecske-raj algoritmus kivizsgálása

A genetikus algoritmushoz hasonlóan, a részecske-raj algoritmus is demonstratív módon a Rastrigin-féle tesztfüggvény 4.2 által kerül elemzésre.

A fent leírt algoritmus alapján és a megfelelő paraméterek segítségével egy sajátkezűleg írt programot készítettem *Python* programnyelvben (Kiss 2020b). A vektorszámításokat a *NumPy* könyvtár segítségével oldottam meg, míg az eredmények grafikonon történő megjelenítéséhez a *matplotlib* könyvtárat használtam. A dolgozatban szereplő Rastrigin-függvény és minden egyéb grafikonos ábra szintén ezzel a szoftverrel készült el.

Az algoritmus elemzése négyféle környezetben zajlott, és mindegyik kísérlet 100-szoros ismétléssel történt. Az eredmények a 5-2 táblázatban találhatók meg.

Az eredményekből egyértelműen látható, hogy az egyenletesen elhelyezkedő populáció esetében a ciklusszám növelése 100-ról 1000-re nem hatott ki az eredmény pontosságára, míg a nem egyenletesen elhelyezkedő populáció esetében a populáció számának a növelése

#	populációszám	ciklusszám	elhelyezés	átlageredmény	medián	szórás
K1	25	100	uniform	0.3283	$8.7512 \cdot 10^{-12}$	0.4678
K2	25	1000	uniform	0.1194	0	0.3233
K3	25	100	nem-uniform	0.7960	0.9950	0.9850
K4	40	100	nem-uniform	0.3781	$1.5721 \cdot 10^{-13}$)	0.6098

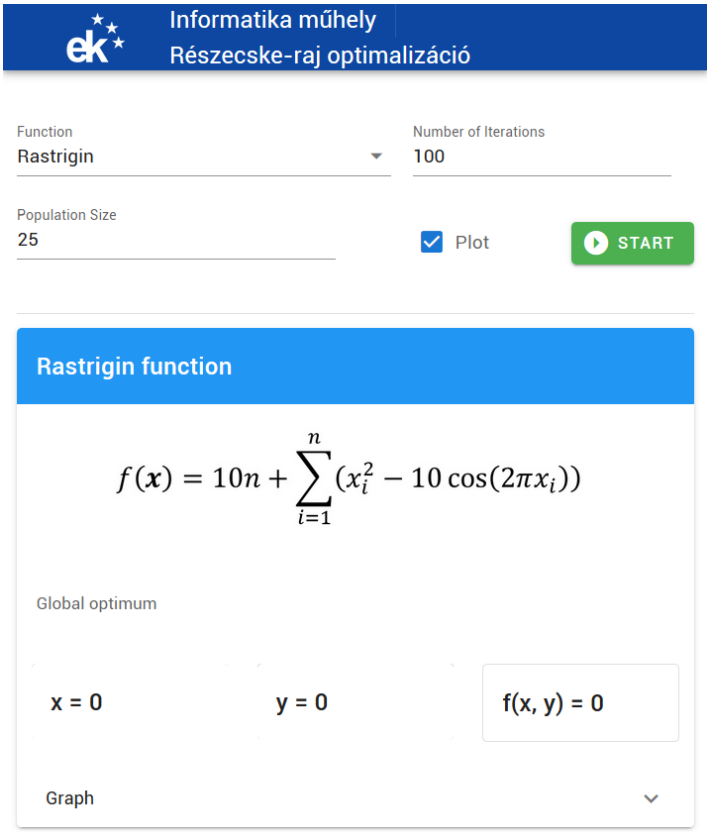
5-2. táblázat. A részecske-raj optimalizáció elemzésének az eredményei

nagyban javított az eredményen. Ez annak köszönhető, hogy az algoritmus komponenseinek az együtthatói időben változnak, így az idő múlásával csökken az egyedek felderítési hajlama, míg a szociális komponens erősödik. Ez azzal jár, hogy a kísérlet végére az egyedek egyre jobban hagyatkoznak az eddigi globálisan elért legjobb eredményre; így ha nem járták be a keresési mezőt, nagy valószínűséggel a legjobb eredmény egy lokális minimumban akad meg. A populáció számának a növelése esélyt ad a keresési mező jobb felderítésére, így megtalálva az optimális értéket.

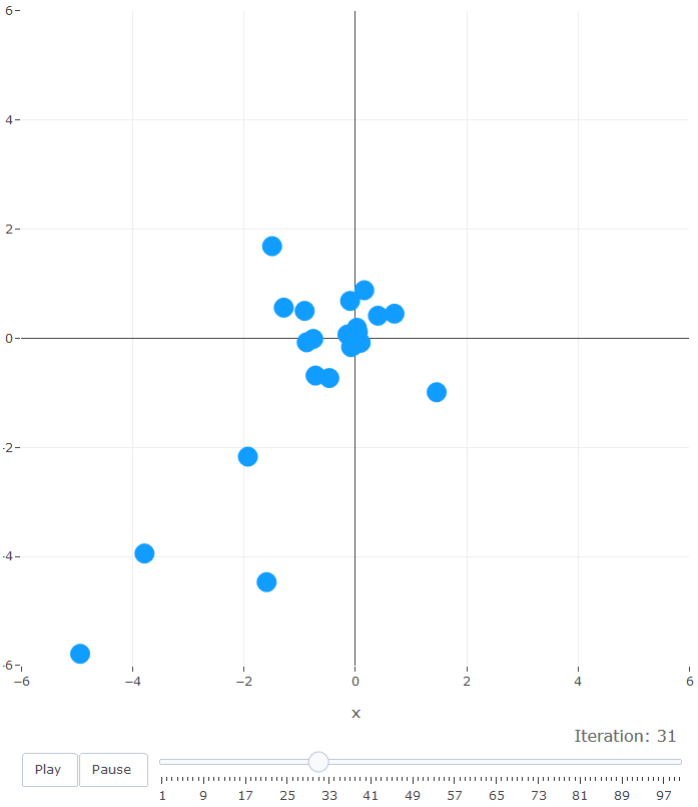
Az algoritmus vizualizációja érdekében készítettem egy *Javascript* programot, amely bármilyen böngészőből lefuttatható (Kiss 2020a). A Rastrigin-féle függvény mellett még egyéb más tesztfüggvények is elérhetőek, melyekre (a kiválasztott paraméterek mellett) ki lehet számolni az optimális értéket (5-2 ábra). A számolás befejeztével megjelenik egy időcsúszkával ellátott grafikon (5-3 ábra), ahol végig lehet követni a populáció helyzetét a generációk folyamán. A raj animált mozgásának köszönve szemügyre vehető egy-egy részecske vándorlása valamint az egész populáció szintjén történő együttműködés, viselkedésminta.

5.5. A részecske-raj algoritmus felhasználási területei

Az algoritmus egyszerűségéből fakadóan sok szakterületen megállta a helyét, így pl. az egészségügyben leukémia-típusú problémák diagnózisában. Közgazdasági területen a kockázati befektetések elemzésére használják. Jelentkezik mechanikában, termodinamikában, valamint egyéb geometriai optimalizációs problémáknál is (pl. szenzorok optimális elhelyezése). Az optimális vezérlés, rendszermodellezés, károk észlelése és még sok más gyakorlati problémához kínál megoldást a részecske-raj algoritmus (Seixas Gomes de Almeida és Coppo Leite 2019).



5-2. ábra. Paraméterek beállítása



5-3. ábra. A részecskék időbeli mozgása

A gyakorlati problémák mellett elméleti és más absztrakt témához is segítséget nyújt. A mesterséges intelligencia és a gépi tanulás kiképzési folyamatában valamint a mesterséges neuronhálózat paramétereinek optimális értékének a megkeresésében is használják a részecske-raj algoritmust (Kanović, Jeličić és Rapaić 2017).

6. fejezet

Következtetés

A numerikus megoldásokkal szemben az evolúciós algoritmusok egyszerű és intuitív módon állnak hozzá az optimalizációs problémákhoz. Az egyedi problémák viszont egyedi megoldásokat is igényelnek; ilyenkor következetes módon kell kiválasztani a számunkra felkínált eszközöket, amik nem utolsó sorban a természet által inspirált optimalizációs megoldások is lehetnek. Az elemzések során megfigyelhettük, hogy ezek az algoritmusok – báris a kezdetleges változatuk – nem minden esetben szolgált pontos eredménnyel. Manapság is aktív kutatási terület ezen algoritmusok további fejlesztése, ami még sok újdonságot hoz a mérnöki feladatok megoldásaiban.

Irodalomjegyzék

Dorigo, Marco, Marco A. Montes de Oca és Andries Engelbrecht (2008). "Particle swarm optimization". *Scholarpedia*. DOI: 10.4249/scholarpedia.1486.

Dorigo, Marco és Mauro Birattari (2007). "Swarm intelligence". *Scholarpedia*. DOI: 10.4249/scholarpedia.1462.

Holland, John H. (2012). "Genetic algorithms". *Scholarpedia*. DOI: 10.4249/scholarpedia.1482.

Kanović, Željko, Zoran Jeličić és Milan Rapaić (2017). *Evolutivni optimizacioni algoritmi u inženjerskoj praksi*. Újvidék: Műszaki Tudományok Kara, Újvidéki Egyetem.

Kiss, Gergely (2020a). *Részecske-raj optimalizáció JavaScript algoritmus*. URL: https://gregvader.github.io/pso-algoritmus-js/ (elérés dátuma 2020. 11. 01.).

— (2020b). *Részecske-raj optimalizáció Python algoritmus*. URL: https://github.com/gregVader/pso-algoritmus (elérés dátuma 2020. 11. 01.).

Mathworks (2016). *Matlab 2016 - Optimization Toolbox*.

Rapaić, Milan (2019). "Evolúciós algoritmusok". jegyzetek az előadásról. Újvidék.

Seixas Gomes de Almeida, Bruno és Victor Coppo Leite (2019). "Particle Swarm Optimization: A Powerful Technique for Solving Engineering Problems". *Swarm Intelligence - Recent Advances, New Perspectives and Applications*. DOI: 10.5772/intechopen.89633.

Vujanović, B. (1980). *Metodi optimizacije*. Újvidék: Műszaki Tudományok Kara, Újvidéki Egyetem.

Wikipedia – A szabad enciklopédia (2020). *Sztochasztikus folyamat*. URL: https://hu.wikipedia.org/wiki/Sztochasztikus_folyamat (elérés dátuma 2020. 10. 10.).

Tárgymutató