SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Naslov

Velimir Kovačić
Voditelj: *Marin Golub*

Zagreb, prosinac 2023.

SADRŽAJ

1. Uvod

2. Algoritam krijesnica

3. Algoritam majmuna

3.1. Uvod

Algoritam majmuna[9] metaheuristički je optimizacijski algoritam utemeljen na načinu na koji se majmuni u prirodi penju na planine. Razvijen je za učinkovitu globalnu optimizaciju multimodalnih optimizacijskih problema s kontinuiranim varijablama u visokim dimenzijama.

Funkcija cilja optimizacijskog problema može se zamisliti kao polje s planinama. Kako bi majmun pronašao najviši vrh u polju, izvodi 3 radnje:

1. Penjanje
2. Gledanje i skakanje
3. Salto

Majmun se sa svoje početne točke penje uz planinu (Penjanje). U jednom trenutku staje, gleda oko sebe kako bi našao višu planinu i ako takva postoji, skače na nju (Gledanje i skakanje). Ponovno se penje uz planinu (Penjanje). Kako bi pronašli još višu planinu, mogu napraviti salto u novu domenu pretraživanja (Salto). Nakon što cijela populacija majmuna izvrši ovaj postupak dovoljno puta, dobije se izlaz algoritma koji je najviša točka koju su majmuni posjetili.

U multimodalnim optimizacijskim problemima, broj lokalnih optimuma raste eksponencijalno s povećanjem dimenzije. Cilj salta u algoritmu je izbjegavanje prekomjernog lokalnog pretraživanja. Penjanje, koje ustvari je lokalno pretraživanje, koristi pseudogradijent koji se uvijek računa na temelju vrijednosti funkcije cilja za 2 točke (neovisno o dimenziji). To omogućava učinkovitiju optimizaciju u optimizacijskih problemima u visokim dimenzijama.

3.2. Algoritam

Algoritam prati opisani iterativni postupak penjanja majmuna dok se ne dosegne određeni broj iteracija N. Izlaz je položaj za koji je zabilježena najveća vrijednost funkcije cilja.

Algorithm 1 Algoritam majmuna

```
1: function MA(M, n, N, Nc, a, b, [c, d])
2:     x = POSTAVLJANJE(M, n)
3:     x* = x[0]
4:     for iter = 1 to N do
5:         PENJANJE(Nc, x, x*, a)
6:         GLEDANJESKAKANJE(x, b)
7:         PENJANJE(Nc, x, x*, a)
8:         SALTO(x, [c, d])
9:     return x*
```

3.2.1. Postavljanje početnih položaja

Položaj majmuna bit će predstavljen n-dimenzionalnim vektorom:

$$\vec{x} = (x_1, x_2, \ldots, x_n) \tag{3.1}$$

Položaj i-tog od M majmuna bit će indeksiran s i:

$$\vec{x}_i = (x_{i1}, x_{i2}, \ldots, x_{in}) \tag{3.2}$$

Na početku je potrebno postaviti početne položaje svih M majmuna. Najjednostavnije je odrediti n-dimenzionalnu hiperkocku i iz nje uniformno uzorkovati položaj za svakog majmuna. Ako pojedini uzorkovani položaj ne zadovoljava ograničenja optimizacijskog problema, uzorkuje se ponovno dok ograničenja ne budu zadovoljena. Primjerice za hiperkocku $[0, 10]^n$:

Algorithm 2 Postavljanje početnih položaja

```
1: function POSTAVLJANJE(M, n)
2:     x = []
3:     for i = 1 to M do
4:         repeat
5:             for j = 1 to n do
6:                 x[i][j] = rand(0, 10)
7:         until x[i] zadovoljava ograničenja
8:     return x
```

3.2.2. Penjanje

Cilj penjanja je doći do položaja za koji je funkcija cilja veća nego za početni položaj. Za pomicanje koristi se pseudogradijent utemeljen na SPSA (*simultaneous perturbation stochastic approximation*.)

1. Za majmuna i generira se vektor $\Delta\vec{x}_i = (\Delta\vec{x}_{i1}, \Delta\vec{x}_{i2}, \ldots, \Delta\vec{x}_{in})$ gdje je

   $$\Delta\vec{x}_{ij} = \begin{cases} a & \text{s vjerojatnošću } 0.5 \\ -a & \text{s vjerojatnošću } 0.5 \end{cases}$$

   Parametar $a > 0$ zove se *dužina koraka* i ovisi o optimizacijskom problemu.

2. Računa se pseudogradijent funkcije cilja na položaju majmuna:

   $$f_i'(\vec{x}_i) = (f_{i1}'(\vec{x}_i), f_{i2}'(\vec{x}_i), \ldots, f_{in}'(\vec{x}_i))$$
   $$f_{ij}'(\vec{x}_i) = \frac{f(\vec{x}_i + \Delta\vec{x}_i) - f(\vec{x}_i - \Delta\vec{x}_i)}{2\Delta x_{ij}}$$

3. Neka je $\vec{y} = (y_1, y_2, \ldots, y_n)$ gdje je

   $$y_j = x_{ij} + a \cdot \text{sgn}(f_{ij}'(\vec{x}_i))$$

4. Ako $\vec{y}$ zadovoljava ograničenja, ažurira se položaj majmuna $\vec{x}_i \leftarrow \vec{y}$

Za svakog majmuna se koraci 1 - 4 ponavljaju sve dok promjena funkcije cilja nije dovoljno mala ili dok se ne prođe maksimalan broj ponavljanja.

Algorithm 3 Penjanje

```
1:  function PENJANJE(Nc, x, x*, a)
2:      for i = 1 to M do
3:          repeat
4:              Δx = []
5:              f' = []
6:              y = []
7:              for j = 1 to n do
8:                  Δx[j] = rand([a, -a], [0.5, 0.5])
9:              for j = 1 to n do
10:                 f'[j] = (f(x[i] + Δx) - f(x[i] - Δx))/(2Δx[j])
11:             for j = 1 to n do
12:                 y[j] = x[i][j] + a*sgn(f'[j])
13:             if y zadovoljava ograničenja then x[i] = y
14:             if f(x[i]) > f(x*) then x* = x[i]
15:         until razlika f(x[i]) dovoljno mala ili dosegnut broj ponavljanja Nc
```

3.2.3. Gledanje i skakanje

Cilj gledanja i skakanja je prebaciti se na točku u blizini čija je vrijednost funkcije cilja veća od trenutne. Definiran je parametar b koji označava doseg majmunovog pogleda.

1. Neka je $\vec{y} = (y_1, y_2, \ldots, y_n)$ gdje je

   y_j uzorkovan iz uniformne distribucije na $[x_{ij} - b, x_{ij} + b]$.

2. Ako $\vec{y}$ zadovoljava ograničenja i vrijedi $f(\vec{y}) \geq f(\vec{x}_i)$, ažurira se položaj majmuna $\vec{x}_i \leftarrow \vec{y}$

Za svakog majmuna se korak 1 ponavlja sve dok se ne zadovolji uvjet u koraku 2.

Algorithm 4 Gledanje i skakanje

```
1: function GLEDANJESKAKANJE(x, b)
2:     for i = 1 to M do
3:         repeat
4:             y = []
5:             for j = 1 to n do
6:                 y[j] = rand(x[i][j] - b, x[i][j] + b))
7:         until y zadovoljava ograničenja i f(y) >= f(x[i])
8:         x[i] = y
```

3.2.4. Salto

Cilj salta je prebaciti se u novu domenu. Određuje se centroid položaja svih majmuna i svaki majmun radi salto u njegovom smjeru. Definiran je raspon $[c, d]$ iz kojeg se uniformno uzorkuje duljina salta.

Računa se centroid p:

$$\vec{p} = \frac{1}{M} \sum_{i=1}^{M} \vec{x}_i \tag{3.3}$$

1. Neka je $\vec{y} = \vec{x}_i + \alpha(\vec{p} - \vec{x}_i)$ gdje je

   α uzorkovan iz uniformne distribucije na $[c, d]$

2. Ako $\vec{y}$ zadovoljava ograničenja, ažurira se položaj majmuna $\vec{x}_i \leftarrow \vec{y}$

Za svakog majmuna se korak 1 ponavlja sve dok se ne zadovolji uvjet u koraku 2.

Algorithm 5 Salto

```
1:  function SALTO(x, [c, d])
2:      p = []
3:      for i = 1 to M do
4:          p += x[i]
5:      p = p/M
6:      for i = 1 to M do
7:          repeat
8:              α = rand(c, d)
9:              y = x + α * (p - x[i])
10:         until y zadovoljava ograničenja
11:         x[i] = y
```

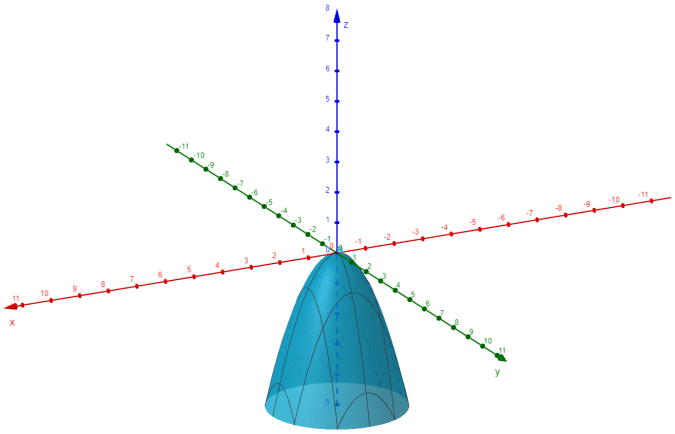
3.3. Programsko ostvarenje

3.3.1. Primjer funkcije za maksimizaciju

Neka se maksimizira funkcija:

$$f(\vec{x}) = -(\vec{x}^\intercal \vec{x}) \tag{4.4}$$
$$\vec{x} \in [-100, 100]^n$$

Funkcija ima maksimum u točki $\vec{0}$ i on iznosi $f(\vec{0}) = 0$. Minimum je na kutovima domene i iznosi $-100^2 n$.



Slika 3.1: Graf funkcije $f(\vec{x})$ za $n = 2$

3.3.2. Vlastito programsko ostvarenje

Vlastito programsko ostvarenje algoritma majmuna povedeno je prevođenjem pseudo-koda u programski jezik Python. Korištene su knjižnice programa za Python: *numpy* i *random*. Sastoji od klase `MA` sa sljedećim metodama:

1. `__init__(self, M, N, Nc, a, b, c, d)`

   Instanciranje klase i postavljanje parametara algoritma

2. `optimize(self, f, cond, n, l, r)`

   Optimizacija funkcije `f` s funkcijom ograničenja `cond`

3. `initialize(self, cond, l, r, M, n)`

   Generiranje početnih položaja svih majmuna

4. `climb(self, M, Nc, n, X, a)`

5. `watchJump(self, M, n, X, b)`

6. `sumersault(self, M, n, X, c, d)`

7. Pomoćne funkcije:

   `sampleHypercube(self, n, l, r)`

   `sampleWatch(self, n, x, b)`

`sampleDx(self, n, a)`

`M`	Broj majmuna
`N`	Broj iteracija algoritma
`Nc`	Broj skakanja
`f`	Funkcija koja se optimira
`cond`	Funkcija kojom se provjerava zadovoljenost ograničenja
`n`	Dimenzionalnost ulaza funkcije
`a`	Duljina skoka
`b`	Udaljenost pogleda
`c`	Donja granica duljine salta
`d`	Gornja granica duljine salta
`l`	Donja granica u svakoj dimenziji hiperkocke
`r`	Donja granica u svakoj dimenziji hiperkocke
`X`	Lista svih majmuna
`x`	Položaj pojedinog majmuna

Primjer izvođenja

Optimira se funkcija (4.4) s $n = 30$ dimenzija s $M = 5$ majmuna, $N = 60$ iteracija, $Nc = 500$ skokova, duljinom koraka $a = 0.0001$, udaljenošću pogleda $b = 10$ i duljinom salta $[c, d] = [-1, 1]$.

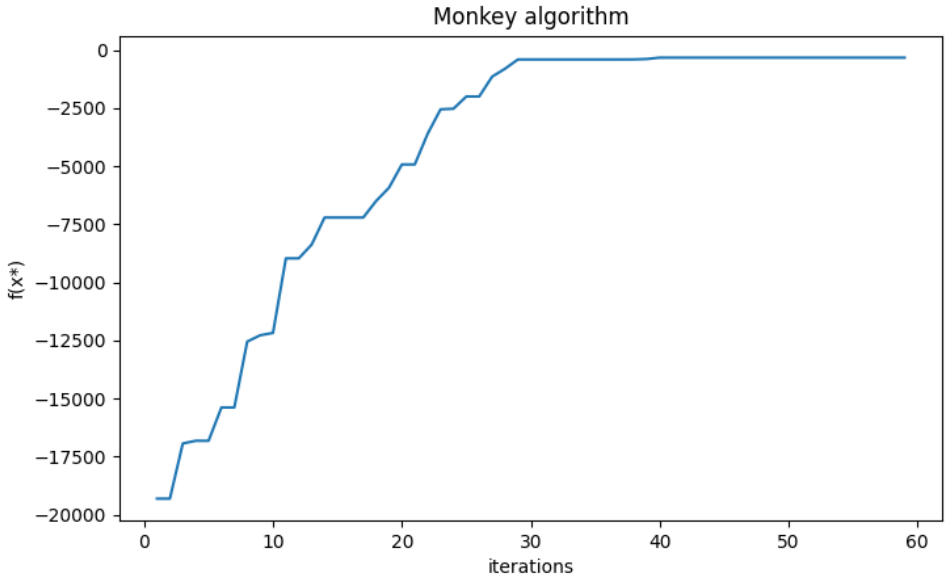
Klasa `MA` nalazi se u datoteci `ma.py`. Može se pokrenuti na sljedeći način:

```
import ma
import numpy as np
import matplotlib.pyplot as plt

f = lambda x : -np.dot(x, x)
cond=lambda x : np.max(x) < 100 and np.min(x) > -100

m = ma.MA(M=5, N=60, Nc=500, a=0.0001, b=10, c=-1, d=1)
x, fs = m.optimize(f=f, cond=cond, n=30, l=-100, r=100)

print("x* =", x)
print("f(x*) =", f(x))
```

```
plt.plot(range(1, len(fs)), fs[1:])
plt.show()
```

Izlaz ovog primjera je:

```
x* = [ 3.29615006  4.05143309  ...  -3.15835887]
f(x*) = -330.1389091136464
```



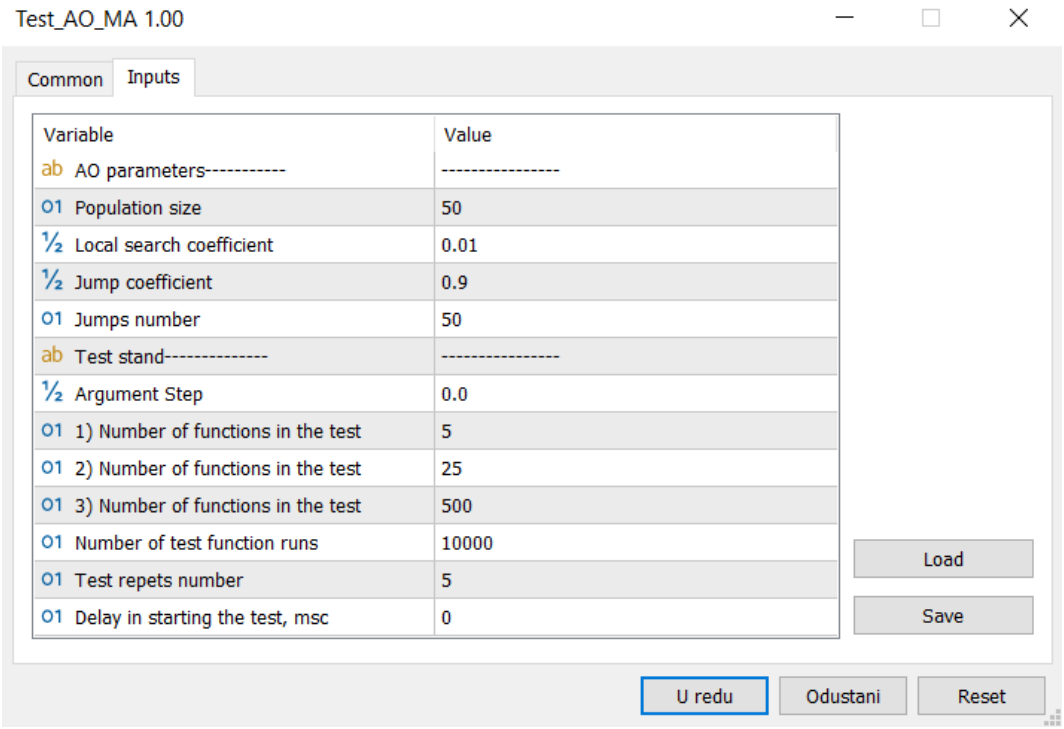
Slika 3.2: Graf ovisnosti izlaza funkcije o broju iteracija za dani primjer

3.3.3. Programsko ostvarenje MQL5

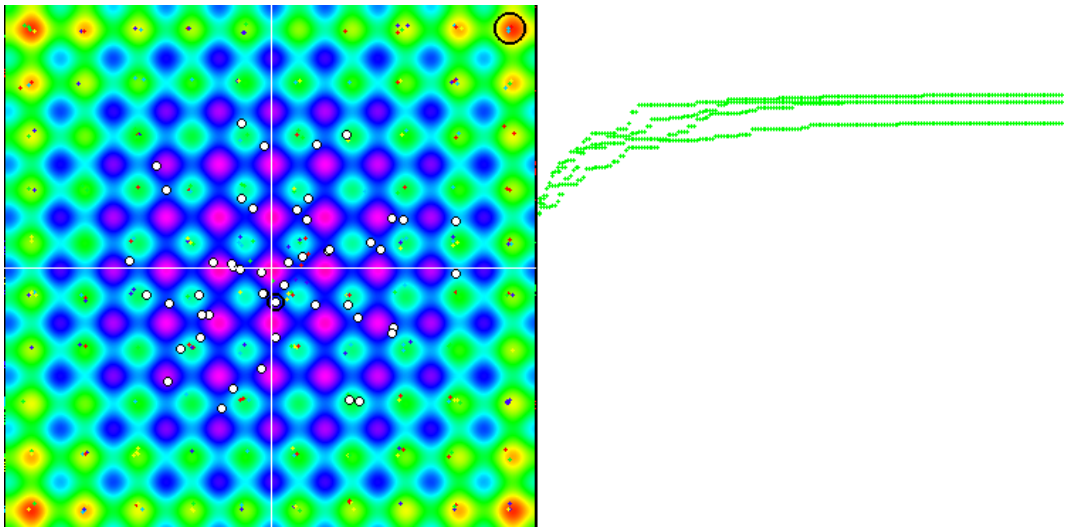
Programsko ostvarenje algoritma majmuna autora A. Dika[4] može se preuzeti na internetskoj stranici MQL5. Važno je napomenuti da se ovo ostvarenje razlikuje se od onog u pseudokodu u pogledu funkcija *gledanje i skakanje* i *salto*.

Za pokretanje potrebno je instalirati program Meta Trader 5, otvoriti korisnički račun i unutar programa otvoriti dijagram za proizvoljni *symbol*. Zip dokument s algoritmima smjesti se u podatkovnu strukturu programa MQL5. Za pokretanje algoritma

na preddefiniranim testnim funkcijama odabire se `Scripts/Test_OA_MA.mq5` u *Navigatoru*. U novootvorenom prozoru postavljaju se parametri i pokreće se algoritam koji u stvarnom vremenu prikazuje položaje majmuna na hiperravnini i ispisuje rezultat i pogrešku.



Slika 3.3: Odabir parametara algoritma u Meta Traderu 5



Slika 3.4: Položaji majmuna (bijele točke) u stvarnom vremenu na hiperravnini (lijevo) i izlazi funkcije cilja za 4 pokretanja algoritma (desno)

Ispis je:

```
5 Rastrigin's; Func runs 10000 result: 68.03668259
Score: 0.84301
25 Rastrigin's; Func runs 10000 result: 55.4356683739
Score: 0.68688
500 Rastrigin's; Func runs 10000 result: 41.238597562
Score: 0.51097
============================
5 Forest's; Func runs 10000 result: 0.445023241351
Score: 0.25173
25 Forest's; Func runs 10000 result: 0.183789574445
Score: 0.10396
500 Forest's; Func runs 10000 result: 0.062930759789
Score: 0.03560
============================
5 Megacity's; Func runs 10000 result: 2.63999999999
Score: 0.22000
25 Megacity's; Func runs 10000 result: 1.11199999999
Score: 0.09267
500 Megacity's; Func runs 10000 result: 0.3276
Score: 0.02730
```

Vlastite funkcije za optimizaciju definiraju se u `Include/Math/functions.mqh` u programskom jeziku temeljenom na C++. Funkcija (4.4) može se definirati na sljedeći način:

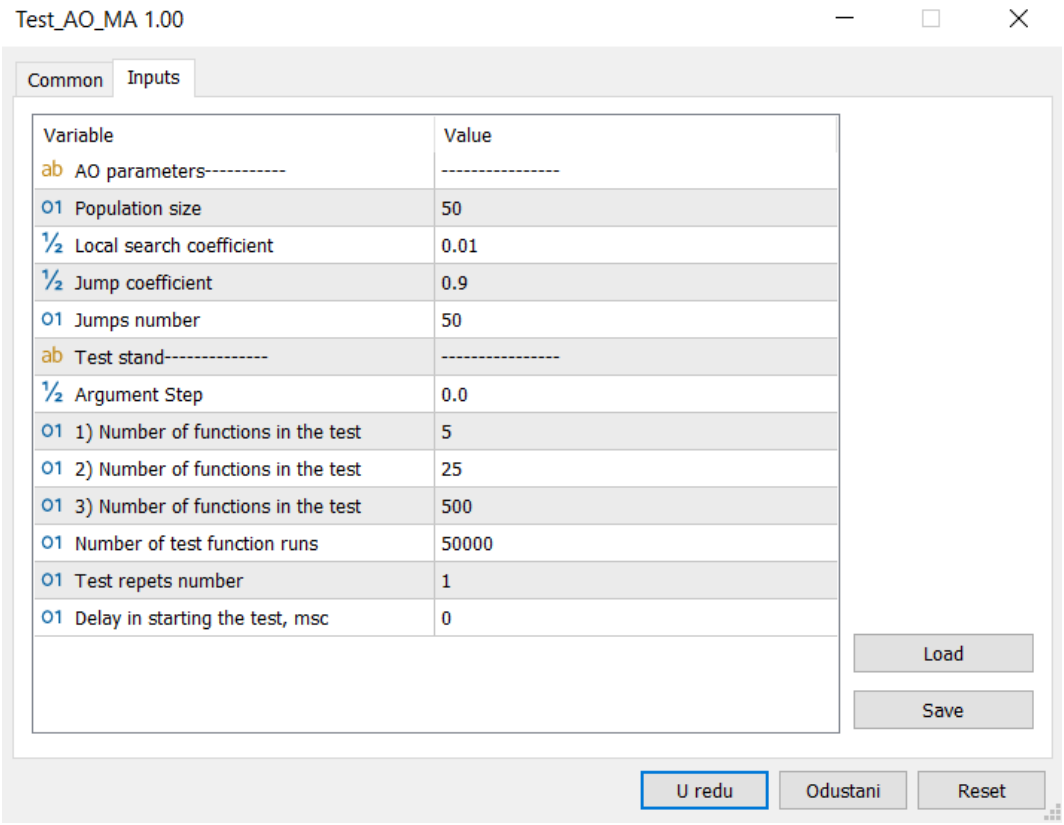
```
class C_Sphere : public C_Function
{
public: //==================================
C_Sphere ()
{
SetNamFun ("Sphere");
```

```
SetMinX (-100.0);
SetMaxX ( 100.0);
SetMinY (-100.0);
SetMaxY ( 100.0);

SetMinFun   (-20000.0);        //4 points
SetMinFuncX (100.0);
SetMinFuncY (100.0);

SetMaxFun   (0.0);            //1 points
SetMaxFuncX (0.0);
SetMaxFuncY (0.0);
}

private: //==================================
double Core (double x, double y)
{
double res = -(x*x + y*y);
return(res);
}
};
```

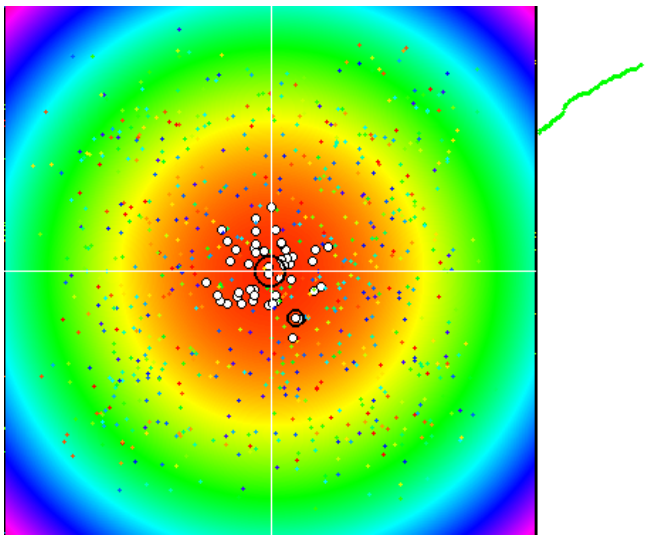
Za pokretanje optimizacije proizvoljne funkcije, potrebno je u `Test_OA_MA.mq5` u funkciji `void OnStart()` upisati poziv funkcije `FuncTests`. Parametri funkcije su: funkcija, dimenzija $n/2$ i boja linije na grafu funkcije cilja. Primjerice za funkciju `C_Sphere`:

```
C_Sphere F;
FuncTests(F, 15, clrLime);
```

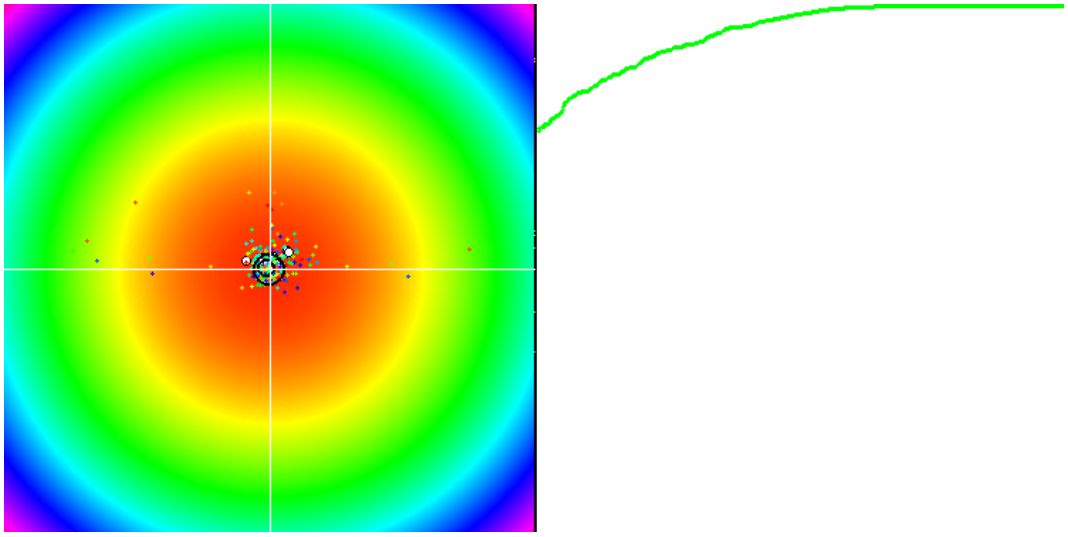
Optimira se funkcija (4.4) s $n = 30$ dimenzija s $M = 50$ majmuna, $N = 50000$ iteracija, $Nc = 50$ skokova, duljinom koraka $a = 0.01$ i udaljenošću pogleda $b = 0.9$.



Slika 3.5: Odabir parametara algoritma za primjer



Slika 3.6: Položaji majmuna hiperravnini i izlazi funkcije cilja tijekom izvođenja



Slika 3.7: Položaji majmuna hiperravnini i izlazi funkcije cilja nakon izvođenja

Ispis je:

```
15 Sphere's; Func runs 50000 result: -2.5525976674831137
Score: 0.99987
```

3.4. Primjene

3.4.1. Optimizacija hibridnih mikromreža

Izmijenjeni algoritam majmuna upotrebljen je za optimizaciju konfiguracije komponenti hibridnih mikromreža.[6] Minimizirana je cijena i emisija stakleničkih plinova. Algoritam je izmijenjen uvođenjem meteoroloških faktora.

3.4.2. Grupiranje

Hibridni algoritam majmuna utemeljen na operatoru pretraživanja iz algoritma kolonije pčela upotrebljen je za grupiranje.[2] Pokazano je da ima dobre performanse na sintetičkim i stvarnim skupovima podataka.

3.4.3. 0-1 problem naprtnjače

Izmijenjenim algoritam majmuna predložen je za rješavanje 0-1 problema naprtnjače.[10] Lokalno pretraživanje ojačano je pohlepnima algoritmom a salto je izmijenjen kako bi se izbjeglo upadanje u lokalne optimume. Uveden je i kooperativni proces radi ubrzanje konvergencije. Pokazano je da je izmijenjeni algoritam majmuna učinkovita alternativa za rješavanje ovog problema.

3.4.4. Raspoređivanje zadataka u računarstvu u oblaku

Algoritam majmuna primijenjen je na raspoređivanje zadataka u računarstvu u oblaku.[5] Minimizirana je cijena i maksimizirana iskorištenost kako bi se pružila bolja usluga klijentu. Pokazano je da takav algoritam radi učinkovitije od prethodno predloženih.

3.4.5. Flow-shop scheduling

Za NP-teški kombinatorni problem *flow-shop scheduling* upotrebljen je hibridni algoritam majmuna s pod-populacijama.[8] Algoritam majmuna nadmašio je mnoge heurističke i metaheurističke algoritme iz literature.

3.4.6. Raspoređivanje elektroničkih komponenata u računalima

Algoritam majmuna korišten je za optimiziranje problema raspoređivanja elektroničkih komponenata u računalima.[7] Pokazano je da daje učinkovitija rješenja nego genetski algoritam.

3.4.7. Set cover problem

Za rješavanje *set cover* problema, dizajniran je binarni algoritam majmuna.[3] Izmijenjen je proces penjanja majmuna i dodan je proces kooperacije. Pokazuje se da takav algoritam nadmašuje mnoge heurističke i metaheurističke algoritme iz literature.

3.4.8. Prepoznavanje tumora u MRI slikama mozga

Automatizirani algoritam majmuna korišten je pronalaženje i segmentaciju lokacija tumora na MRI slikama mozga.[1] Tehniku je moguće upotrijebiti za pomoć radiolozima pri pronalaženju tumora.

4. Zaključak

5. Literatura

[1] Saravanan Alagarsamy, T. Abitha, S. Ajitha, S. Sangeetha, i Vishnuvarthanan Govindaraj. Identification of high grade and low grade tumors in mr brain image using modified monkey search algorithm. *IOP Conference Series: Materials Science and Engineering*, 993(1):012052, dec 2020. doi: 10.1088/1757-899X/993/1/012052. URL https://dx.doi.org/10.1088/1757-899X/993/1/012052.

[2] Xin Chen, Yongquan Zhou, i Qifang Luo. A hybrid monkey search algorithm for clustering analysis. *The Scientific World Journal*, 2014:938239, Mar 2014. ISSN 2356-6140. doi: 10.1155/2014/938239. URL https://doi.org/10.1155/2014/938239.

[3] Broderick Crawford, Ricardo Soto, Rodrigo Olivares, Gabriel Embry, Diego Flores, Wenceslao Palma, Carlos Castro, Fernando Paredes, i José-Miguel Rubio. A binary monkey search algorithm variation for solving the set covering problem. *Natural Computing*, 19(4):825–841, Dec 2020. ISSN 1572-9796. doi: 10.1007/s11047-019-09752-8. URL https://doi.org/10.1007/s11047-019-09752-8.

[4] Andrey Dik. Population optimization algorithms: Monkey algorithm (ma), 2023. URL https://www.mql5.com/en/articles/12212.

[5] Punit Gupta i Prateek Tewari. Monkey search algorithm for task scheduling in cloud iaas. U *2017 Fourth International Conference on Image Information Processing (ICIIP)*, stranice 1–6, 2017. doi: 10.1109/ICIIP.2017.8313789.

[6] Carlos M. Ituarte-Villarreal, Nicolas Lopez, i Jose F. Espiritu. Using the monkey algorithm for hybrid power systems optimization. *Procedia Computer Science*, 12:344–349, 2012. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2012.09.082. URL https://www.sciencedirect.com/

science/article/pii/S1877050912006734. Complex Adaptive Systems 2012.

[7] Elmar Kuliev, Vladimir Kureichik, i Vladimir Kureichik. Monkey search algorithm for ece components partitioning. *Journal of Physics: Conference Series*, 1015(4):042026, may 2018. doi: 10.1088/1742-6596/1015/4/042026. URL https://dx.doi.org/10.1088/1742-6596/1015/4/042026.

[8] M.K. Marichelvam, Ömür Tosun, i M. Geetha. Hybrid monkey search algorithm for flow shop scheduling problem under makespan and total flow time. *Applied Soft Computing*, 55:82–92, 2017. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2017.02.003. URL https://www.sciencedirect.com/science/article/pii/S1568494617300716.

[9] Ruiqing Zhao i Wansheng Tang. Monkey algorithm for global numerical optimization. *Journal of Uncertain Systems*, 2:165–176, 01 2008.

[10] Yongquan Zhou, Xin Chen, i Guo Zhou. An improved monkey algorithm for a 0-1 knapsack problem. *Applied Soft Computing*, 38:817–830, 2016. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2015.10.043. URL https://www.sciencedirect.com/science/article/pii/S1568494615006833.

6. Sažetak

Sažetak.