

SVEUČILIŠTE U ZAGREBU  
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

# Algoritam majmuna i algoritam krijesnica

*Velimir Kovačić*

Voditelj: *Marin Golub*

Zagreb, prosinac 2023.

# SADRŽAJ

<b>1. Uvod</b>	<b>1</b>
<b>2. Algoritam majmuna</b>	<b>2</b>
2.1. Uvod . . . . .	2
2.2. Algoritam . . . . .	3
2.2.1. Postavljanje početnih položaja . . . . .	3
2.2.2. Penjanje . . . . .	4
2.2.3. Gledanje i skakanje . . . . .	5
2.2.4. Salto . . . . .	6
2.3. Programsko ostvarenje . . . . .	7
2.3.1. Primjer funkcije za maksimizaciju . . . . .	7
2.3.2. Vlastito programsko ostvarenje . . . . .	8
2.3.3. Programsko ostvarenje MQL5 . . . . .	11
2.4. Algoritam majmuna u primjeni . . . . .	16
2.4.1. Optimizacija hibridnih mikromreža . . . . .	16
2.4.2. Grupiranje . . . . .	16
2.4.3. 0-1 problem naprtnjače . . . . .	16
2.4.4. Raspoređivanje zadataka u računarstvu u oblaku . . . . .	16
2.4.5. Flow-shop scheduling . . . . .	16
2.4.6. Raspoređivanje elektroničkih komponenata u računalima . . . . .	16
2.4.7. Set cover problem . . . . .	17
2.4.8. Prepoznavanje tumora u MRI slikama mozga . . . . .	17
<b>3. Algoritam krijesnica</b>	<b>18</b>
3.1. Uvod . . . . .	18
3.2. Algoritam . . . . .	18
3.2.1. Privlačnost . . . . .	18
3.2.2. Kretanje . . . . .	19

3.2.3.	Pseudokod . . . . .	20
3.3.	Programsko ostvarenje . . . . .	20
3.3.1.	Programsko ostvarenje MEALPY . . . . .	20
3.3.2.	Programsko ostvarenje MQL5 . . . . .	22
3.4.	Algoritam kriješnica u primjeni . . . . .	23
3.4.1.	Economic Emissions Load Dispatch problem . . . . .	23
3.4.2.	Kombinatorna optimizacija . . . . .	23
3.4.3.	Punjenje hladnjaka radi uštede energije . . . . .	24
3.4.4.	Segmentacija slika . . . . .	24
3.4.5.	Procjena potražnje vodenih resursa . . . . .	24
3.4.6.	Navigacija mobilnih robota u promjenjivim okolinama . . . . .	24
3.4.7.	Odabir značajki za detekciju upada na mrežma . . . . .	24
3.4.8.	Optimizacija životnog vijeka mreže bežičnih senzora . . . . .	24
<b>4.</b>	<b>Zaključak</b>	<b>25</b>
<b>5.</b>	<b>Literatura</b>	<b>26</b>
<b>6.</b>	<b>Sažetak</b>	<b>29</b>

# 1. Uvod

Metaheuristika se odnosi na oblikovni obrazac koji ugrađuje prethodno znanje u izgradnju raznih optimizacijskih algoritama. Smatra se potpodručje stohastičke optimizacije. Stohastička optimizacija je kombinacija algoritama i slučajnih procesa radi pronalaženja optimalnih rješenja.

Prirodom nadahnete metaheuristike su one koje crpe ideje za pretraživanje prostora stanja iz prirode. Algoritmi rojeva su potkategorija prirodno nadahnutih metaheuristika koji su nadahnuti kolektivnim ponašanjem životinja ili kukaca. Roj se sastoji od više individua čije je ponašanje vrlo jednostavno, ali zajedno mogu djelovati inteligentno. Postoje mnogi takvi algoritmi, najpoznatiji su mravlji algoritam i algoritam pčela.

U ovom seminaru razmatraju se manje poznati algoritmi: algoritam majmuna i algoritam krijesnica. Oba algoritma opisana su i dan je pseudokod. Navedeni su načini na koje se algoritmi mogu lokalno pokrenuti. Dani su neki problemi na koje su algoritmi primijenjeni.

## 2. Algoritam majmuna

### 2.1. Uvod

Algoritam majmuna<sup>[20]</sup> 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.

## 2.2. 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*
```

---

### 2.2.1. Postavljanje početnih položaja

Položaj majmuna predstavlja  $n$ -dimenzionalni vektor:

$$\vec{x} = (x_1, x_2, \dots, x_n) \quad (2.1)$$

Položaj  $i$ -tog od  $M$  majmuna:

$$\vec{x}_i = (x_{i1}, x_{i2}, \dots, x_{in}) \quad (2.2)$$

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
```

---

### 2.2.2. Penjanje

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

1. Za majmuna  $i$  generira se vektor  $\Delta \vec{x}_i = (\Delta \vec{x}_{i1}, \Delta \vec{x}_{i2}, \dots, \Delta \vec{x}_{in})$  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 pseudogradient funkcije cilja na položaju majmuna:

$$f'_i(\vec{x}_i) = (f'_{i1}(\vec{x}_i), f'_{i2}(\vec{x}_i), \dots, f'_{in}(\vec{x}_i))$$

$$f'_{ij}(\vec{x}_i) = \frac{f(\vec{x}_i + \Delta \vec{x}_i) - f(\vec{x}_i - \Delta \vec{x}_i)}{2\Delta x_{ij}}$$

3. Neka je  $\vec{y} = (y_1, y_2, \dots, y_n)$  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:        $\Delta x = []$ 
5:        $f' = []$ 
6:        $y = []$ 
7:       for j = 1 to n do
8:          $\Delta x[j] = \text{rand}([a, -a], [0.5, 0.5])$ 
9:       for j = 1 to n do
10:         $f'[j] = (f(x[i] + \Delta x) - f(x[i] - \Delta x)) / (2\Delta x[j])$ 
11:       for j = 1 to n do
12:         $y[j] = x[i][j] + a * \text{sgn}(f'[j])$ 
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
```

---

### 2.2.3. Gledanje i skakanje

Cilj gledanja i skakanja je prebaciti se na položaj u blizini čija je vrijednost funkcije cilja veća nego za trenutni. Definiran je parametar  $b$  koji označava doseg najmnovog pogleda.

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

---

### 2.2.4. 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  $p$ :

$$\vec{p} = \frac{1}{M} \sum_{i=1}^M \vec{x}_i \quad (2.3)$$

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

$\alpha$  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:        $\alpha = \text{rand}(c, d)$ 
9:        $y = x + \alpha * (p - x[i])$ 
10:    until y zadovoljava ograničenja
11:    x[i] = y
```

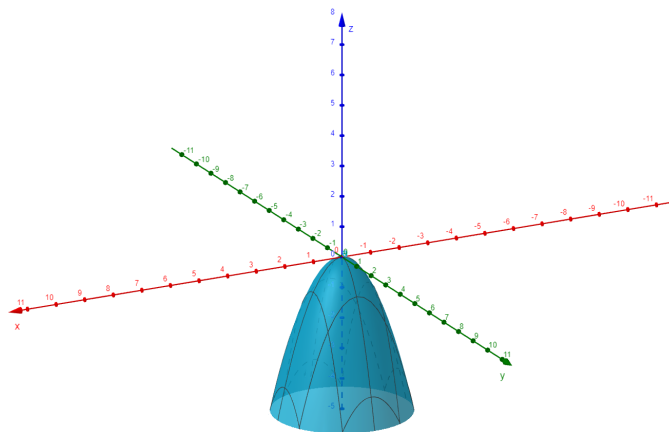
---

## 2.3. Programsko ostvarenje

### 2.3.1. Primjer funkcije za maksimizaciju

Neka se maksimizira  $n$ -dimenzionalna funkcija:

$$\begin{aligned} f(\vec{x}) &= -(\vec{x}^\top \vec{x}) \\ \vec{x} &\in [-100, 100]^n \end{aligned} \tag{4.4}$$



**Slika 2.1:** Graf funkcije  $f(\vec{x})$  za  $n = 2$

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

### 2.3.2. Vlastito programsko ostvarenje

Vlastito programsko ostvarenje algoritma majmuna povedeno je prevođenjem pseudokoda 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)`

Parametar	Značenje
M	Broj majmuna
N	Broj iteracija algoritma
$N_c$	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

**Tablica 2.1:** Parametri funkcija i njihova značenja

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

```

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()

```

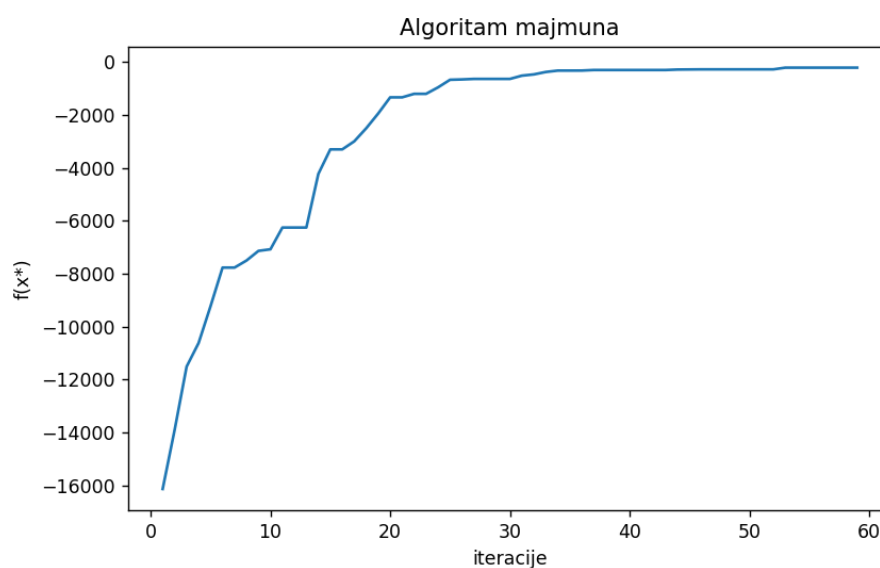
**Kod 1:** Način pokretanja algoritma majmuna

```

x* = [-4.44660877 -2.19507304 ... -1.93646392]
f(x*) = -222.05419841714073

```

**Ispis 1:** Ispis primjera



**Slika 2.2:** Graf ovisnosti izlaza funkcije o broju iteracija za dani primjer

### 2.3.3. Programsko ostvarenje MQL5

Programsko ostvarenje algoritma majmuna autora A. Dika<sup>[7]</sup> 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*.

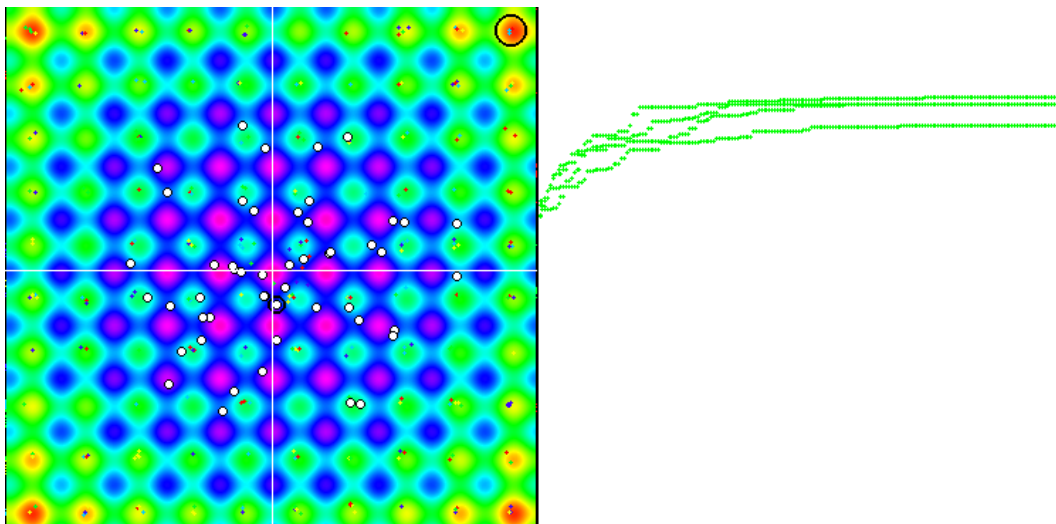
Za pokretanje potrebno je instalirati program Meta Trader 5, otvoriti korisnički račun i unutar programa otvoriti dijagram za proizvoljni *symbol*. Zip dokument s algoritmima smjesti se u podatkovnu strukturu programa MQL5.

Algoritam se na preddefiniranim testnim funkcijama pokreće odabirom `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.

Variable	Value
ab AO parameters-----	-----
01 Population size	50
1/2 Local search coefficient	0.01
1/2 Jump coefficient	0.9
01 Jumps number	50
ab Test stand-----	-----
1/2 Argument Step	0.0
01 1) Number of functions in the test	5
01 2) Number of functions in the test	25
01 3) Number of functions in the test	500
01 Number of test function runs	10000
01 Test repets number	5
01 Delay in starting the test, msc	0

Buttons: Load, Save, U redu, Odustani, Reset

**Slika 2.3:** Odabir parametara algoritma u Meta Traderu 5



**Slika 2.4:** Položaji majmuna (bijele točke) u stvarnom vremenu na hiperravnini (lijevo) i izlazi funkcije cilja za 4 pokretanja algoritma (desno)

```

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: 2.639999999999
Score: 0.22000
25 Megacity's; Func runs 10000 result: 1.111999999999
Score: 0.09267
500 Megacity's; Func runs 10000 result: 0.3276
Score: 0.02730

```

## Ispis 2: Ispis u MQL5 za preddefinirane testne funkcije

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);
    }
};
```

## Kod 2: Definicija vlastite funkcije u MQL5

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);
```

**Kod 3:** Pokretanje optimizacije vlastite funkcije u MQL5

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

Test\_AO\_MA 1.00

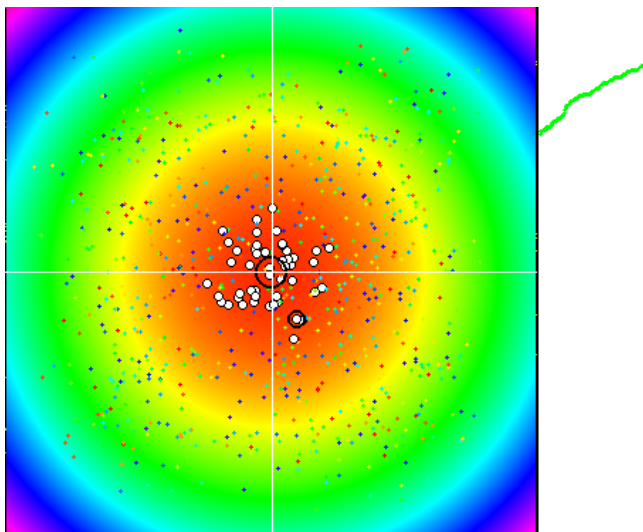
Common Inputs

Variable	Value
ab AO parameters-----	-----
01 Population size	50
1/2 Local search coefficient	0.01
1/2 Jump coefficient	0.9
01 Jumps number	50
ab Test stand-----	-----
1/2 Argument Step	0.0
01 1) Number of functions in the test	5
01 2) Number of functions in the test	25
01 3) Number of functions in the test	500
01 Number of test function runs	50000
01 Test repets number	1
01 Delay in starting the test, msc	0

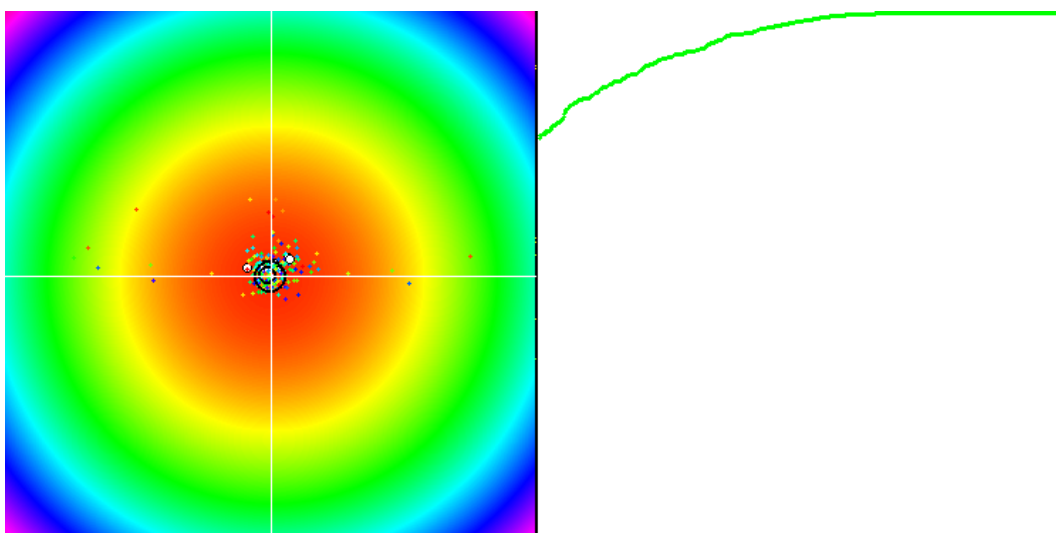
Load Save

U redu Odustani Reset

**Slika 2.5:** Odabir parametara algoritma za primjer



**Slika 2.6:** Položaji majmuna na hiperravnini (lijevo) i izlazi funkcije cilja tijekom izvođenja (desno)



**Slika 2.7:** Položaji majmuna na hiperravnini (lijevo) i izlazi funkcije cilja nakon izvođenja (desno)

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

### Ispis 3: Ispis u MQL5 za vlastitu testnu funkciju

## **2.4. Algoritam majmuna u primjeni**

### **2.4.1. Optimizacija hibridnih mikromreža**

Izmijenjeni algoritam majmuna upotrebljen je za optimizaciju konfiguracije komponenti hibridnih mikromreža.<sup>[11]</sup> Minimizirana je cijena i emisija stakleničkih plinova. Algoritam je izmijenjen uvođenjem meteoroloških faktora.

### **2.4.2. Grupiranje**

Hibridni algoritam majmuna utemeljen na operatoru pretraživanja iz algoritma kolonije pčela upotrebljen je za grupiranje.<sup>[4]</sup> Pokazano je da ima dobre performanse na sintetičkim i stvarnim skupovima podataka.

### **2.4.3. 0-1 problem naprtnjače**

Izmijenjenim algoritam majmuna predložen je za rješavanje 0-1 problema naprtnjače.<sup>[21]</sup> 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.

### **2.4.4. Raspoređivanje zadataka u računarstvu u oblaku**

Algoritam majmuna primijenjen je na raspoređivanje zadataka u računarstvu u oblaku.<sup>[10]</sup> Minimizirana je cijena i maksimizirana iskorištenost kako bi se pružila bolja usluga klijentu. Pokazano je da takav algoritam radi učinkovitije od prethodno predloženih.

### **2.4.5. Flow-shop scheduling**

Za NP-teški kombinatorni problem *flow-shop scheduling* upotrebljen je hibridni algoritam majmuna s pod-populacijama.<sup>[13]</sup> Algoritam majmuna nadmašio je mnoge heurističke i metaheurističke algoritme iz literature.

### **2.4.6. Raspoređivanje elektroničkih komponenata u računalima**

Algoritam majmuna korišten je za optimiziranje problema raspoređivanja elektroničkih komponenata u računalima.<sup>[12]</sup> Pokazano je da daje učinkovitija rješenja nego

genetski algoritam.

#### **2.4.7. Set cover problem**

Za rješavanje *set cover* problema, dizajniran je binarni algoritam majmuna.<sup>[5]</sup> Izmijenjen je proces penjanja majmuna i dodan je proces kooperacije. Pokazuje se da takav algoritam nadmašuje mnoge heurističke i metaheurističke algoritme iz literature.

#### **2.4.8. Prepoznavanje tumora u MRI slikama mozga**

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

## 3. Algoritam krijesnica

### 3.1. Uvod

Algoritam krijesnica<sup>[19]</sup> metaheuristički je optimizacijski algoritam sličan algoritmu rojeva čestica. Nadahnut je načinom na koji se krijesnice u prirodi međusobno privlače.

Krijesnice imaju sposobnost međusobnog odašiljanja svjetlosnih signala svojim svjetlećim organima. Neke vrste odašilju trepteće, a neke konstanto svjetlo. Takvi signali služe za privlačenje partnera za parenje i privlačenje potencijalnog plijena. U nekim vrstama mužjaci privlače ženke, a u drugima obratno.

### 3.2. Algoritam

U samom algoritmu razmatramo međusobno privlačenje krijesnica. Potrebno je idealizirati svojstva svijetljenja:

1. Ne razlikujemo muške i ženske krijesnice, sve svijetle i sve se međusobno privlače.
2. Privlačnost je proporcionalna intenzitetu svjetla kojeg odašilju i opada kvadratom udaljenosti  $I \propto \frac{1}{r^2}$ . Ona krijesnica s većim intenzitetom svjetla privlači onu s manjim.
3. Intenzitet svjetla krijesnice određen je funkcijom cilja.

#### 3.2.1. Privlačnost

Za intenzitet svjetla same krijesnice moguće je u najjednostavnijem slučaju uzeti  $I_0(\vec{x}) \propto f(\vec{x})$  (za maksimizacijske probleme) ili  $I_0(\vec{x}) \propto -f(\vec{x})$  (za minimizacijske probleme). Ključno je pitanje kakav intenzitet svjetla vide druge krijesnice, to jest koliku imaju privlačnost  $\beta$ .

Intenzitet svjetla na udaljenosti  $r$  je:

$$I(r) = \frac{I_0}{r^2} \quad (3.1)$$

Apsorpcija svjetla okoline  $\gamma$  je:

$$I(r) = I_0 e^{-\gamma r} \quad (3.2)$$

Ova dva izraza mogu se kombinirati u Gaussovoj formi čime se izbjegava nedefinirana vrijednost intenziteta svjetla na udaljenosti  $r = 0$ :

$$I(r) = I_0 e^{-\gamma r^2} \quad (3.3)$$

Mogu se upotrijebiti i druge monotonno padajuće funkcije:

$$I(r) = \frac{I_0}{1 + \gamma r^2} \quad (3.4)$$

Kako se definirao intenzitet svjetlosti, tako se može definirati privlačnost:

$$\beta(r) = \beta_0 e^{-\gamma r^2} \quad (3.5)$$

Privlačnost se može definirati i kao bilo koja monotonno padajuća funkcija oblika:

$$\beta(r) = \beta_0 e^{-\gamma r^m} \quad (3.6)$$

Karakteristična udaljenost  $\Gamma$  udaljenost je na kojoj privlačnost opada s  $\beta_0$  na  $\beta_0 e^{-1}$ . Ona iznosi  $\Gamma = \gamma^{-\frac{1}{m}}$ . Isto tako, na temelju karakteristične udaljenosti nekog problema, može se odrediti konstanta apsorpcije svjetla  $\gamma = \frac{1}{\Gamma^m}$

### 3.2.2. Kretanje

Položaj krijesnice predstavlja  $n$ -dimenzionalni vektor:

$$\vec{x} = (x_1, x_2, \dots, x_n) \quad (3.7)$$

Položaj  $i$ -te od  $N$  krijesnica:

$$\vec{x}_i = (x_{i1}, x_{i2}, \dots, x_{in}) \quad (3.8)$$

Udaljenost između krijesnica računamo euklidskom udaljenošću:

$$r_{ij} = \|\vec{x}_i - \vec{x}_j\| = \sqrt{\sum_{k=1}^n (x_{ik} - x_{jk})^2} \quad (3.9)$$

Kretanje krijesnice  $i$  prema krijesnici  $j$  većeg intenziteta svjetla je:

$$\vec{x}_i \leftarrow \vec{x}_i + \beta_0 e^{-\gamma r_{ij}^m} (\vec{x}_j - \vec{x}_i) + \alpha \text{rand}(-1, 1) \quad (3.10)$$

Drugi pribrojnik predstavlja pomak u smjeru privlačne krijesnice, a treći predstavlja slučajni pomak.

Uobičajeno je uzeti da su parametri  $m = 2$ ,  $\beta_0 = 1$  i  $\alpha \in [0, 1]$ . To ostavlja parametar  $\gamma$  čiji odabir najviše utječe na konvergenciju algoritma, uobičajeno ga je odrediti na temelju karakteristične udaljenosti  $\Gamma$ .

### 3.2.3. Pseudokod

---

**Algorithm 6** Algoritam krijesnica

---

```

1: function FA(n, N, maxGen,  $\gamma$ )
2:   X = INICIJALIZACIJA(n, d)
3:   while gen < maxGen do
4:     for i = 1 to n do
5:       for j = 1 to n do
6:         if I(X[j]) > I(X[i]) then
7:           Pomakni krijesnicu X[i] prema krijesnici X[j]
8:           Ažuriraj intenzitet svjetla krijesnice X[i]
9:       Ažuriraj najbolje rješenje x*
10:  return x*
```

---

## 3.3. Programsko ostvarenje

### 3.3.1. Programsko ostvarenje MEALPY

Algoritam krijesnica i mnogi drugi metaheuristički algoritmi programski su ostvareni u sklopu knjižnice programa otvorenog koda MEALPY<sup>[17]</sup> programskog jezika Python. Knjižnica se može preuzeti naredbom konzole:

```
> pip install mealpy
```

**Kod 4:** Naredba konzole za preuzimanje knjižnice MEALPY

Prethodno opisani problem funkcije (4.4) s  $n = 30$  definira se u programu kao rječnik i pokreće se algoritam krijesnica s 1000 epoha, brojem krijesnica  $N = 50$ ,  $\gamma = 0.1$ ,  $\beta_0 = 1$ ,  $\alpha = 0.5$  i  $m = 2$ :

```
import numpy as np
from mealpy import FloatVar, FFA

n = 30

def objective_function(solution):
    return -np.sum(solution**2)

problem = {
    "obj_func": objective_function,
    "bounds": FloatVar(lb=(-10., )*n, ub=(10., )*n),
    "minmax": "max",
}

model = FFA.OriginalFFA(epoch=1000, pop_size=50,
    gamma = 0.1, beta_base = 1,
    alpha = 0.5, exponent = 2)

g_best = model.solve(problem)

print("Solution:", g_best.solution)
print("Fitness:", g_best.target.fitness)
```

#### Kod 5: Pokretanje optimizacije vlastite funkcije

```
Solution: [-1.77332972  2.96553989 ... -3.77894221]
Fitness: -146.04412518749243
```

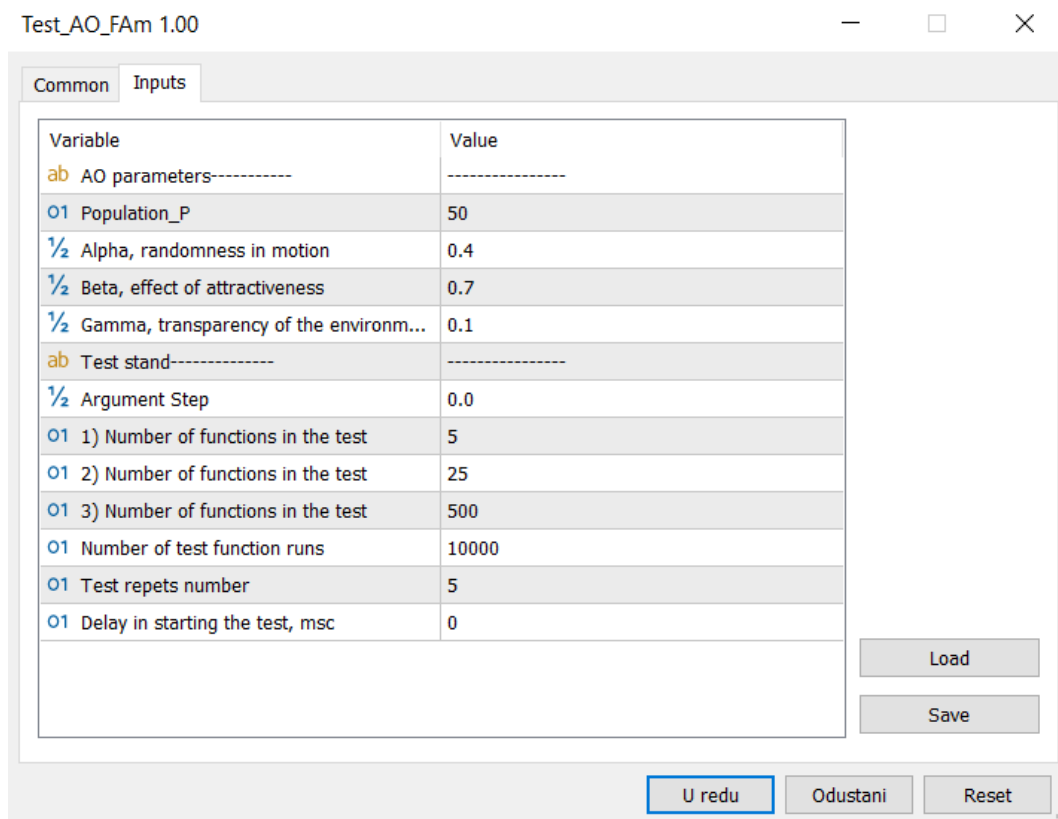
#### Ispis 4: Ispis primjera



### 3.3.2. Programsko ostvarenje MQL5

Programsko ostvarenje A. Dika<sup>[6]</sup> u jeziku MQL5 može se preuzeti i pokrenuti na način analogan onom opisanom u odjeljku 2.3.3.

<https://www.mql5.com/en/articles/11873> Primjer pokretanja (4.4) s  $n = 30$  s 1000 epoha,  $N = 50$ ,  $\gamma = 0.1$ ,  $\beta_0 = 1$ ,  $\alpha = 0.5$ :

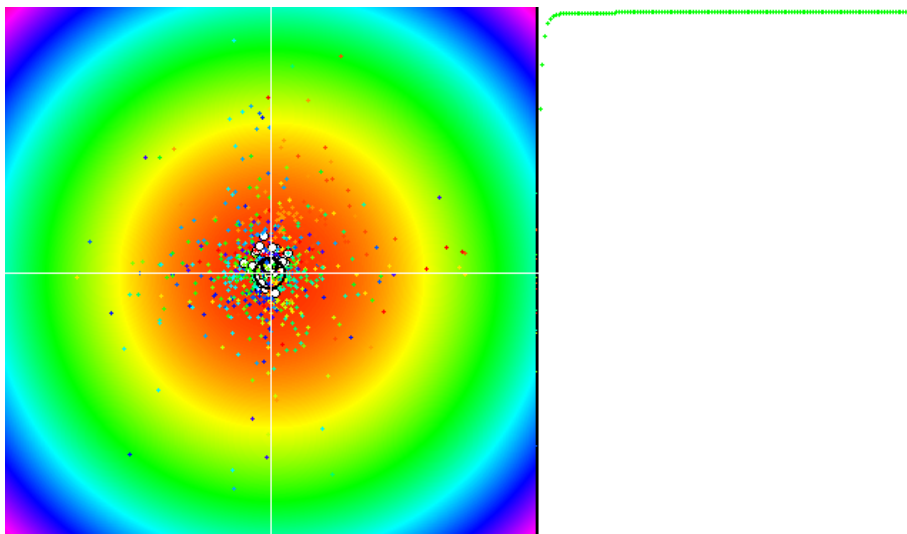


The screenshot shows a software window titled "Test\_AO\_FAm 1.00" with a standard Windows interface (minimize, maximize, close buttons). The window has two tabs: "Common" and "Inputs", with "Inputs" currently selected. Inside the "Inputs" tab, there is a table with two columns: "Variable" and "Value". The table contains the following entries:

Variable	Value
ab AO parameters-----	-----
01 Population_P	50
1/2 Alpha, randomness in motion	0.4
1/2 Beta, effect of attractiveness	0.7
1/2 Gamma, transparency of the environm...	0.1
ab Test stand-----	-----
1/2 Argument Step	0.0
01 1) Number of functions in the test	5
01 2) Number of functions in the test	25
01 3) Number of functions in the test	500
01 Number of test function runs	10000
01 Test repets number	5
01 Delay in starting the test, msc	0

Below the table, there are two buttons: "Load" and "Save". At the bottom of the window, there are three buttons: "U redu" (highlighted with a blue border), "Odustani", and "Reset".

**Slika 3.1:** Odabir parametara algoritma za primjer



**Slika 3.2:** Položaji kriješnica na hiperravnini (lijevo) i izlazi funkcije cilja tijekom izvođenja (desno)

```
15 Sphere's; Func runs 10000 result: -70.09166510586255
Score: 0.99650
```

#### Ispis 5: Ispis primjera

### 3.4. Algoritam kriješnica u primjeni

#### 3.4.1. Economic Emissions Load Dispatch problem

Algoritam kriješnica primijenjen je na problem *Economic Emissions Load Dispatch* u kojem se istovremeno minimizira cijena i količina emisija stakleničkih plinova.<sup>[2]</sup> Pokazalo se da je uz pravilan odabir parametara moguće postići dobra rješenja.

#### 3.4.2. Kombinatorna optimizacija

Hibridna inačica algoritma kriješnica, uz heuristiku lokalnog pretraživanja, upotrijebljena je na problemu bojanja grafa s 3 boje.<sup>[9]</sup> Rezultati su bili obećavajući i pokazali da je moguće koristiti algoritam i za druge kombinatorne probleme.

### **3.4.3. Punjenje hladnjaka radi uštede energije**

Poboljšani algoritam krijesnica (IFA) korišten je za rješavanje problema punjenja hladnjaka.<sup>[8]</sup> Imao je bolje performanse od nekoliko optimizacijskih algoritama iz literature.

### **3.4.4. Segmentacija slika**

Algoritam krijesnica i algoritam  $k$ -NN korišteni su u kombinaciji za grupiranje piksela slika u  $k$  grupa.<sup>[16]</sup>

### **3.4.5. Procjena potražnje vodenih resursa**

Novi dinamični algoritam krijesnica (NDFA) korišten je za procjenu potražnje vodenih resursa u gradu Nanchangu u Kini.<sup>[18]</sup> Podaci o potražnji od 2003. do 2012. korišteni su za učenje modela, a predviđanje je ispitivano na podacima od 2013. do 2015. Točnost predviđanja bila je 97.91%.

### **3.4.6. Navigacija mobilnih robota u promjenjivim okolinama**

Navigacija grupe mobilnih robota u promjenjivim okolinama ostvarena je algoritmom krijesnica u kojem roboti predstavljaju krijesnice i vodeći se algoritmom, istražuju okolinu.<sup>[15]</sup> Pokazalo se da su performanse ovog rješenja bolje nego drugih predloženih.

### **3.4.7. Odabir značajki za detekciju upada na mrežma**

Algoritam krijesnica upotrebljen je za odabir značajki na temelju kojih se detektira upad.<sup>[3]</sup> Rezultat je pokazao je obećavajuć napredak u odnosu na druge prijedloge rješenja.

### **3.4.8. Optimizacija životnog vijeka mreže bežičnih senzora**

Za optimizaciju životnog vijeka mreže bežičnih senzora korišten je poboljšani algoritam krijesnica (IFA).<sup>[22]</sup> Simulacijama je pokazano da taj algoritam ima bolje i ujednačenije performanse od drugih algoritama.

## 4. Zaključak

Oba algoritma (algoritam majmuna i algoritam krijesnica) unatoč vrlo jednostavnim formulacijama pokazuju obećavajuće rezultate na stvarnim problemima od znanstvenih i matematičkih do industrijskih i komercijalnih.

Prednost algoritma majmuna je dobra metoda za izlaženje iz lokalnih maksimuma, a to je funkcija salta. Zbog svoje slabe zastupljenosti, nije uključen u knjižnice prirodom nadahnutih optimizacijskih algoritama poput MEALPY, nego ga je potrebno samostalno programski ostvariti. Pokazuje se korisnim u primjeni na različite optimizacijske probleme.

Algoritam krijesnica unaprjeđenje je algoritma roja čestica načinom na koji se krijesnice privlače. Algoritam je poznat te se može pronaći u različitim knjižnicama optimizacijskih algoritama što olakšava primjenu. Kao i algoritam majmuna može se primijeniti na različite optimizacijske probleme s dobrim rezultatima.

Općenito, prirodnim nadahnuti metaheuristički algoritmi rojeva na učinkovit način će pronaći neko rješenje problema. Takvo rješenje neće nužno biti najbolje moguće, ali može biti dovoljno dobro za pojedino područje primjene. Daljnjim istraživanjem mogu se pronaći učinkovitiji algoritmi i poboljšati postojeći.

## 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] Theofanis Apostolopoulos i Aristidis Vlachos. Application of the firefly algorithm for solving the economic emissions load dispatch problem. *International Journal of Combinatorics*, 2011:523806, Dec 2010. ISSN 1687-9163. doi: 10.1155/2011/523806. URL <https://doi.org/10.1155/2011/523806>.
- [3] Selvakumar B i Muneeswaran K. Firefly algorithm based feature selection for network intrusion detection. *Computers & Security*, 81:148–155, 2019. ISSN 0167-4048. doi: <https://doi.org/10.1016/j.cose.2018.11.005>. URL <https://www.sciencedirect.com/science/article/pii/S0167404818303936>.
- [4] 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>.
- [5] 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>.

- [6] Andrey Dik. Population optimization algorithms: Firefly algorithm (fa), 2023. URL <https://www.mql5.com/en/articles/11873>.
- [7] Andrey Dik. Population optimization algorithms: Monkey algorithm (ma), 2023. URL <https://www.mql5.com/en/articles/12212>.
- [8] Leandro dos Santos Coelho i Viviana Cocco Mariani. Improved firefly algorithm approach applied to chiller loading for energy conservation. *Energy and Buildings*, 59:273–278, 2013. ISSN 0378-7788. doi: <https://doi.org/10.1016/j.enbuild.2012.11.030>. URL <https://www.sciencedirect.com/science/article/pii/S0378778812006445>.
- [9] Iztok Fister Jr, Xin-She Yang, Iztok Fister, i Janez Brest. Memetic firefly algorithm for combinatorial optimization. *arXiv preprint arXiv:1204.5165*, 2012.
- [10] 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.
- [11] 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.
- [12] 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>.
- [13] 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>.
- [14] Anand Nayyar, Dac-Nhuong Le, i Nhu Gia Nguyen. *Advances in swarm intelligence for optimizing problems in computer science*. CRC press, 2018.

- [15] B.K. Patle, Anish Pandey, A. Jagadeesh, i D.R. Parhi. Path planning in uncertain environment by using firefly algorithm. *Defence Technology*, 14(6):691–701, 2018. ISSN 2214-9147. doi: <https://doi.org/10.1016/j.dt.2018.06.004>. URL <https://www.sciencedirect.com/science/article/pii/S2214914718300333>.
- [16] Akash Sharma i Smriti Sehgal. Image segmentation using firefly algorithm. U *2016 International Conference on Information Technology (InCITE) - The Next Generation IT Summit on the Theme - Internet of Things: Connect your Worlds*, stranice 99–102, 2016. doi: 10.1109/INCITE.2016.7857598.
- [17] Nguyen Van Thieu i Seyedali Mirjalili. Mealpy: An open-source library for latest meta-heuristic algorithms in python. *Journal of Systems Architecture*, 2023. doi: 10.1016/j.sysarc.2023.102871.
- [18] Hui Wang, Wenjun Wang, Zhihua Cui, Xinyu Zhou, Jia Zhao, i Ya Li. A new dynamic firefly algorithm for demand estimation of water resources. *Information Sciences*, 438:95–106, 2018. ISSN 0020-0255. doi: <https://doi.org/10.1016/j.ins.2018.01.041>. URL <https://www.sciencedirect.com/science/article/pii/S0020025518300537>.
- [19] Xin-She Yang. Firefly algorithms for multimodal optimization. U Osamu Watanabe i Thomas Zeugmann, urednici, *Stochastic Algorithms: Foundations and Applications*, stranice 169–178, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-04944-6.
- [20] Ruiqing Zhao i Wansheng Tang. Monkey algorithm for global numerical optimization. *Journal of Uncertain Systems*, 2:165–176, 01 2008.
- [21] 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>.
- [22] Miodrag Zivkovic, Nebojsa Bacanin, Eva Tuba, Ivana Strumberger, Timea Bezdán, i Milan Tuba. Wireless sensor networks life time optimization based on the improved firefly algorithm. U *2020 International Wireless Communications and Mobile Computing (IWCMC)*, stranice 1176–1181, 2020. doi: 10.1109/IWCMC48107.2020.9148087.

## 6. Sažetak

Algoritam majmuna i algoritam krijesnica metaheuristički su optimizacijski algoritmi rojeva. U ovom radu dan je pregled samih algoritama, načini na koje se mogu izvesti na računalu i njihove primjene.