**Dokumentácia**

**Cat Swarm Optimization**

**Lukáš Gereg**

**Martin Matta**

**Barbora Kubalcová**

**Martina Marasová**

**Technická univerzita v Košiciach Katedra kybernetiky a umelej inteligencie**

**Fakulta elektrotechniky a informatiky Inteligentné systémy, 2. ročník**

Obsah

[Mačací algoritmus (CSO) – popis a inšpirácia 3](#_Toc121674748)

[Matematický model 4](#_Toc121674749)

[Hlavná časť algoritmu 4](#_Toc121674750)

[Pozorovací mód (seeking mode) 5](#_Toc121674751)

[Prenasledovací mód (tracing mode) 6](#_Toc121674752)

[Terminačné kritérium 6](#_Toc121674753)

[Ohodnocovacia funkcia 6](#_Toc121674754)

[SAT – ukážkový problém 7](#_Toc121674755)

[Trieda Cat 7](#_Toc121674756)

[Trieda SeekingMode 7](#_Toc121674757)

[Trieda TracingMode 8](#_Toc121674758)

[Trieda FitnessEvaluator 8](#_Toc121674759)

[Trieda CatSwarmOptimization 8](#_Toc121674760)

[main.py 9](#_Toc121674761)

[Príklady problémov, ktoré môžeme riešiť pomocou CSO 10](#_Toc121674762)

[OSSP (Open-shop scheduling problem) 10](#_Toc121674763)

[Popis problému 10](#_Toc121674764)

[Poskytovanie malých pôžičiek 10](#_Toc121674765)

[Problém obchodného cestujúceho (TSP) 10](#_Toc121674766)

[Experiment 11](#_Toc121674767)

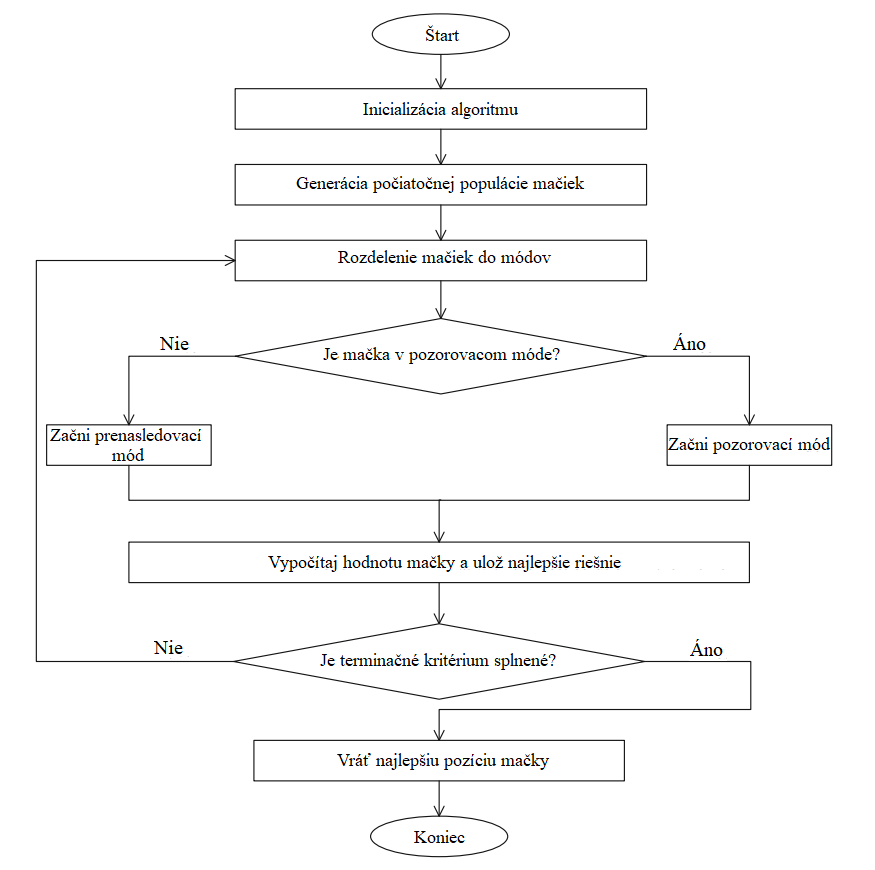
[Zdroje 12](#_Toc121674768)

# Mačací algoritmus (CSO) – popis a inšpirácia

CSO je algoritmus inšpirovaný správaním mačiek v reálnom svete. **Prírodné mačky** strávia väčšinu času v pokoji, sústredia sa na pozorovanie svojho okolia a striehnu na korisť. Aj keď sú v tejto fáze nečinné, sú veľmi ostražité a ostávajú v pozore. Ak zaznamenajú nejaký pohyb, pustia sa do akcie a začnú korisť prenasledovať. **Umelá mačka** má preto dva módy – pozorovací a prenasledovací.

* **Pozorovací mód** (seeking mode) - mačka stojí na mieste a pozoruje okolie
* **Prenasledovací mód** (tracking mode) - mačka prenasleduje korisť a mení svoju pozíciu

Na začiatku problému rozhodneme, koľko mačiek chceme v algoritme použiť – čím viac, tým lepšie. Vytvoríme populáciu *n* mačiek a náhodne ich rozmiestnime po *m*-dimenzionálnom priestore. Každá mačka v tomto priestore predstavuje jedno riešenie. Mačky následne rozdelíme do dvoch skupín – tie, čo pozorujú a tie, čo prenasledujú. Každú mačku následne ohodnotíme pomocou ohodnocovacej funkcie a uložíme najlepšiu mačku. Mačka predstavuje jedno potencionálne riešenie problému. Ako sa už spomínalo vyššie, väčšinu svojho času strávia pozorovaním, preto bude skupina pozorujúcich mačiek väčšia. Pomer mačiek v skupinách nám udáva **MR** (mixture ratio), ktorý si taktiež stanovíme na začiatku.



Obr. 1 – Vývojový diagram CSO algoritmu

## Matematický model

Každá mačka je umiestnená v *m*-dimenzionálnom priestore kandidátov, a teda má *m* súradníc, ktoré určujú jej pozíciu *Xn*. Ďalej má označenie, v ktorom z módov sa aktuálne nachádza, určitú rýchlosť *vn* pre každú dimenziu a hodnotu z ohodnocovacej funkcie. Posledné riešenie po ukončení algoritmu bude súčasne aj najlepším nájdeným riešením, keďže uchovávame doposiaľ najlepšie riešenie až do splnenia ukončovacej podmienky.

### Hlavná časť algoritmu

Postup algoritmu je znázornený na vývojom diagrame (Obr. 1). Slovný opis:

1. Vytvor inicializačnú populáciu mačiek, umiestni ich náhodne do *m*-dimenzionálneho priestoru a každej mačke priraď náhodnú rýchlosť.
2. Každej mačke priraď mód vo zvolenom pomere (MR).
3. Pre každú mačku vypočítaj hodnotu a do pamäte ulož najlepšiu mačku. Najlepšie riešenie predstavuje pozícia mačky.
4. Na základe prideleného módu, vykonaj hľadanie alebo pozorovanie.
5. Ak je splnené terminačné kritérium, ukonči proces, inak sa vráť ku kroku 2.

##### Pseudokód

Input: π, n n – počet mačiek, π - problém

Output: Xbest Є S Xbest – pozícia najlepšej mačky

Initialize n cats

**Repeat**

**For** i = 1 : n

Set mode for cat[i]

Evaluate cat[i]

**If** g(Xbest) < g(cat[i].Xi) **then**

Xbest = cat[i].Xi

**Endif**

**Endfor**

**For** i = 1 : n

**If** cat[i] is in seeking mode **then**

Start seeking mode

**Else**

Start tracing mode Mačka sa pohne k najlepšiemu riešeniu

**Endif**

**Endfor**

**Until** term()

**Return** Xbest

### Pozorovací mód (seeking mode)

V tomto móde mačka ostáva na mieste a pozoruje svoje okolie. V prípade detekcie koristi (poprípade nebezpečenstva) sa rozhodne zmeniť svoju polohu na základe štyroch faktorov:

* **SMP** (seeking memory pool) – počet vytvorených kópií (klonov) mačky
* **SRD** (seeking range of the selected dimension) – „ako ďaleko“ sa vie mačka pozrieť (posun v dimenzii). Ak sa niektorá z dimenzií bude meniť, tak rozdiel medzi novou a starou hodnotou bude SRD.
* **CDC** (counts of dimension to change) – koľko dimenzií bude zmenených
* **SPC** (self-position consideration) – boolovská hodnota. Rozhodnutie mačky ostať na aktuálnej pozícii alebo odísť.

#### Postup

1. Urob *i* počet kópií každej mačkyi, pričom *i* = SMP. Ak je SPC pravdivé, vytvor *i* = (SMP - 1) kópií mačky, pričom jedna ostane na pôvodnom mieste.
2. Na základe CDC vypočítaj novú pozíciu pre každú kópiu mačky pomocou rovnice

kde

*Xcn* je nová pozícia mačky,

*Xc* je aktuálna pozícia,

*R* je náhodné číslo z intervalu <0, 1>

1. Vypočítaj hodnoty mačiek (*FS*) pre ich nové pozície. Ak sú všetky hodnoty z ohodnocovacej funkcie rovnaké, nastav pravdepodobnosť výberu na 1 pre všetkých kandidátov, inak vypočítaj pravdepodobnosť výberu podľa vzorca

kde

*Pi* je pravdepodobnosť výberu aktuálneho kandidáta

*FSi* hodnota mačky z ohodnocovacej funkcie

*FSmax* maximálna hodnota ohodnocovacej funkcie

*FSmin*  minimálna hodnota ohodnocovacej funkcie

Riešime

*FSb = FSmax* pre minimalizačný problém

*FSb = FSmin* pre maximalizačný problém

1. Pomocou vybranej stratégie (roul-wheel, tournament, best fitness-self, random choice) náhodne vyber cieľový bod a zmeň pozíciu mačkyi.

V našej implementácii používame „best fitness-self“ a „random choice“ stratégiu“

* Random choice – úplne náhodný výber
* Best fitness-self – zoradíme mačky podľa výsledku ohodnocovacej funkcie od najhoršej po najlepšiu a vrátime najlepšiu mačku

### Prenasledovací mód (tracing mode)

Po tom ako mačka zaznamenala korisť v pozorovacom móde, zmenila svoj mód na prenasledovací a začína loviť. Mačka sa teda pohne nejakou rýchlosťou smerom ku koristi.

#### Postup

* + 1. Aktualizuj rýchlosť mačky v každej dimenzii (*vk,d*) na základe vzťahu

kde

*vk,d* je rýchlosť mačky v dimenzii *d*

*Xbest,d* je pozícia najlepšej mačky

*Xk,d* pozícia mačkyk

*c1* konštanta

*r1* náhodná hodnota z intervalu <0, 1>

* + 1. Skontroluj, či výsledná rýchlosť je v intervale <0, *vmax*>. Ak nie, nastav rýchlosť na *vmax*.
    2. Aktualizuj pozíciu mačky pomocou rovnice

kde

*Xk,d,new* je nová pozícia mačkyk v dimenzii *d*

*Xk,d,old* je aktuálna pozícia mačkyk v dimenzii *d*

### Terminačné kritérium

Terminačné kritérium nám určuje, kedy sa ma prehľadávanie priestoru kandidátov ukončiť. Jeho správne nastavenie je veľmi dôležité, pretože keď skončíme prehľadávanie skôr, nemusíme dostať najlepší výsledok. Ako terminačné kritérium sa môže brať počet iterácií, veľkosť zlepšenia, čas a podobne.

V našej implementácii riešime problém MAX SAT, preto je naším terminačným kritériom počet splnených klauzúl – buď všetky alebo čo najviac. V prípade, že by problém nemal riešenie (neboli by splnené všetky klauzuly), pridáme do terminačného kritéria aj maximálny počet iterácií.

### Ohodnocovacia funkcia

Funkcia dostane na vstupe jednu mačku a hľadá, koľko klauzúl je splnených pri danom nastavení pozície mačky. Vráti počet splnených klauzúl, pričom maximálna hodnota, ktorú môžeme dostať, je dĺžka listu s problémom.

# MAX SAT – ukážkový problém

SAT problém booleovský problém splniteľnosti. Na začiatku si zadefinujeme problém, respektíve požiadavky, ktoré chceme aby boli splnené. **Hľadáme také nastavenie premenných, aby boli splnené všetky klauzuly** (podmienky) alebo aby ich bolo splnených čo najviac. Klauzuly majú konjunktívno-normálnu formu, čo znamená, že je to konjunkcia disjunkcií (vnútri zátvorky je OR, medzi zátvorkami je AND).

## Trieda Cat

Reprezentuje jednu mačku. Konštruktor berie parametre:

* *problem\_size* – (int) počet dimenzií
* *seeking\_mode* – (bool) či je v pozorovacom móde

*def change\_position\_at\_index(index, value)*

Funkcia zmení pozíciu mačky na indexe o určitú hodnotu

* *index* – (int)index dimenzie, ktorá sa bude meniť
* *value* – (int) hodnota dimenzie sa nastaví na túto hodnotu

*Def evaluate\_fitness(fitness\_eval)*

Funkcia pridelí mačke ohodnotenie z ohodnocovacej funkcie. Vracia seba kvôli fluent API. Upravuje členskú premennú *fitness.*

* *fitness\_eval* – referencia na ohodnocovaciu trieda

*def copy\_self()*

Funkcia slúži na vytvorenie kópie samej seba. Neberie žiaden parameter. Funkcia vracia kópiu samej seba (mačky).

## Trieda SeekingMode

Trieda reprezentuje vlastnosti a správanie mačky v pozorovacom móde. Konštruktor berie parametre:

* *can\_stay* – (bool) či mačka ostane na mieste alebo nie (SPC)
* *num\_of\_looks* – (int) počet klonov mačky (SMP)
* *num\_of\_dimensions\_to\_change* – (int) počet dimenzií, ktoré chceme zmeniť (CDC)
* *distance* – (int) ako ďaleko vie mačka pozrieť (SRD)
* *fitness\_eval* – referencia na ohodnocovaciu triedu

*def begin\_strategy(cat)*

Funkcia spustí vykonávanie pozorovacieho módu. Vracia najlepšiu mutáciu mačky.

* *cat* – referencia na mačku

## Trieda TracingMode

Trieda reprezentuje vlastnosti a správanie mačky v prenasledovacom móde. Konštruktor berie parametre:

* *clauses\_count­* – (int) počet klauzúl v probléme

*def begin\_strategy(cat, best\_cat)*

Funkcia vykoná posun mačky smerom k doposiaľ najlepšej mačke. Vracia mačku s novou pozíciou.

* *cat* – referencia na mačku
* *best\_cat* – referencia na doposiaľ najlepšiu mačku (najlepšie riešenie)

## Trieda FitnessEvaluator

Trieda obsahujúca ohodnocovaciu funkciu. Konštruktor berie parameter:

* *problem* – referencia na problém

*def get\_fitness(cat)*

Funkcia dosadí pozíciu mačky do klauzúl. Vráti počet splnených klauzúl.

* *cat –* referencia ma mačku

## Trieda CatSwarmOptimization

Trieda obsahuje riešenie celého algoritmu. Konštruktor berie parametre:

* *maxsat­\_problem* – referencia na objekt problému (obsahuje klauzuly, veľkosť dimenzií a počet kandidátov)
* *cso\_setting* – referencia na slovník s počiatočným nastavením (SMP, SCS, SPC, SRD, MR, počet iterácií)

*def magic()*

Funkcia vykoná CSO algoritmus. Neberie žiaden parameter ani nič nevracia. Na konci svojho vykonávania vypíše do konzoly pozíciu najlepšej mačky, jej ohodnotenie z ohodnocovacej funkcie, ktorá určuje aj počet splnených klauzúl, a počet klauzúl v probléme. Funkcia vracia najlepšiu mačku.

## Trieda Testing

Trieda má za úlohu otestovať úspešnosť tohto algoritmu. Konštruktor berie parametre:

* *problem\_size* – (int) počet dimenzíí v problméme
* *clauses\_count* – (int) počet klauzúl
* *max\_changes\_in\_clause –* (int) koľko môže mať klauzula maximálne prvkov

*def generate\_clases()*

Funkcia vygeneruje problém o veľkosti *problem\_size* a s počtom klauzúl *clauses\_count*. Neberie žiaden parameter. Vracia problém (dictionary).

*def test(cso, test\_sample\_size)*

Funkcia vykoná testovanie na počte test\_*sample\_size* na *cso* algoritme. Po skončení vygeneruje štatistiku z výsledkov. Nič nevracia. Berie parametre:

* *cso* – (trieda Cso) referencia na triedu CatSwarmOptimization
* *test\_sample\_size* – (int) počet iterácií testovania

Príklad využitia triedy a výsledky experimentov sú na Obr. 2.

## main.py

V našej implementácií je problém definovaný nasledovne:

problem =

{

“size”: 5,

“count”: 7,

1: [0, 0, None, None, None],

2: [0, None, 0, None, None],

3: [None, 0, 0, None, None],

4: [None, None, None, 0, None],

5: [None, None, None, 0, 0],

6: [None, 1, 0, 1, None],

7: [0, None, None, 0, None],

}

„size” – počet dimenzií

„count“ – počet klauzúl

(1 – 7) – klauzuly, kde 0 znamená *not xn*, 1 znamená *xn* a None znamená, že táto dimenzia neovplyvňuje výsledok klauzuly. Klauzuly sú v konjunktívno-normálnej forme.

Napríklad klauzuly 1 a 2 vieme prepísať ako (not x1 OR not x2) AND (not x1 OR not x3).

Nastavenia algoritmu sú definované nasledovne:

Settings =

{

“iterations” : 5,

“smp” : 3,

“spc” : True,

“cdc” : 2,

“srd” : 1,

“mr” : 0.33,

“population\_size” : 3,

}

“iterations” - maximálny počet iterácií

“smp” – koľko krát sa môže mačka skopírovať

“spc” – či mačka ostane na mieste alebo nie

“cdc” – koľko dimenzií sa bude meniť pri mutácii

“srd” – vzdialenosť, na ktorú vie mačka pozerať

“mr” – mixture ratio – pomer rozdelenia mačiek do módov

“population\_size” - veľkosť populácie (počet mačiek)

# Experiment

V našom experimente sme skúšali meniť **MR** (mixture ratio) na rôzne zastúpenie mačiek v pozorovacom a prenasledovacom móde. Pri **MR = 0** sú všetky mačky v hľadacom móde, pri **MR = 1** sú všetky mačky v prenasledovacom móde. Vyskúšaná bola aj odporúčaná hodnota pomeru pre tento algoritmus, a to **MR = 0.2**. Všetky 3 varianty boli vyskúšané na   
3 mačkách, 80 dimenziách a v 100 000 iteráciách. Výsledky sú na obrázkoch Obr. 2 – 4.

#### Obrázok, na ktorom je text Automaticky generovaný popis

Obr. 2 – výpis z prebehnutých testov pre MR = 0

#### Obrázok, na ktorom je text Automaticky generovaný popis

Obr. 3 – výpis z prebehnutých testov pre MR = 1

Obrázok, na ktorom je text, snímka obrazovky, obrazovka

Automaticky generovaný popis

Obr. 4 – výpis z prebehnutých testov pre MR = 0.2

*Statistics for when each simulation ended:*

Počet iterácií potrebných na nájdenie riešenia. Máme maximálny počet iterácií, ktorý sme ochotní akceptovať pre nájdenie riešenia. Ak algoritmus nenájde optimálne riešenie do tohto počtu, vráti poslednú najlepšiu mačku.

*Numbers for finding optimal result:*

Koľko krát bol algoritmus optimálny (*True* - mačky našli lokálny extrém). Hodnota *False* znamená, že problém buď nemá úplné riešenie alebo ho mačky nenašli.

*Most cats ended in this position:*

Pozícia optimálnej mačky, ktorá sa vyskytovala najčastejšie.

### Výsledky

Experimentálne sme zistili, že najlepšie výsledky mal algoritmus pre nastavenie   
MR = 0.2, čo znamená, že približne 80% mačiek je v pozorovacom móde a približne 20% v prenasledovacom. Nastavenie MR = 0 (pozorovanie - seeking) bolo podstatne efektívnejšie ako pri MR = 1 (prenasledovanie - tracing), a to o takmer 13 000 riešení. Výsledok síce záležal na prvotnom rozložení mačiek v priestore, ale v konečnom dôsledku veľmi neovplyvnil efektívnosť čiastkových riešení.

# Príklady problémov, ktoré môžeme riešiť pomocou CSO

Prírodou inšpirované algoritmy sú veľmi efektívne pri riešení optimalizačných úloh, strojovom učení, dolovaní dát, pri výpočtových úlohách a podobných problémoch. CSO nie je výnimkou, a preto si ukážeme jeho využitie na vybraných problémoch.

## OSSP (Open-shop scheduling problem)

Tento problém je jeden z najznámejších komplexných optimalizačných problémov. Má niekoľko reálnych využití, ako napríklad testovanie čipov (sysem-on-chip testing), riešenie viacerých prístupov na jednej frekvencii pomocou rozdelenia signálu na viaceré časové sloty (area of satellite-switched time-division multiple access), výpočty vlnových dĺžok v optických sieťach a tak ďalej.

### Popis problému

Máme súbor *m* strojov *M = {M1, …, Mn}* a súbor *n* prác *{J1, …, Jn}.* Každá práca   
*Ji =* (*i Є* [1, *n*]) pozostáva z *m* operácií *Ji =* {*oi1­, oi2, …, oim*}, a každá operácia *oij* = (*mij, tij*) z práce *Ji* má byť vykonaná na stroji *Mj* (*j* Є [1, *m*]) v stanovenom čase *pij*.

Problém:

* Všetky operácie sa musia vykonať
* Každý stroj môže vykonávať v jednom okamihu len jednu operáciu
* Operácie v práci sa nesmú vykonávať súčasne

Cieľom je nájsť také nastavenie, aby čas vykonávania práce bol minimálny a aby boli splnené všetky vyššie stanovené podmienky.

## Poskytovanie malých pôžičiek

Pred tým ako banka niekomu požičia peniaze, zistia si jeho schopnosť tieto peniaze vrátiť. Napriek tomu banky (poskytovatelia pôžičky) nemajú garanciu bezproblémového vrátenia celej sumy. Na rozhodnutie či pôžičku poskytnúť alebo nie napomáhajú rôzne algoritmy – medzi nimi aj CSO. Dokonca sa ukázalo, že analýza pomocou tohto algoritmu je presnejšia, ako štandardné metódy analýzy.

## Problém obchodného cestujúceho (TSP)

V tomto probléme máme *n* miest, všetky sú navzájom prepojené a každé prepojenie medzi jednotlivými mestami má svoju cenu. Úlohou je navštíviť všetky mestá a vrátiť sa späť do východiskového bodu za čo najnižšiu cenu cesty.

Ohodnocovacia funkcia by v tomto prípade vracala cenu cesty pre každú mačku a hľadali by sme mačku s najmenšou cenou.

# Zdroje

Bozorg-Haddad, O. (Ed.). (2018). Advanced optimization by nature-inspired algorithms (Vol. 720). Singapore: Springer.

Chu, Shu-Chuan & Tsai, Pei-Wei & Pan, Jeng-Shyang. (2006). Cat Swarm Optimization.   
854-858. 10.1007/11801603\_94.

<https://www.geeksforgeeks.org/cat-swarm-optimization/>

Bouzidi, A., Riffi, M.E. & Barkatou, M. Cat swarm optimization for solving the open shop scheduling problem. J Ind Eng Int 15, 367–378 (2019). <https://doi.org/10.1007/s40092-018-0297-z>

<https://www.degruyter.com/document/doi/10.1515/jisys-2022-0018/html>