Stiskanje slik z algoritmi po vzorih iz narave

Gregor Jurgec
Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko
Smetanova 17, Maribor
gregor.jurgec@gmail.com

Iztok Fister
Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko
Smetanova 17, Maribor
iztok.fister@um.si

POVZETEK

V članku predstavljamo postopek stiskanja slik na inovativen način s pomočjo optimizacijskih algoritmov iz dveh različnih družin, tj. evolucijskih algoritmov in algoritmov na osnovi inteligence rojev. V naši raziskavi za reševanje problema stiskanja slik uporabljamo diferencialno evolucijo in optimizacijo s kukavičjim iskanjem. Osnovna algoritma izboljšamo z adaptacijskimi in hibridizacijskimi metodami. Stiskanje slik izvedemo s pomočjo množice primitivnih geometrijskih oblik trikotnikov. Razvita algoritma testiramo na realnih primerih stiskanja slik in kakovost rezultatov stiskanja primerjamo med seboj. Z izboljšanima algoritmoma dobimo zelo dobre rezultate stisnjenih slik, saj podobnost teh slik v primerjavi z izvornimi slikami v najslabšem primeru presega 89,70 %, v najboljšem primeru pa dosežemo 92,08 %.

Kjučne besede

evolucijski algoritmi, inteligenca rojev, diferencialna evolucija, kukavičje iskanje, stiskanje slik

1. UVOD

Z razvojem naprednih senzorjev za zajem slik visoke ločljivosti se povečuje velikost zajetih posnetkov. Pri tem pridemo do problema shranjevanja slik, saj nestisnjene slike zavzemajo ogromno prostora na spominskih medijih. Da prihranimo prostor, uporabljamo različne optimizacijske algoritme, ki stisnejo podatke na račun izgube informacij o sliki. Za te izgube seveda želimo, da so čim manjše. Algoritmi stiskanja slik iščejo podobnosti v podatkih, oz. skušajo podatke, zajete s senzorji, spraviti v čim kompaktnejšo obliko.

Glavna razlika med evolucijskimi algoritmi in algoritmi na osnovi inteligence rojev je, da evolucijski algoritmi [4] pri optimizaciji simulirajo naravno evolucijo z operatorji mutacije, križanja in selekcije [3], medtem ko delovanje algoritmov na osnovi inteligence rojev temelji na obnašanju bioloških vrst (npr. živali ali žuželk), ki rešujejo vsakodnevne probleme porazdeljeno in kolektivno [1]. S kombinacijo značilnosti obeh družin lahko razvijemo hibridne algoritme, ki so sposobni reševanja tudi kompleksnejših optimizacijskih problemov. S hibridizacijo različnih delov algoritma dodamo osnovnemu algoritmu domensko specifično znanje domene [4, 5].

V tem članku rešujemo problem stiskanja slik z izboljšanima optimizacijskima algoritmoma diferencialne evolucije [10] in kukavičjega iskanja [11] sposobna generiranja slik, ki zasedajo manj prostora, ampak slabše kakovosti. Sliko predstavljamo s trikotniki narisanimi na platno, ki so določeni s koordinatami oglišč in barvo. Trikotniki so učinkoviti za to nalogo saj pokrivajo več slikovnih pik kot ena sama točka in omogočajo prekrivanje in prosojnost barv v določeni regiji. Naš cilj je, da bi bila izrisana slika čim bolj podobna originalni sliki. V tem članku se zgledujemo in nadgrajujemo delo iz [13, 7], kjer s pomočjo trikotnikov in samo adaptivne diferencialne evolucije oziroma genetskega algoritma poskušajo zgraditi približen model slike.

S tem delom želimo pokazati, da lahko z izboljšanima algoritmoma na osnovi obnašanja kukavic in diferencialne evolucije uspešno stisnemo izvorno sliko. Pri tem lahko ugotovimo, da sta oba algoritma primerna za stiskanje slik s trikotniki, tj. na inovativen način in s pomočjo novih algoritmov, po vzoru iz narave. Tak način stiskanja slik lahko postane dobra alternativa obstoječim algoritmom stiskanja slik. Seveda ti algoritmi potrebujejo še dodatne izboljšave, vendar so se kot osnova izkazali zelo dobro, kar dokazujejo tudi rezultati v nadaljevanju.

Struktura članka v nadaljevanju je naslednja. Drugo poglavje se ukvarja s podanimi splošnimi informacijami o kodiranju slik. V tretjem in četrtem poglavju predstavimo oba izboljšana algoritma za stiskanje slik. Peto poglavje primerja rezultate stiskanja realnih slik z izboljšanima algoritmoma.

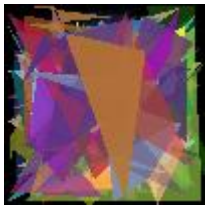
2. KODIRANJE SLIKE

V našem delu uporabljamo izgubni način stiskanja slik, ker imajo naše stisnjene slike nepravilnosti in so popačene. Naša slika je predstavljena z množico barvnih trikotnikov, narisanih na črno podlago, ki je iste velikosti kot originalna slika. Množica trikotnikov predstavlja sliko, podobno originalni. Vsak trikotnik je sestavljen iz treh točk s koordinatami x, y in barvo, s katero je obarvan, v barvnem prostoru rdečezeleno-modro-alfa (angl. Red, Green, Blue, Alpha, krajše RGBA). Zelo pomembno je, da uporabimo prosojnost tri-

kotnikov, s čimer dosežemo prekrivanje trikotnikov in s tem dobimo veliko različnih barv, ki sestavljajo sliko.

Množico barvnih trikotnikov, ki sestavljajo sliko, kodiramo v vektor na naslednji način: vektor $\vec{x} = (\vec{T}_1, \vec{T}_2, \vec{T}_3, ..., \vec{T}_{\max})$ predstavlja množico trikotnikov $\vec{T}_i, \forall i \in \{1, ..., \max\}$, kjer max določa število trikotnikov, ki sestavljajo sliko. Vsak od teh trikotnikov je kodiran v vektor $\vec{T}_i = (x_1, y_1, x_2, y_2, x_3, y_3, R, G, B, A)$, ki je sestavljen iz celih števil, kjer pari $x_i, y_i, \forall i \in \{1, 2, 3\}$ predstavljajo koordinatne točke oglišč trikotnika, skalarji R, G, B barvo in A prosojnost trikotnika.

Slika 1 prikazuje množico prekrivajočih barvnih trikotnikov na črnem ozadju v zgodnji fazi delovanja optimizacijskih algoritmov.



Slika 1: Osnovni prikaz slike s trikotniki.

2.1 Optimizacijski problem in kriterijska funkcija

Optimizacija je postopek iskanja najboljših vhodnih parametrov pri znanem modelu in znanih izhodnih parametrih [6, 4]. Optimizacijski problem P definiramo kot četvorko

$$P = \langle I, S, f, cilj \rangle,$$

kjer pomenijo: I množico vseh nalog, S množico dopustnih rešitev problema, f kriterijsko funkcijo (angl. objective function), s katero ocenjujemo kakovost rešitve, in *cilj*, s katerim povemo ali kriterijsko funkcijo maksimiziramo ali minimiziramo. Vrednosti kriterijske funkcije so lahko skalarji ali v primeru več-kriterijskih problemov vektorji [9, 12].

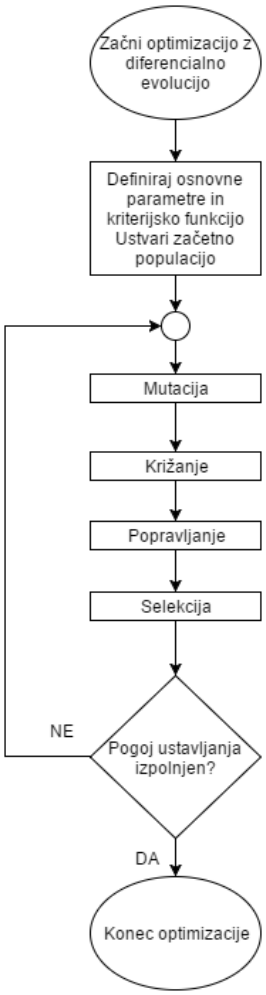
Kriterijska funkcija v našem primeru primerja vsako isto ležečo slikovno piko generirane slike s slikovno piko na originalni sliki. Vsako slikovno piko primerjamo po vseh treh barvnih komponentah R, G, B. Vsoto razlik isto ležečih slikovnih pik pri vsaki od treh komponent delimo s produktom med višino in širino slike ter z vsemi možnimi odtenki vsake od barv RGB. Z rezultatom dobimo vrednost različnosti slik, ki ga odštejemo od ena in dobimo podobnost med slikama. To pomnožimo s 100 % in ta vrednost potem predstavlja podobnost najdene slike v primerjavi z originalno v odstotkih. Naš cilj je, da je podobnost čim večja in zato moramo našo kriterijsko funkcijo maksimizirati.

Naša kriterijska funkcija $f(\vec{x})$ je opisana z enačbo 1, kjer $\vec{x}^*$ predstavlja originalno, referenčno sliko, $\vec{x}$ pa stisnjeno, generirano sliko. Indeksa $x_i, \forall i \in \{1, ...višinaSlike\}$ in $y_j, \forall j \in \{1, ...širinaSlike\}$ določata trenutni položaj opazovane slikovne pike. Oznake R, G in B pa definirajo, v kateri komponenti barve RGB se nahajamo. Konstanta 255 določa število različnih odtenkov barve.

$$f(\vec{x}) = 100\% \cdot \left(1 - \left(\frac{\sum_{i=1}^{višinaSlike} \sum_{j=1}^{širinaSlike} \left| \vec{x}^{*}_{i,j}{}^{R} - \vec{x}_{i,j}{}^{R} \right|}{255 \cdot višinaSlike \cdot širinaSlike} + \frac{\sum_{i=1}^{višinaSlike} \sum_{j=1}^{širinaSlike} \left| \vec{x}^{*}_{i,j}{}^{G} - \vec{x}_{i,j}{}^{G} \right|}{255 \cdot višinaSlike \cdot širinaSlike} + \frac{\sum_{i=1}^{višinaSlike} \sum_{j=1}^{širinaSlike} \left| \vec{x}^{*}_{i,j}{}^{B} - \vec{x}_{i,j}{}^{B} \right|}{255 \cdot višinaSlike \cdot širinaSlike} \right) \right) \quad (1)$$

3. DIFERENCIALNA EVOLUCIJA

Diferencialna evolucija (angl. Differential Evolution, krajše DE) rešuje optimizacijski problem z izboljševanjem posameznih rešitev iz populacije. Iz obstoječih rešitev tvori poskusne rešitve (angl. candidate solution) z dodajanjem razlike naključno izbranih vektorjev (mutacijska strategija), te pa je potrebno vrednotiti. Vrednosti elementov vektorjev poskusne rešitve, ki so izven definiranih območij popravimo na zgornje ali spodnje meje v operaciji popravljanja. Če je poskusna rešitev boljša od obstoječe, ta rešitev zamenja obstoječo v populaciji. Ta postopek ponavljamo za vsakega posameznika v populaciji rešitev v vsaki generaciji algoritma, dokler ni izpolnjen pogoj zaustavljanja [10]. Model diferencialne evolucije je prikazan na sliki 2.



Slika 2: Model diferencialne evolucije

Psevdo-kod originalnega algorithma DE prikazuje algoritem 1.

Algoritem 1 Algoritem DE

```
inicializacija;
najboljsaVrednost^0 ← oceniVseResitve;
while pogojUstavljanjaIzpolnjen do
    for i ← 1, N_p do
        x_i^p ← generirajPoskusnoResitev(x_i^t);
        poskusnaVrednost ← oceniPoskusnoResitev(x_i^p);
        if poskusnaVrednost > f(x_i^t) then
            x_i^t ← x_i^p;
            najboljsaVrednost_i^t ← poskusnaVrednost;
        end if
    end for
end while
```

3.1 Samoprilagajanje nadzornih parametrov v diferencialni evoluciji (jDE)

V literaturi najdemo veliko različnih rešitev, kako nastaviti nadzorna parametra F in C_r. Izbira primernih vrednosti nadzornih parametrov je odvisna od problema, ki ga rešujemo. Da se izognemo temu, so avtorji Brest in ostali v [2] predlagali izboljšano verzijo algoritma DE, ki nadzorna parametra F in C_r samo-prilagaja med samim izvajanjem. Poimenovali so ga algoritem jDE. Vsak posameznik v populaciji je razširjen z vrednostjo omenjenih parametrov, ki se prilagajata in izboljšujeta skozi evolucijo in posledično izboljšata posameznike v poskusni populaciji. Nova nadzorna parametra F_i^p in ${C_r}_i^p$ izračunamo pred mutacijo, tako da imata vpliv na operatorje DE in izračun poskusne rešitve v trenutni generaciji, po enačbah:

$$F_i^p = \begin{cases} F_l + U_1(0,1) \cdot F_u & \text{če } U_2(0,1) < \tau_1, \\ F_i^G & \text{drugače}, \end{cases}$$

$${C_r}_i^p = \begin{cases} U_3(0,1) & \text{če } U_4(0,1) < \tau_2, \\ {C_r}_i^t & \text{drugače}, \end{cases}$$

kjer je $U_j(0,1)$, $j \in \{1,2,3,4\}$ naključna uniformna vrednost iz intervala [0,1] ter τ_1 in τ_2 verjetnosti za spremembo parametrov F in C_r (tudi hitrosti učenja). Avtorji zanju predlagajo vrednosti $\tau_1 = \tau_2 = 0,1$. Vrednosti $F_l = 0,1$ in $F_u = 0,9$ zagotavljata, da je novi parameter F vedno naključno v intervalu [0.1, 1.0]. Novi parameter C_r pa zavzema vrednosti v intervalu [0,1] [13].

3.2 Modificirana diferencialna evolucija

Modificirana verzija diferencialne evolucije (krajše MoDE) za stiskanje slik vsebuje več prilagoditev in izboljšav. Najprej prilagodimo osnovni algoritem na obstoječ algoritem jDE, ki je opisan v prejšnjem poglavju. Nato razširimo jDE algoritem s strategijo $'DE/best/1/bin'$, s katero dosežemo, da se posamezniki približujejo najboljši rešitvi, okrog katere bi lahko bil globalni maksimum. V primeru, da pride do izgube raznolikosti v populaciji, uporabimo ponovno inicializacijo večine posameznikov, s čimer raznolikost populacije povečamo.

3.2.1 Kombiniranje mutacijskih strategij 'DE/rand/1/bin' in 'DE/best/1/bin'

Pri tej izboljšavi gre za izbiro mutacijske strategije. Kombiniramo med strategijo $'DE/rand/1/bin'$ in $'DE/best/1/bin'$. Privzeto uporabljamo strategijo $'DE/rand/1/bin'$. Verjetnost izbire strategije $'DE/best/1/bin'$ v vsaki generaciji pri vsakem posamezniku v populaciji je 10^{-5}, ki smo jo določili po občutku skozi testiranja. Izbiro strategije in s tem baznega vektorja uvedemo pred začetkom mutacije.

$$IzbiraStrategije = \begin{cases} 'DE/best/1/bin' & \text{če } U(0,1) < 10^{-5}, \\ 'DE/rand/1/bin' & \text{drugače}. \end{cases}$$

V zgornji enačbi predstavlja $U(0,1)$ naključno število iz intervala [0,1] iz uniformne porazdelitve. Vektor mutacije $\vec{m}_i$ pri strategiji $'DE/best/1/bin'$ izračunamo po enačbi:

$$\vec{m}_i = \vec{x}_{best}^t + F \cdot \left(\vec{x}_{r1}^t - \vec{x}_{r2}^t\right),$$

kjer $\vec{x}_{best}^t$ označuje najboljšega posameznika v trenutni populaciji, F je krmilni parameter, $\vec{x}_{r1}^t$ in $\vec{x}_{r2}^t$ pa sta naključno izbrana vektorja iz trenutne populacije, kjer $r1 \neq r2$. Vektor mutacije $\vec{m}_i$ pri privzeti strategiji $'DE/rand/1/bin'$ izračunamo po enačbi:

$$\vec{m}_i = \vec{x}_{r1}^t + F \cdot \left(\vec{x}_{r2}^t - \vec{x}_{r3}^t\right).$$

3.2.2 Ponastavitev posameznika

Da se izognemo lokalnemu optimumu, uporabimo mehanizem ponastavitve posameznika (angl. random restart) z naključnimi vrednostmi elementov pri operaciji križanja. Križanje je prikazano z naslednjo enačbo:

$$k_{i,j} = \begin{cases} x_{\min,j} + \left(x_{\max,j} - x_{\min,j}\right) \cdot U(0,1) & \text{če } U_j(0,1) < C_r \\ & \vee\, j_{rand} = j \wedge U_j(0,1) < 10^{-5}, \\ m_{i,j} & \text{če } U_j(0,1) < C_r \vee j_{rand} = j, \\ x_{i,j}^t & \text{drugače}. \end{cases}$$

Če je pri križanju v vsaki iteraciji dimenzije problema slučajno število $U(0,1)$ pod pragom 10^{-5}, takrat celoten vektor $\vec{k}_i$ inicializiramo na začetne vrednosti in prekinemo križanje, drugače pa na pozicijo, ki jo označuje indeks j, v vsaki iteraciji kopiramo vrednost vektorja mutacije $\vec{m}_i$ ali vrednost trenutnega posameznika $\vec{x}_i^t$ z istim indeksom.

3.2.3 Razpršenost populacije in ponastavitev posameznikov

Po selekciji izračunamo razpršenost populacije. Razpršenost populacije služi kot merilo raznolikosti populacije. Izračunamo jo s standardnim odklonom od povprečne vrednosti kriterijske funkcije posameznikov v populaciji, ki ga izračunamo po enačbi:

$$povprečje = \frac{\sum_{i=1}^{N_p} fitnes_i}{N_p},$$

kjer seštejemo vse vrednosti kriterijske funkcije posameznikov v populaciji in jih delimo s številom posameznikov N_p. Povprečje uporabimo pri izračunu standardnega odklona v enačbi:

$$stdOdklon = \sqrt{\frac{\sum_{i=1}^{N_p} \left(povprečje - fitnes_i\right)^2}{N_p}}.$$

Če je standardni odklon oziroma razpršenost populacije manjša od praga, ki ga določimo pri testiranju, ponastavimo 90 % najslabših posameznikov na začetne vrednosti in s tem povečamo raznolikost populacije.

4. KUKAVIčJE ISKANJE

V nadaljevanju predstavljamo osnovni algoritem s kukavičjim iskanjem [11] (angl. Cuckoo Search, krajše CS), kjer je vhod v algoritem začetna populacija, sestavljena iz posameznikov inicializiranih z naključnimi vrednostmi elementov, ki predstavljajo rešitve. Izhod algoritma je najboljša rešitev oz. najboljši posameznik glede na kriterijsko funkcijo. Osnovni algoritem vsebuje inicializacijo in glavno zanko, ki teče do pogoja ustavljanja. Za vsakega posameznika iz populacije najprej generiramo novo rešitev z distribucijo Levy in jo ocenimo s kriterijsko funkcijo. Iz populacije izberemo naključnega posameznika, njegovo vrednost kriterijske funkcije primerjamo z vrednostjo kriterijske funkcije potencialne nove rešitve in tega zamenjamo z novo rešitvijo v primeru, da je ta boljša. Sledi zamenjava najslabšega posameznika s posameznikom, ki ga dobimo z naključno inicializacijo posameznika v primeru, da je naključno generirano število iz intervala $[0, 1] < p_\alpha$. Na koncu vsake iteracije poiščemo in shranimo najboljšo rešitev v populaciji v globalno rešitev, ki je izhod algoritma. Model algoritma s kukavičjim iskanjem je prikazan na sliki 3.

Psevdo-kod originalnega algorithma CS prikazuje algoritem 2.

Algoritem 2 Algoritem CS

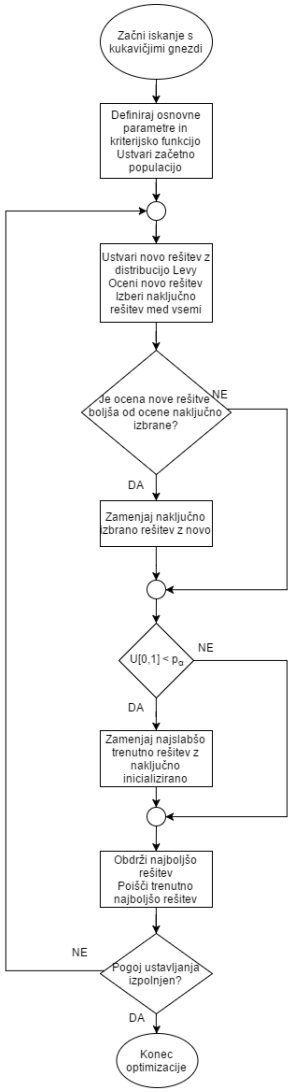
1: inicializacija;
2: $\overrightarrow{najboljsaVrednost}^0 \leftarrow$ oceniVseResitve;
3: **while** pogojUstavljanjaIzpolnjen **do**
4: **for** $i \leftarrow 1, N_p$ **do**
5: $\vec{x}_i^p \leftarrow$ generirajPoskusnoResitev($\vec{x}_i^t$);
6: $poskusnaVrednost \leftarrow$ oceniPoskusnoResitev($\vec{x}_i^p$);
7: $j_{rand} \leftarrow U(1, N_p)$;
8: **if** $poskusnaVrednost > najboljsaVrednost^t_{j_{rand}}$ **then**
9: $\vec{x}^t_{j_{rand}} \leftarrow \vec{x}_i^p$;
10: $najboljsaVrednost^t_{j_{rand}} \leftarrow poskusnaVrednost$;
11: **end if**
12: **if** $U(0, 1) < p_\alpha$ **then**
13: $inicializacija(\vec{x}^t_{najslabsi})$;
14: **end if**
15: shraniGlobalnoNajboljsoResitev;
16: **end for**
17: **end while**

4.1 Modificirano kukavičje iskanje

Modificirana verzija CS (krajše MoCS) vsebuje dve glavni izboljšavi. Najprej razširimo algoritem CS z generiranjem N_p kukavičjih jajc v eni iteraciji, nato uvedemo hibridizacijo z mutacijsko strategijo DE.

4.1.1 Razširitev algoritma CS

Najprej generiramo N_p novih rešitev (kukavičja gnezda) in jih ocenimo s kriterijsko funkcijo. Vrednosti kriterijske funkcije novih rešitev primerjamo z vrednostmi kriterijske funkcije rešitev iz trenutne populacije na vsakem indeksu od 1 do N_p in nove rešitve zamenjamo z rešitvami iz trenutne populacije v primeru, da so te boljše. Razlika z osnovnim



Slika 3: Model algoritma s kukavičjim iskanjem

algoritmom je v tem, da generiramo N_p novih rešitev in ne izbiramo naključne rešitve za primerjavo z novo rešitvijo, ampak primerjamo kar rešitve z istim indeksom.

4.1.2 Hibridizacija z mutacijsko strategijo DE

Odkrite posameznike modificiramo po vzoru mutacijske strategije DE [8]. Tu gre za neke vrste pristransko naključno iskanje, kjer razliko naključno izbranih posameznikov prištejemo rešitvi posameznika, ki ga trenutno zavračamo. Naključne posameznike izberemo tako, da generiramo dve polji permutacij dolžine N_p, nato izračunamo razliko vektorjev z istim indeksom in vsakokrat dobimo drugo razliko. Modificirano naključno iskanje izrazimo z naslednjo enačbo:

$$\vec{x}_i^{t+1} = \vec{x}_i^t + U(0,1) \times H(p_\alpha - \varepsilon) \times \left(\vec{x}_j^t - \vec{x}_k^t\right),$$

kjer sta $\vec{x}_j^t$ in $\vec{x}_k^t$ naključno izbrana posameznika iz populacije, $U(0,1)$ funkcija, ki vrne naključno realno število z uniformno porazdelitvijo iz intervala [0,1], $H(p_\alpha - \varepsilon)$ je funkcija Heaviside, s katero odločimo, ali zavržemo ali obdržimo novega posameznika, katere rezultat je 0, če je $p_\alpha - \varepsilon < 0$

in 1, če je $p_\alpha - \varepsilon \geq 0$, p_α je parameter zamenjave, ε naključno število iz uniformne distribucije iz intervala [0,1] in $\vec{x}_i^t$ je izbrani vektor iz populacije.

5. REZULTATI

Testiranje algoritmov smo izvedli v dveh korakih. V prvem koraku smo poiskali najboljše parametre velikosti populacije in števila trikotnikov med izbranimi vrednostmi za stiskanje vsake izmed izbranih slik pri 10^5 vrednotenjih. Ko smo najboljše parametre našli, smo stiskanje s temi parametri ponovili, le da smo v drugem koraku povečali število vrednotenj na 10^7 in tako dobili najboljše rezultate z izbranimi vrednostmi. Rezultate stiskanja slik predstavljamo v nadaljevanju.

5.1 Izbira slik in iskanje parametrov

Najprej smo določili tri referenčne, originalne slike, ki smo jih stiskali z našima algoritmoma. Zaradi zahtevnosti kriterijske funkcije smo vsako od slik pomanjšali na resolucijo 100x100. Raznolikost referenčnih slik, prikazanih na sliki 4, smo dosegli z izbiro dveh barvnih in ene sivinske slike.

Slika 4: Izbrane slike za stiskanje.

Izbira optimalnih parametrov algoritmov je zelo pomembna, saj ti vplivajo na pravilno delovanje algoritma, s tem pa na kakovost končnih slik in hitrost konvergence. Prav zaradi pomembnosti parametrov algoritmov smo najprej poiskali najboljše parametre vsakemu od algoritmov za stiskanje vsake slike posebej pri maksimalno 10^5 vrednotenjih kriterijske funkcije. Poudariti moramo, da smo kot ustavitveni pogoj v algoritmih vedno uporabili število vrednotenj kriterijske funkcije, s čimer smo zagotovili poštenost meritev in rezultatov. Naslednja dva parametra sta bila še velikost populacije, ki smo jo izbirali iz množice $N_p = \{50, 100, 150, 200\}$, in število trikotnikov, uporabljenih pri stiskanju slik. Te smo izbirali s pomočjo porazdelitve eksponentne funkcije:

$$stTrikotnikov = 20 \times 2^x, \quad \text{za } x = \{0, 1, 2, 3, 4\},$$

kjer smo dobili naslednja dopustna števila trikotnikov:

$$stTrikotnikov = \{20, 40, 80, 160, 320\}.$$

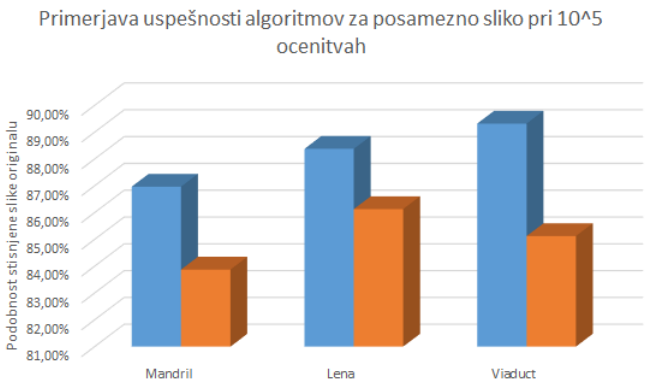
Tabela 1: Izbrani najboljši parametri

Ime slike	MoDE		MoCS	
	N_p	N_t	N_p	N_t
Mandril	50	20	50	80
Lena	50	20	50	20
Viaduct	50	80	50	40

Pri vsakem zagonu smo kombinirali vse možne kombinacije parametrov velikosti populacije in števila trikotnikov, s čimer smo izvedli 20 neodvisnih zagonov algoritma za eno sliko. Za vsako sliko smo postopek ponovili desetkrat, tj. 200 neodvisnih zagonov. Ker imamo dva algoritma in tri slike, se je skupaj izvedlo kar 1200 neodvisnih zagonov algoritma.

Vsak algoritem krmilimo z nadzornimi parametri. Pri DE smo izbrali začetne vrednosti za $F = 0,5$ in $C_r = 0,9$. Sama nastavitev teh parametrov ne igra posebne vloge, saj ima naš algoritem vključeno samo-prilagajanje teh parametrov. Nekoliko drugače je pri CS, saj moramo nastaviti parameter zamenjave p_α, ki je fiksen skozi celoten algoritem in smo ga v našem primeru nastavili na 0,25. Kot najboljše nastavljive parametre (tabela 1) smo izbrali tiste, ki so imeli največjo povprečno vrednost kriterijske funkcije in ne morebitne posamezne najboljše rešitve z največjo vrednostjo kriterijske funkcije.

Rezultati prvega dela testiranja, ko smo iskali najboljše parametre za algoritma, so pokazali dobre rezultate, ki jih lahko še izboljšamo s povečanjem števila vrednotenj kriterijske funkcije. Slika 5 prikazuje, kako uspešna sta bili naša algoritma pri stiskanju slik pri iskanju najboljših nastavljivih parametrih do 10^5 vrednotenj kriterijske funkcije.



Slika 5: Podobnost rezultatov algoritmov MoDE in MoCS za stiskanje slik pri 10^5 vrednotenjih.

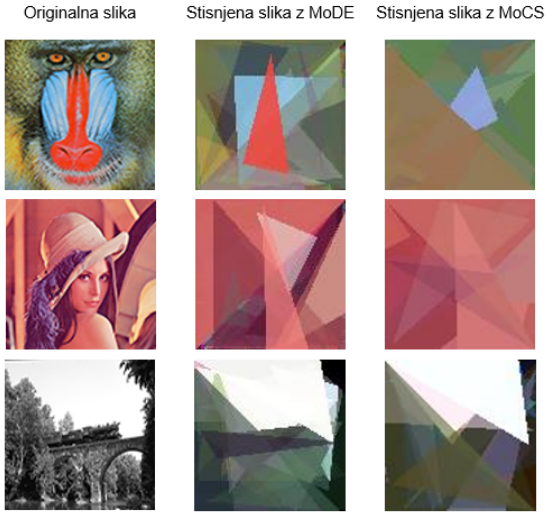
Iz slike 5 lahko razberemo, da so rezultati MoDE nekoliko boljši v primerjavi z MoCS za vsako sliko pri 10^5 vrednotenjih. Za sliko Mandril je MoDE dosegel za 3,09 % večjo podobnost originalni sliki v primerjavi z MoCS, za sliko Lena 2,25 % in za sliko Viaduct 4,20 %.

Slika 6 prikazuje originalne in najboljše stisnjene slike iz prvega koraka testiranja z algoritmoma MoDE in MoCS. Iz slik je razvidno, da v tej fazi testiranja stiskanja slik pri 10^5

Tabela 2: Rezultati podobnosti stisnjene slike in čas stiskanja z 10^7 vrednotenji.

Ime slike	MoDE		MoCS	
	Podobnost	Čas stiskanja	Podobnost	Čas stiskanja
Mandril	90,08 %	4h 41m 11s	89,70 %	8h 12m 45s
Lena	91,92 %	4h 48m 35s	90,31 %	5h 43m 38s
Viaduct	92,08 %	5h 51m 31s	90,56 %	6h 26m 24s

vrednotenjih MoCS ne dosega tako dobrih rezultatov kot MoDE.



Slika 6: Primerjava originalnih slik s stisnjenimi slikami z algoritmoma MoDE in MoCS z 10^5 vrednotenji kriterijske funkcije.

5.2 Stiskanje slik z 10^7 vrednotenji kriterijske funkcije

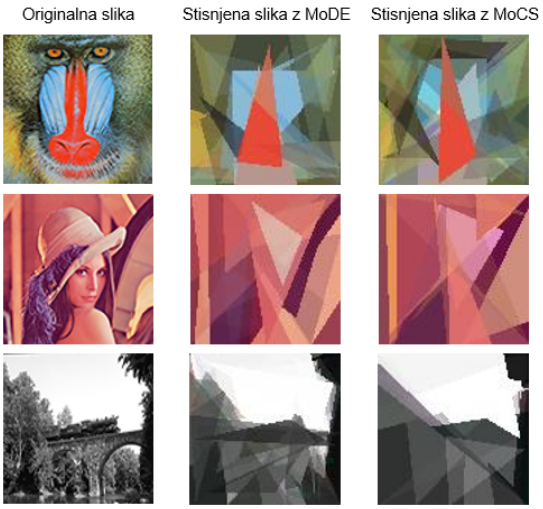
Po izvedbi testiranja, s katerim smo dobili najboljše nastavitve parametrov, smo te uporabili pri časovno zahtevnejšem testiranju stiskanja slik. Število vrednotenj kriterijske funkcije smo povečali iz 10^5 na 10^7 in ustrezno nastavili najboljše parametre velikosti populacije N_p in število trikotnikov N_T, kot jih prikazuje tabela 1.

Rezultati, prikazani v tabeli 2, so bili veliko boljši kot pri stiskanju z 10^5 vrednotenji. Algoritem MoDE je tudi pri tem številu vrednotenj boljše stisnil vsako izmed slik, kot je to storil algoritem MoCS.

Iz slike 7 je jasno razvidno, da je razvit algoritem MoDE boljši algoritem za stiskanje slik kot razvit algoritem MoCS. V stisnjenih slikah z algoritmom MoDE so jasno vidne linije objektov in njihove barve. Pri stisnjenih slikah z algoritmom MoCS so te linije nekoliko zabrisane in v primerjavi z MoDE so nekateri objekti združeni v enega samega.

6. ZAKLJUčEK

Razvili smo dva algoritma za stiskanje slik po vzorih iz narave. Podrobno smo opisali nekatere od možnih izboljšav algoritma DE in CS, ki pomagajo pri konvergenci algoritma. Rezultati izboljšanih algoritmov dokazujejo, da ti algoritmi



Slika 7: Primerjava originalnih slik s stisnjenimi slikami z algoritmoma MoDE in MoCS z 10^7 vrednotenji kriterijske funkcije.

zaenkrat še niso primerni za naš problem stiskanja slik, saj ne dajejo rezultatov, ki bi bili sprejemljivi v praksi. Pričakujemo lahko, da bosta omenjena algoritma na podlagi dodatnih izboljšav postala primerna tudi za te namene. Trenutno je stiskanje s temi algoritmi primerno za znanstvene namene in poskuse, ker lahko pokažemo, da so ti algoritmi po vzorih iz narave splošno namenski.

Rezultati stisnjenih slik so po oceni naše kriterijske funkcije glede podobnosti z originalno sliko zelo dobri. Dosegajo podobnost do dobrih 92 %, kar se zdi zelo obetavno. Ko pogledamo stisnjene slike, pa lahko opazimo, da so na slikah vidni le robovi slike in da so barve poligonov, ki jih ti robovi omejujejo, podobne istemu področju originalne slike. Iz samih stisnjenih slik lahko brez težav razberemo, kateri originalni sliki pripada, same objekte pa je že težje prepoznati.

6.1 Možnosti za izboljšave, nadgradnje

Obstaja več možnosti, s katerimi bi lahko uspešnost naših algoritmov še dodatno izboljšali. V naslednjih nekaj odstavkih predstavljamo samo nekatere izmed njih.

Lotili bi se lahko pred-procesiranja slik, kjer slike razdelimo na več pod-slik in vsako od njih obdelamo neodvisno z algoritmom stiskanja. Glede na to, da smo pri sivinskih slikah dobili zelo dobre rezultate, bi lahko bila naslednja izboljšava po vzoru sivinskih slik. Tukaj sliko razdelimo na tri barvne ravnine RGB, vsako od njih obdelamo posebej in na koncu združimo rezultate nazaj v barvno sliko.

Algoritmi po vzorih iz narave se nenehno izpopolnjujejo in izboljšujejo s strani različnih raziskovalcev. Prav gotovo obstajajo še kakšne izboljšave uporabljenih algoritmov v našem delu, ki bi jih lahko prilagodili za naš problem stiskanja slik in tako izpopolnili uspešnost uporabljenih algoritmov. Ena od izboljšav pri obeh algoritmih bi bila uporabiti dodatne mehanizme za izogibanje lokalnemu optimumu.

Velik problem pri stiskanju slik z algoritmi po vzorih iz narave je njihova časovna zahtevnost. Kot smo prikazali v članku, je ugotavljanje dobrih parametrov, testiranje in stiskanje potekalo več dni. Če ne bi bili časovno omejeni, bi bilo zanimivo narediti več zagonov algoritmov nad skupino najboljših parametrov in bi tako lahko dobili drugačne, boljše parametre za stiskanje slik.

Algoritem bi lahko nadgradili tako, da bi lahko slike stiskali v drugem barvnem modelu, kot sta recimo HSV ali CMYK.

7. REFERENCES

[1] C. Blum and D. Merkle. *Swarm Intelligence*. Springer-Verlag, Berlin, 2008.

[2] J. Brest, S. Greiner, B. Bošković, M. Mernik, and V. Žumer. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *T. Evolut. Comput.*, 10(6):646–657, 2006.

[3] C. Darwin. *On the Origin of Species*. Harvard University Press, London, 1964.

[4] A. Eiben and J. Smith. *Introduction to Evolutionary Computing*. Springer-Verlag, Berlin, 2003.

[5] I. Fister, M. Mernik, and J. Brest. *Hybridization of Evolutionary Algorithms, Evolutionary Algorithms*. InTech, 2011.

[6] I. Fister, D. Strnad, X.-S. Yang, and I. Fister Jr. Adaptation and hybridization in nature-inspired algorithms. In *Adaptation and Hybridization in Computational Intelligence*, pages 3–50, Berlin, 2015. Springer-Verlag.

[7] A. Izadi, V. Ciesielski, and M. Berry. Evolutionary non photo-realistic animations with triangular brushstrokes. In *AI 2010: Advances in artificial intelligence*, pages 283–292, Berlin, 2011. Springer-Verlag.

[8] U. Mlakar, I. F. Jr., and I. Fister. Hybrid self-adaptive cuckoo search for global optimization. *Swarm and Evolutionary Computation*, pages –, 2016.

[9] S. Sivanandam and S. Deepa. *Introduction to genetic algorithms*. Springer-Verlag, Heidelberg, 2008.

[10] R. Storn and K. Price. Differential evolution: A simple and efficient heuristic for global optimization over continuous spaces. *J. Global Optim.*, 11(4):341–359, 1997.

[11] X.-S. Yang and S. Deb. Cuckoo search via levy flights. In *World Congress & Biologically Inspired Computing (NaBIC 2009)*, pages 210–214, IEEE Publication, 2009.

[12] A. Zalzala and P. Fleming. *Genetic algorithms in engineering systems*. The institution of electrical engineers, London, UK, 1997.

[13] A. Zamuda and U. Mlakar. Differential evolution control parameters study for self-adaptive triangular brushstrokes. *Informatica*, 39(2):105–113, 2015.