SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 4371

Simulacija I vizualizacija kretanja roja čestica prilikom rješavanja optimizacijskih problema u 2D prostoru

Tanja Šarčević

Zagreb, lipanj 2016.

Ugodna mi je dužnost zahvaliti se mentoru prof. dr. sc. Marinu Golubu na savjetima i pomoći pri izradi ovog rada. Također se zahvaljujem roditeljima Katici i Željku, bratu Filipu i prijateljima i kolegama na stalnoj podršci i razumijevanju tijekom preddiplomskog studija.

Sadržaj

1. Uvod

Optimizacija predstavlja širok problem i za njegovo rješavanje postoji velik broj pristupa. Ipak, najčešće su to teški problemi, kao što su kombinatorni problemi, za čije rješavanje postoji praktično ograničenje – vrijeme. Iz tog razloga rješenja koja se zasnivaju na egzaktnim metodama koje uvijek daju optimalno rješenje često je nemoguće primijeniti. Za probleme čija je rješenja neophodno pronaći u konačnom vremenu razvijene su metode koje ne garantiraju pronalaženje optimalnog rješenja već nastoje zadovoljiti vremensko ograničenje i ostale postavljene uvjete.

Jednu od skupina takvih metoda predstavljaju prirodom inspirirani algoritmi. Promatranje skladnog ponašanja životinjskih skupina, ljudskog tijela te napredak vrsta tijekom dugotrajne evolucije i drugih pojava iz svijeta biologije, navodi na zaključak da se takvo ponašanje može primijeniti na modeliranje konkretnih algoritama za optimizaciju. Velika podskupina prirodom inspiriranih algoritama jest optimizacija rojem čestica inspirirana kolektivnim ponašanjem skupina životinja, gdje je ključan faktor za funkcioniranje skupine interakcija samih jedinki i međusobna razmjena informacija.

U radu je predstavljena osnovna inačica algoritma, njegove modifikacije i parametri, te na koji način oni utječu na brzinu pronalaska rješenja i njegovu kvalitetu. Nadalje, opisano je i nekoliko naprednijih inačica algoritma. Konačno, prikazan je rad algoritma na problemu labirinta i konkretnim funkcijama, te su analizirani njegove rezultati ovisno o parametrima.

2. Algoritam roja čestica

2.1. Opis

Algoritam roja čestica (*eng. Partical Swarm Optimization, PSO*) inspiriran je grupnim i skladnim ponašanjem životinja u svojim okruženjima kao što su jata riba, jata ptica, rojevi pčela itd. Pojedinci u grupnim tvorevinama usklađuju svoje ponašanje prema cijeloj populaciji. Kreću se prostorom prateći najbliže i najbolje jedinke, usklađuju svoju brzinu s brzinom svojih susjeda, izbjegavaju sudare, itd. Uočeno je da je takav sustav pojedinaca s međusobnim interakcijama vrlo pogodan za rješavanje optimizacijskih problema. R.C.Eberhart i J.Kennedy u svojim radovima (*Kennedy and Eberhart*, 1995., *Eberhart and Kennedy*, 1995.) objavljuju svoje ideje o korištenju sustava roja čestica kao optimizatora.

Algoritam roja čestica područje je računalne inteligencije (*eng. Computational Intelligence*) te podskupina algoritama rojeva (*eng. Sworm Intelligence*).

Algoritam je primjer metaheuristike. Metaheuristika se definira kao heuristika opće namjene čiji je zadatak usmjeravanje problemski specifičnih heuristika prema području u prostoru rješenja u kojem se nalaze dobra rješenja. Algoritmom je moguće pretražiti vrlo velik prostor rješenja, no ne može se garantirati da će pronađeno rješenje biti upravo ono optimalno.

2.2. Izvedba

Algoritam roja čestica populacijski je algoritam. Populacija se sastoji od niza čestica koje se kreću kroz višedimenzijski prostor pretraživanja. Cilj čitave populacije je pronaći optimum danog prostora rješenja, odnosno postaviti sve jedinke u pronađeni optimum (minimum ili maksimum) prostora. Jedinke imaju nasumično određene početne pozicije i nasumične male brzine. Nastavak kretanja čestica modelira sociološku interakciju među njima. Izbor novih pozicija čestica ovisi o brzini, njezinom vlastitom najboljem pronađenom rješenju (individualni faktor) te najboljem

pronađenom rješenju populacije (socijalni faktor). Funkcija cilja izračunava se svakom promjenom pozicije. Nakon određenog vremena i određenim brojem promjena pozicija čestice se grupiraju i konvergiraju prema jednom ili više optimuma.

2.3. Pseudokod algoritma roja čestica

Populacija se sastoji od VEL_POP broja jedinki (čestica). Svaka čestica ima poziciju (x), brzinu, tj. maksimalnu promjenu pozicije po iteraciji(v) i vrijednost. Svaka se od navedenih karakteristika čuva u strukturi, a indeksi predstavljaju identifikacijski broj čestice.
Algoritam se izvršava iterativno, a kraj se može definirati na više načina. Najčešće se zadaje određeni broj iteracija ili definira zadovoljavajuće rješenje.

Algoritam 1 Algoritam roja čestica

```
//inicijaliziraj populaciju
za i = 1 do VEL_POP
        x(i) <- random(xmin, xmax)
        v(i) <- random(vmin, vmax)
kraj
ponavljaj
        //evaluiraj populaciju
        za i=1 do VEL_POP
                vrijednost(i) <- funkcija(x(i))
        kraj
        //azuriraj najbolje rjesenje cestice
        za i=1 do VEL_POP
                ako je vrijednost(i) bolja od pbest(i) tada
                        pbest_f(i) <- vrijednost(i)
                        pbest(i) <- x(i)
                kraj
        kraj
        //azuriraj globalno najbolje rjesenje
        za i=1 do VEL_POP
                ako je vrijednost(i) bolja od gbest_f(i) tada
                        gbest_f(i) <- vrijednost(i)
                        gbest(i) <- x(i)
                kraj
        kraj
        //azuriraj brzinu i poziciju cestice
        za i=1 do VEL_POP
                v(i) <- v(i) + c1*rand()*(pbest(i) – pozicija(i))
                                + c2*rand()*(gbest – pozicija(i))
                x(i) <- x(i) + v(i)
        kraj
kraj
```

- Inicijalizacija
  Na početku svakoj se čestitci pridružuje početna pozicija i brzina. Obje vrijednosti nasumično su odabrani brojevi iz definiranog raspona.
- Evaluacija populacije
  Za svaku se česticu izračunava vrijednost funkcije za poziciju na kojoj se nalazi.
- Ažuriranje najboljeg osobnog rješenja
  Izračunata vrijednost funkcije uspoređuje se s trenutnim najboljem rješenjem same čestice i po potrebi ažurira; ako je nova vrijednost bolja dosadašnje najbolje osobne vrijednosti, najbolja osobna vrijednost postaje nova vrijednost.
  Osim vrijednosti, pamti se i pozicija za koju čestica ima najbolju vrijednost.
- Ažuriranje najboljeg globalnog rješenja
  Ažuriranje najboljeg rješenja cijele populacije odvija se usporedbom trenutno dobivene vrijednosti čestice i dosadašnje najbolje globalne vrijednosti.
- Ažuriranje brzine
  Svakoj čestici u populaciji izračunava se nova brzina tako da se trenutnoj brzini pridruži individualna komponenta modulirana faktorom individualnosti (*c1*) i nasumičnim brojem iz raspona [0, 1] te socijalna komponenta modulirana faktorom socijalnosti (*c2*) i nasumičnim brojem iz raspona [0, 1]. Ažuriranje se izvodi prema sljedećem izrazu:

  $$v_i = v_i + c1 * rand() * (pbest - x) + c2 * rand() * (gbest - x) \quad (2.1)$$

  Faktore *c1* i *c2* moguće je mijenjati a to će rezultirati različitim ponašanjem populacije. Postavljanjem faktora c1 na veću vrijednost povećava se individualnost jedinke a time i potiče pretraživanje prostora. Postavljanjem faktora c2 na veću vrijednost čestice će težiti istraživanju prostora oko pronađenog najboljeg globalnog rješenja.
- Ažuriranje pozicije
  Svakoj se čestici na temelju brzine ažurira i pozicija tako da se trenutna brzina zbroji s izračunatom brzinom.

  $$x_i = x_i * v_i \quad (2.2)$$

2.4. Heuristika i modifikacije algoritma

2.4.1. Parametri

Parametri bitno utječu na ponašanje algoritma i performanse optimizacije stoga ih je bitno dobro podesiti. Izbor parametara tema je mnogih istraživanja, a u nastavku je ukratko opisan njihov utjecaj.

Veličina populacije ovisi o vrsti problema, no obično je veličinu idealno postaviti na vrijednost od oko 20 do 50 čestica.

Brzina kojom se čestice mogu kretati treba biti ograničena na manji postotak veličine domene. U suprotnom algoritam može doći u divergentno stanje. Prevelike brzine rezultirat će prevelikim promjenama pozicija u malom broju iteracija, te je moguće da čestice prelete i izbjegnu područja dobrih rješenja. Postavljanjem malih brzina moguće su situacije gdje čestice ostanu zatočene u lokalnom optimumu.

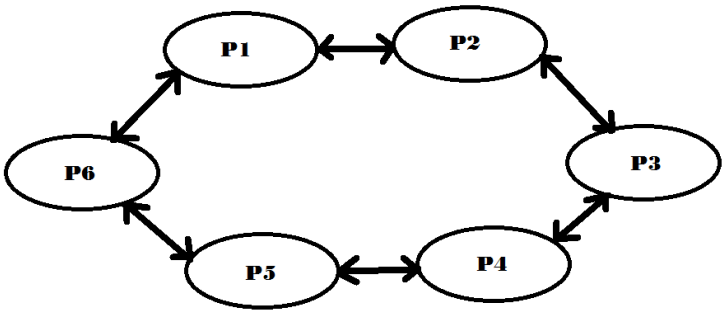
Faktori učenja(individualni i socijalni) određuju utjecaj najboljih rješenja na česticu. Ranije je spomenuto kako veći individualni faktor omogućuje bolje istraživanje prostora, a veći socijalni faktor veću konvergenciju prema globalnom rješenju. Faktori trebaju poprimiti vrijednost između 0 i 4 (M. Čupić, 2013.).

2.4.2. Lokalno susjedstvo i topologije

Klasični algoritam lako zapne u lokalnim optimumima. Razlog tomu je utjecaj globalnog parametra na čitavu populaciju. Čim jedna jedinka pronađe potencijalno dobro rješenje, sve su jedinke automatski privučene k tom rješenju. Ovakva vrsta algoritma roja čestica naziva se *potpuno-informirani algoritam roja čestica* (eng. *Fully informed PSO*). Preranu konvergenciju potpuno-informiranog algoritma moguće je spriječiti manjom modifikacijom. U ovoj inačici algoritma česticama potrebno je uskratiti informaciju o globalnom najboljem rješenju. Umjesto te informacije formiraju se topološki uređaji koji predstavljaju susjedstvo svake pojedine čestice. Susjedstvo čestice manji je podskup populacije u kojem dvije čestice jesu ili nisu u susjedskom odnosu te jedna čestica može imati proizvoljno mnogo susjeda. Broj susjeda obično se definira na jednaku vrijednost za sve čestice. Čestica tada u svakom trenutku umjesto informacije o najboljem pronađenom rješenju čitave populacije ima informaciju o najboljem rješenju koje su pronašle čestice iz njezinog susjedstva.

Susjedstvo se može ostvariti na više načina. Najintuitivnije je ono geometrijsko, tj. formirano prema Euklidskoj udaljenosti između pozicija čestica. Najbližih n čestica čine susjedstvo. Češće se pak koristi socijalno definiranje susjedstva gdje veze među česticama nemaju nikakve veze s pozicijama. U populaciji se točno odredi koja čestica

komunicira s kojom te su tijekom algoritma česticama dostupne informacije samo od čestica iz tako definiranog susjedstva. Slika 2.1. prikazuje primjer ovakvog susjedstva. Veze među česticama označene su strjelicama



Slika 2.1. Lokalno susjedstvo

Uvođenjem susjedstva uvodi se i pojam lokalnog optimuma – najbolje rješenje pronađeno u susjedstvu. Izraz za ažuriranje stoga se mijenja i opisuje ga sljedeći izraz:

$$v_i = v_i + c1 * rand() * (pbest_i - x) + c2 * rand() * (lbest_i - x) \quad (2.3.)$$

$lbest$ u ovom je slučaju najbolje rješenje susjedstva određene čestice.

3. Algoritam pametne kapi vode

3.1. Osnovna načela algoritma pametne kapi vode

Nalik ponašanju pčela u rojevima i mrava u kolonijama, i u prirodnom toku rijeke može se primijetiti slično kolektivno ponašanje. Prirodan tok rijeke mijenja se i najčešće pronalazi najbolji put od svih mogućih od svog početka do kraja. Tako dobiveni putovi rezultat su interakcije između kapi vode i korita rijeke.

Algoritam pametne kapi vode (*eng. Intelligent Water Drop Algorithm*) optimizacijski je algoritam inspiriran ponašanjem kapljica vode koje tvore optimalnu putanju rijeke. Algoritam spada u algoritme rojeva i metaheuristike. Kapljice vode koje predstavljaju jedinke populacije ovog algoritma komuniciraju međusobno u pronalasku optimalnog rješenja.

Pretpostavlja se da jedna kap vode može prenositi dio tla. Stoga je jedna od karakteristika kapi vode da određenu količinu tla prenosi s jednog mjesta na drugo mjesto nizvodno. Druga karakteristika kapi vode njezina je brzina. Kapi vode kreću se od početne do krajnje točke mijenjajući pritom brzinu i količinu tla koje prenose. Gibanjem kapi između dvije točke brzina kapljice se povećava. Povećava se i količina tla koje prenosi, odnosno količina tla u riječnom koritu se između te dvije točke smanjuje.

Temeljem opisanog ponašanja kapljica vode modelira se algoritam pametne kapi vode u kojem svaka pametna kap vode ima sljedeće značajke:

- Brzinu
- Količinu tla

Tijekom izvođenja algoritma karakteristike se mijenjaju. Čestica se kreće od početne do krajnje točke s diskretnim koracima. Brzina se inicijalno postavlja na nasumičnu vrijednost svim česticama populacije, a početna vrijednost količine tla postavlja se na nulu. U svakom sljedećem koraku brzina se može povećati, kao i količina tla koju čestica prenosi. Količina tla koju čestica prenosi obrnuto je proporcionalna količini tla u koritu.

Između dvije pozicije brzina se povećava nelinearno, obrnuto proporcionalno količini tlu u koritu. Drugim riječima, na pozicijama s manje zemlje u koritu čestice se kreću brže. Također, što se više čestica kreće određenim putem, manje će tla biti na tom putu.

Problem algoritmu pametne kapi vode zadaje se pomoću grafa. Graf se zadaje kao set čvorova N (*nodes*) i set bridova E (*edges*). Algoritam se izvodi iterativno. U svakom koraku iteracije čestice se kreću od čvora do čvora grafa prateći bridove. Jedna iteracija je gotova kada svaka od čestica ažurira svoje rješenje. Populacija pamti dvije vrste rješenja: najbolje rješenje iteracije T_{IB} (*iteration best*) i ukupno najbolje rješenje T_{TB} (*total best*). Nakon svake iteracije ove se dvije vrijednosti ažuriraju. Iz rješenja svih čestica u završenoj iteraciji izračunava se najbolje rješenje iteracije, te se ono koristi za ažuriranje ukupnog najboljeg rješenja. Kraj algoritma može se definirati konačnim brojem iteracija ili zadovoljavajućom vrijednošću ukupnog najboljeg rješenja.

3.2. Opis algoritma pametne kapi vode

Algoritam ima dvije vrste parametara:

- Statički parametri – ostaju konstantni tijekom izvođenja algoritma
- Dinamički parametri – mijenjaju se tijekom izvođenja i ponovno se inicijaliziraju nakon svake iteracije

- Inicijalizacija statičkih parametara
  - Algoritmu se zadaje problem u obliku grafa (N, E) gdje je N set čvorova, a E set bridova.
  - Vrijednost najboljeg ukupnog rješenja postavlja se na najgoru moguću vrijednost.
  - Maksimalan broj iteracija zadaje korisnik, a brojač iteracija postavlja se na nulu.
  - Veličina populacije inicijalizira se najčešće na broj čvorova zadanog grafa.
  - Inicijaliziraju se parametri brzine (av, bv, cv) i parametri tla (as, bs, cs) na vrijednosti: $av = 1, bv = 0.1, cv = 1, as = 1, bs = 0.1, cs = 1$. (Shah-Hosseini, 2009.)
  - Inicijalizira se parametar lokalnog ažuriranja tla i parametar globalnog ažuriranja tla na malu pozitivnu vrijednost [0, 1].
  - Početna vrijednost količine tla na bridovima grafa i početna brzina čestica određena je od strane korisnika, i određuje se eksperimentalno ovisno o zadanom problemu.
- Inicijalizacija dinamičkih parametara
  - Svakoj je čestici pridružena lista posjećenih čvorova koja je na početku prazna.
  - Brzina svake čestice postavlja se na vrijednost koju je odabrao korisnik, a vrijednost količine tla za svaku česticu postavlja se na nula.

- Pridruživanje čestica čvorovima
  Čestice se nasumično raspoređuju na čvorove na način da se jednoj čestici pridruži jedan čvor. Ažurira se lista posjećenih čvorova svake čestice.
- Sljedeći koraci ponavljaju se za sve čestice kojima se dodijeli djelomično rješenje:
  - Svaka čestica odabire novi čvor koji nije u njenoj listi posjećenih čvorova koristeći sljedeći izraz za vjerojatnost:

$$p_i(j) = \frac{f(tlo(i,j))}{\sum_k f(tlo(i,k))} \tag{3.1.}$$

    tako da je

$$f\big(tlo(i,j)\big) = \frac{1}{\varepsilon_s + g(tlo(i,j))} \tag{3.2.}$$

$$g\big(tlo(i,j)\big) = \begin{cases} tlo(i,j), & ako\ je \min\big(tlo(i,l)\big) \geq 0 \\ tlo(i,j) - \min\big(tlo(i,l)\big), & inače \end{cases} \tag{3.3.}$$

    Novi čvor se dodaje u listu posjećenih čvorova.
  - Svaka čestica ažurira vlastitu brzinu prema izrazu:

$$brzina(t+1) = brzina(t) + \frac{a_v}{b_v + c_v * tlo^2(i,j)} \tag{3.4.}$$

  - Izračunava se vrijednost tla koje čestica prenosi od svog starog prema novom čvoru prema izrazu:

$$\Delta tlo(i,j) = \frac{a_s}{b_s + c_s * vrijeme^2(i,j;brzina(t+1))} \tag{3.5.}$$

    tako da je

$$vrijeme\big(i,j;brzina(t+1)\big) = \frac{HUD(j)}{brzina(t+1)} \tag{3.6.}$$

    gdje je $HUD(j)$ heuristička funkcija definirana za zadani problem.

  - Ažurira se vrijednost tla na bridu između starog i novog čvora $tlo(i,j)$ te vrijednost tla koje čestica prenosi tlo_i:

$$tlo(i,j) = (1-\rho_n) * tlo(i,j) - \rho_n * \Delta\text{tlo(i,j)} \tag{3.7.}$$
$$tlo_i = tlo_i + \Delta tlo(i,j)$$

- Ažuriranje najboljeg rješenja iteracije T^{IB}
- Ažuriranje vrijednosti tla na bridovima najboljeg rješenja trenutne iteracije prema izrazu:

$$tlo(i,j) = (1+\rho) * tlo(i,j) \tag{3.8.}$$

- Ažuriranje ukupnog najboljeg rješenja
  Ukupno najbolje rješenje ažurira se na vrijednost najboljeg rješenja trenutne iteracije ako je ono veće od ukupno najboljeg.

4. Algoritam kretanja svjetlećih crva

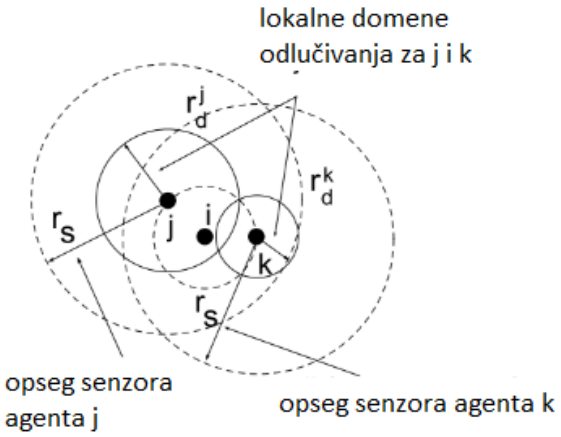
4.1. Opis algoritma kretanja svjetlećih crva

Algoritam kretanja svjetlećih crva (*eng. Glowworm Swarm Algorithm*) inspiriran je kolektivnim ponašanjem svjetlećih crva, odnosno krijesnica. Algoritam dijeli neke svojstva s nešto poznatijim algoritmima baziranim na kolektivnoj inteligenciji, kao što je mravlji algoritam i algoritam roja čestica, no s nekoliko značajnih razlika.

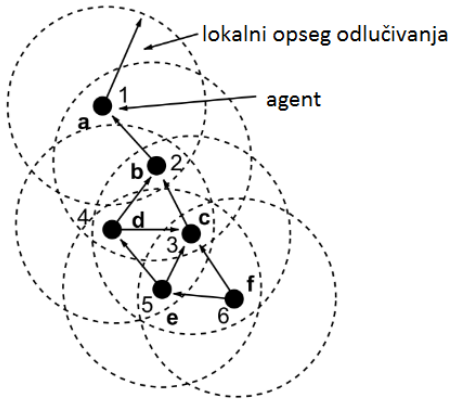
Agenti algoritma kretanja svjetlećih crva zamišljeni su kao krijesnice koje se kreću i komuniciraju u grupi. Nekoliko je glavnih karakteristika prema kojima su modelirani agenti algoritma i njihovo kretanje:

- Svjetlost koju emitiraju
- Senzor za svjetlost

Agenti na sebi prenose određenu količinu osvjetljenja koja se naziva *luciferin*. Agenti pomoću funkcije dobrote računaju dobrotu svoje trenutne pozicije, te ju šifriraju u vrijednost luciferina koje emitiraju svojim susjedima. Agent identificira svoje ogovarajuće susjedstvo i računa svoje iduće pokrete prema njemu, koje je ograničeno dometom agentovog senzora. Svaki pojedini agent u populaciji izabire susjeda s većom vrijednošću luciferina od svoje i usmjerava se prema njemu. Na primjer, slika 4.1. prikazuje situaciju gdje je agent i u dometu senzora agenata j i k i jednako je udaljen od njih. Međutim, j i k imaju različite veličine susjedstva i samo j prima informaciju od i. Slika 4.2. prikazuje usmjereni graf odabira agenata svog kretanja prema susjeda s većim osvjetljenjem od svojeg.



Slika 4.1. Lokalno susjedstvo algoritma kretanja svjetlećih crva



Slika 4.2. Usmjereni graf kretanja agenata

Ovakvo kretanje omogućava česticama da se podijele u podgrupe koje konvergiraju prema više lokalnih optimuma zbog toga što su kretnje bazirane isključivo na lokalnim informacijama i odabranim susjedskim interakcijama.

4.2. Pseudokod algoritma svjetlećih crva

Algoritam započinje postavljanjem agenata u prostor rješenja na nasumične vrijednosti pozicija. Svi agenti na početku sadrže jednaku količinu luciferina. Algoritam se sastoji od dvije faze:

1. Faza ažuriranja luciferina

2. Faza kretanja baziranog na tranzicijskom pravilu

Algoritam 2 Algoritam kretanja svjetlećih crva

```
//inicijaliziraj populaciju
za i=1 do VEL_POP
        x(i) <- random()
kraj
t <- 1
dok je t ≤ iter_max
        //azuriraj luciferin
        za i=1 do VEL_POP
                l(i,t) <- (1-ρ)*l(i, t-1) + ϒ*J(x(i,t))
        kraj
        //faza kretanja
        za i=1 do VEL_POP
                N(i, t) = {j : d(t) < r(i, d, t); l(i,t) < l(j,t)}
                za svaki j ∈ Ni(t)
                        p(i, j, t) <- vjerojatnost_kretanja_prema_agentu(j)
                kraj
                j <- odaberi_agenta(p)
                xi(t + 1) <- azuriraj_kretanje(i)
                r(i, d, t + 1) <- azuriraj_opseg_susjedstva(i, d)
        kraj
        t <- t+1
kraj
```

Faza ažuriranja luciferina:

Ažuriranje ovisi o vrijednosti funkcije na poziciji agenta. Tijekom ove faze svaki agent populacije dodaje količinu luciferina, proporcionalnu dobroti trenutne pozicije u prostoru rješenja, prethodnoj količini luciferina. Također, količina luciferina reducirana je kako bi se simulirao raspad luciferina s vremenom. Luciferin se ažurira prema izrazu:

$$l_i(t+1) = (1-\rho)l_i(t) + \gamma J\big(x_i(t+1)\big) \tag{4.1.}$$

gdje $l_i(t)$ predstavlja razinu luciferina vezanu za agenta i u trenutku t, ρ je konstanta raspadanja luciferina ($0 < \rho > 1$), γ je konstanta poboljšanja luciferina, a $J(x_i(t))$ predstavlja vrijednost funkcije na poziciji agenta i u trenutku t.

Faza kretanja:

Tijekom faze kretanja, svaki se agent koristeći vjerojatnost odlučuje kretati prema susjedu čija je vrijednost luciferina veća nego njegova vlastita. To znači da su jedinke privučene onim jedinkama čija je svjetlost jača. Za svakog agenta i, vjerojatnost kretanja prema susjedu j dana je izrazom (*vjerojatnost_kretanja_prema_agentu(agentID)*):

$$p_{i,j}(t) = \frac{l_j(t) - l_i(t)}{\sum_{k \in N_i(t)} l_k(t) - l_i(t)} \tag{4.2.}$$

gdje je $j \in N_i(t), N_i(t) = \{j: d_{i,j}(t) < r_d^i(t); l_i(t) < l_j(t)\}$ set susjeda agenta i u trenutku t, $d_{i,j}(t)$ predstavlja euklidsku udaljenost između agenata i i j u trenutku t, a $r_d^i(t)$ je promjenjivi opseg susjedstva povezanog s agentom i u trenutku t.
Pretpostavimo da je agent i odabrao agenta $j \in N_i(t)$ s vjerojatnošću $p_{i,j}(t)$. Tada se vremensko diskretni model kretanja agenata zadaje s (*azuriraj_kretanje(agentID)*):

$$x_i(t+1) = x_i(t) + s\left(\frac{x_j(t) - x_i(t)}{\|x_j(t) - x_i(t)\|}\right), \tag{4.3.}$$

gdje je $x_i(t) \in \mathbb{R}^m$ pozicija agenta i u trenutku t u m-dimenzionalnom realnom prostoru $\mathbb{R}^m$, $\|\cdot\|$ predstavlja operator Euklidske norme, a s (< 0) je veličina koraka.

Pravilo ažuriranja opsega susjedstva:

Svakom agentu i pridjeljujemo susjedstvo čije je opseg r_d^i dinamičke prirode. Činjenica da se za veličinu susjedstva ne koristi fiksna vrijednost zahtjeva nekakvo opravdanje. U slučaju kada agenti ovise samo o lokalnim informacijama kako bi odlučili o idućim pokretima, očekuje se da bi broj dosegnutih maksimuma bio funkcija radijalnog opsega senzora. Ustvari, ako bi opseg senzora svakog agenta pokrio cijeli prostor pretraživanja, svi bi se agenti kretali prema globalnom optimumu dok bi lokalni bili ignorirani. Kako se pretpostavlja da nikakve informacije o funkciji nisu dostupne (npr. broj lokalnih optimuma), teško je podesiti opseg susjedstva na vrijednost koja odgovara različitim funkcijama. Primjerice, unaprijed određeni opseg susjedstva r_d funkcionirala bi relativno bolje na funkcijama gdje je razmak minimuma veći od r_d nego na onima gdje je razmak manji od r_d. Iz tog razloga algoritam kretanja svjetlećih crva koristi prilagodljiv opseg susjedstva kako bi prepoznao postojanje višestrukih optimuma u multimodalnim funkcijama.
Neka je r_0 početni opseg susjedstva svake čestice (tako da je $r_d^i(0) = r_0 \; \forall i$). Za ažuriranje opsega susjedstva svakog agenta koristi se sljedeće pravilo (*azuriraj_opseg_susjedstva*):

$$r_d^i(t+1) = \min\left\{r_s, max\{0, r_d^i(t) + \beta(n_t - |N_i(t)|)\}\right\}, \tag{4.4.}$$

gdje je β konstantni parametar, a n_i parametar koji kontrolira broj susjeda.

5. Testne funkcije za optimizaciju

Budući da je cilj vizualno prikazati rad algoritma pri rješavanju optimizacijskog problema, ispitne funkcije trebaju bit odabrane tako da je moguće na njima valjano prikazati rad algoritma. Funkcije trebaju ispitati različite situacije s kojima se optimizacijski algoritam može suočiti pri rješavanju problema. Također, potrebno je evaluirati obilježja optimizacijskog algoritma:

- preciznost,
- brzinu,
- učinak.

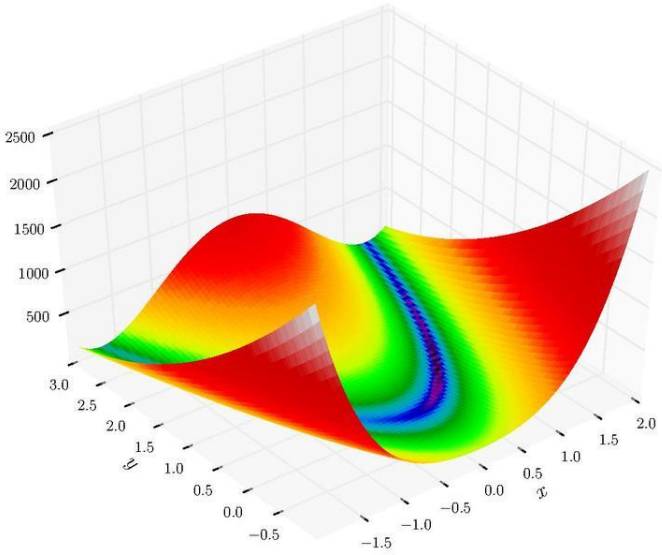
U primijenjenoj matematici testne funkcije poznate su kao umjetni reljefi. U nastavku su navedene i opisane neke od njih.

Rosenbrockova funkcija

$$f(x) = \sum_{i=1}^{N-1} 100(x_{i+1} - {x_i}^2)^2 + (1 - x_i)^2 \quad gdje\ je\ x = [x_1, \dots, x_N] \in \mathbb{R}^N \qquad (5.1.)$$

Minimum: $Min = \begin{cases} n = 2 \ \rightarrow f(1,1) = 0 \\ n = 3 \ \rightarrow f(1,1,1) = 0 \\ n > 3 \ \rightarrow f\left(\underbrace{1, \dots, 1}_{n\ puta}\right) = 0 \end{cases}$

Domena pretraživanja: $-\infty \leq x_i \leq \infty, \quad 1 \leq i \leq n$



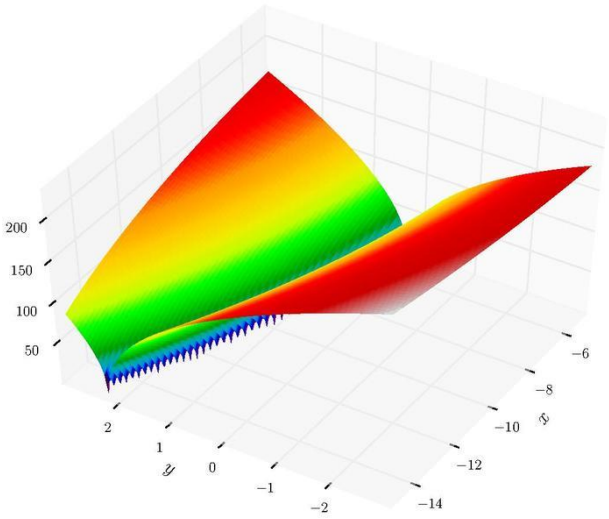
Slika 5.1. Rosenbrockova funkcija

Bukinova funkcija

$$f(x,y) = 100\sqrt{|y - 0.01x^2|} + 0.01|x + 10| \qquad (5.2.)$$

Minimum: $f(-10,1) = 0$

Domena pretraživanja: $-15 \leq x \leq -5, \quad -3 \leq y \leq 3$



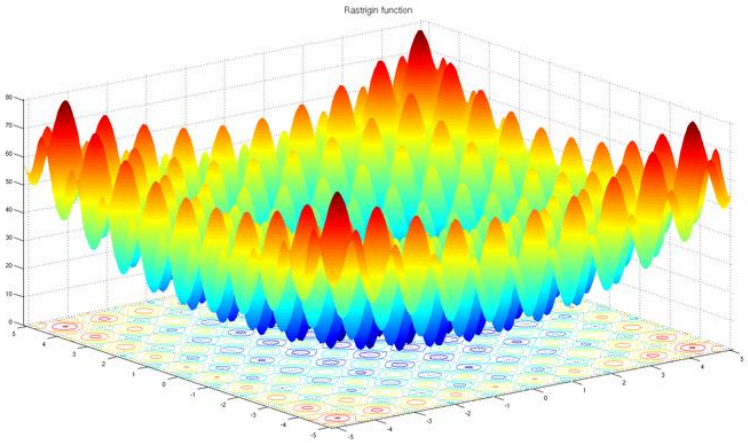
Slika 5.2. Bukinova funkcija

Rastriginova funkcija

$$f(x) = 10n + \sum_{i=1}^{n} [x_i{}^2 - 10\cos(2\pi x_i)] \qquad (5.3.)$$

Globalni minimum: $f(0) = 0$

Domena pretraživanja: $x_i \in [-5.12, 5.12]$



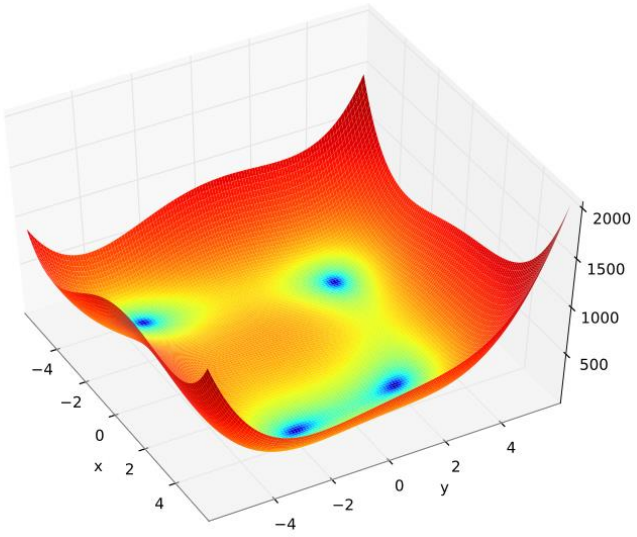
Slika 5.3. Rastriginova funkcija

Himmelblauova funkcija:

$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2 \qquad (5.4.)$$

Lokalni maksimum: $f(-0.270845, -0.923039) = 181.617$
Lokalni minimumi:

$f(3.0,2.0) = 0.0$
$f(-2.805118,3.131312) = 0.0$
$f(-3.77931, -3.283186) = 0.0$
$f(3.584428, -1.848126) = 0.0$



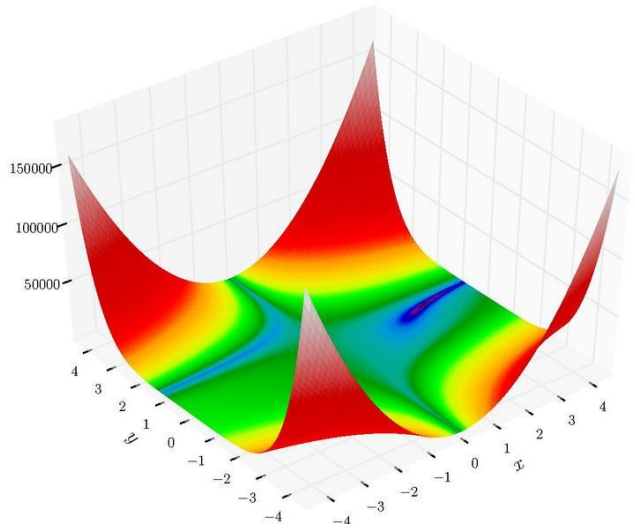
Slika 5.4. Himmelblauova funkcija

Bealova funkcija:

$$f(x,y) = (1.5 - x - xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2 \qquad (5.5.)$$

Lokalni minimum: $f(3,0.5) = 0$

Domena pretraživanja: $-4.5 \leq x, y \leq 4.5$



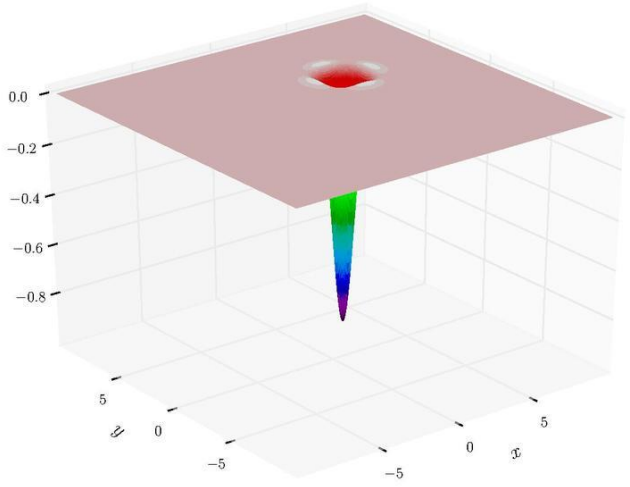
Slika 5.5. Bealova funkcija

Easomova funkcija:

$$f(x,y) = -\cos(x)\cos(y)\exp(-((x-\pi)^2 + (y-\pi)^2)) \qquad (5.6.)$$

Minimum: $f(\pi, \pi) = -1$

Domena pretraživanja: $-100 \leq x, y \leq 100$



Slika 5.6. Easomova funkcija

6. Ostvareno programsko rješenje vizualizacije algoritma roja čestica

Kako bi se vizualno prikazao rad algoritma roja čestica, razvijeno je programsko rješenje koje simulira kretanje čestica po dvodimenzionalnom prostoru optimizirajući pritom zadanu funkciju s dvije varijable.

6.1. Odabir problematike i domene

Algoritam roja čestica pogodan je za optimiziranje velikog broja različitih problema. Ipak, algoritam bi se trebao što intuitivnije vizualno prikazati kako bi se jasno prezentirali osnovni entiteti koji sudjeluju u ostvarivanju algoritma i rješavanju danog problema.

Odabrani problem predstavlja funkcija s dvije varijable. Ovakav izbor omogućuje vrlo zoran vizualni prikaz samog problema, kao i tijeka izvođenja algoritma. Funkcija je prikazana u obliku dvodimenzionalnog prostora u kojem je nijansama boje jasno prikazan spektar vrijednosti prostora rješenja.

Agenti svojim kretanjem kroz prostor prikazuju korake u rješavanju problema. Već iz samog kretanja moguće je uočiti karakterističnо ponašanje agenata za različiti odabir parametara.

6.2. Problem labirinta

Proširena problematika promatra se u obliku labirinta koji čini niz prepreka te jedno ili više ciljnih stanja. Agenti nastoje pronaći izlaz iz labirinta krećući se po posve neoznačenom prostoru, osim same ciljne točke i zidova koji predstavljaju nedopuštene pozicije.

6.3. Programsko ostvarenje algoritma

Programsko rješenje ostvaruje algoritam čiji je prostor rješenja zadan u dvije dimenzije. Inverzijom ovisnosti postignuto je dovoljno je generički izrađeno rješenje, takvo da je lako u budućnosti ugraditi višedimenzionalni prostor rješenja. Osnovni algoritam roja čestica proširen je nekolicinom modifikacija:

1. Lokalno susjedstvo
2. Faktor inercije
3. Kažnjavanje nedopuštenih kretnji

Većina parametara (npr. broj iteracija, veličina populacije, individualni i socijalni faktor…) korisniku su ponuđeni na izbor te se na početku izvođenja algoritma postavljaju na vrijednosti koje je korisnik odabrao preko korisničkog sučelja. Maksimalni pomak čestice po dimenzijama zadaje se postotkom veličine domene. Početna pozicija i brzina čestica postavljaju se na nasumične vrijednosti, a najbolje rješenje čestice na vrijednost početne pozicije.
Osim parametara, korisnik bira i tip susjedstva u algoritmu. Dvije su verzije algoritma ovisno o izboru susjedstva:

1. Verzija koja koristi globalno najbolje rješenje za izračunavanje idućih rješenja
2. Verzija koja koristi najbolje rješenje lokalnog susjedstva za izračunavanje idući rješenja

6.3.1. Globalno susjedstvo

Programsko ostvarenje globalnog susjedstva potpuno je u skladu s opisanim osnovnim algoritmom roja čestica. Najbolje rješenje pronalazi se iterativno na razini cijele populacije i sukladno tome ažurira.

6.3.2. Lokalno susjedstvo

Osim opcije za izbor lokalnog susjedstva, korisnik bira i veličinu susjedstva u obliku postotka populacije. Rješenje programski ostvaruje socijalno definiranje lokalnog susjedstva. Svaka od čestica označena je indeksom u polju. Susjedstvo svake čestice čini $nSize$ čestica koje se nalaze u polju oko nje. Ilustrirajmo to na primjeru. Promatra se čestica $particles[i]$ i situacija da je veličina susjedstva $nSize$ jednaka 4. Tada lokalno susjedstvo zajedno s promatranom česticom čine još i čestice $particles[i-1], particles[i-1], particles[i+1], particles[i+2]$.
Najbolja vrijednost uobičajeno se ažurira na najbolju vrijednost čestice u populaciji.

6.3.3. Faktor inercije

Proces pretraživanja prostora rješenja uobičajeno je podijeliti u dva koraka. U prvom koraku obavlja se grubo pretraživanje prostora kako bi se locirala „zanimljiva“ područja, a potom se u drugom koraku obavlja fino pretraživanje unutar lociranih kandidata. Ovakvo ponašanje ostvaruje se modificiranjem izraza za ažuriranje brzine faktorom inercije. Izraz za ažuriranje brzine u ovom slučaju ima oblik:

$$v_{i,d} = w(t) * v_{i,d} + c1 * rand() * \left(pbest_{i,d} - x\right) + c2 * rand() * (gbest_d - x)$$

Inercijska komponenta brzine množi se vremenski promjenjivim faktorom w. Inicijalno, w se postavlja na vrijednost 0.9 (M.Č), a povećanjem broja iteracija t vrijednost se smanjuje prema nekoj minimalnoj vrijednosti. Ažuriranje w(t) obavlja se prema izrazu:

$$w(t) = \begin{cases} \frac{t}{T} * (w_{min} - w_{max}) + w_{max}, & t \leq T \\ w_{min}, & t > T \end{cases}$$

gdje su w_{max} i w_{min} najveća i najmanja željena vrijednost, a T iteracija u kojoj težinski faktor treba pasti na w_{min}. Vrijednost T postavljena je na polovicu ukupnog broja iteracija. Ovakvim odabirom postiže se da prvu polovicu iteracija čestice nasumičnije istražuju prostor rješenja, a drugu polovicu konvergiraju prema pronađenim dobrim rješenjima.

```
//pocetne vrijednosti
this.linWeightStart = 0.9;
this.linWeightEnd = 0.4;
this.linWeightTreshold = numberOfIterations/2;

…

//postavi faktor inercije
double w;
if(iteracija>linWeightTreshold) {
        w = linWeightEnd;
} else {
        w = linWeightStart + (linWeightEnd -
        linWeightStart)*1.0*iteracija/linWeightTreshold;
}

…

//azuriraj brzinu svake cestice koristeci faktor inercije
for(int i=0; i<particles.length; i++) {
        for (int d=0; d<dims; d++) {
                particles[i].velocity[d] =
        (int)Math.round(w*particles[i].velocity[d]
        +c1*rand.nextDouble()*(particles[i].bestVars[d] -
        particles[i].vars[d])
                        + c2*rand.nextDouble()*(socialBest[d]-
                particles[i].vars[d]));
```

Slika 6.1. Izvorni kod primjene faktora inercije

6.3.4. Sustav kažnjavanja za nedopuštene akcije

Programski ostvarena vizualizacija zahtjeva određena ograničenja u gibanju čestica. Npr. nedopušteno je da se jedinke kreću izvan okvira domene. Za ovakav je prijestup potrebno je ostvariti neku vrstu kažnjavanja, odnosno promjene u poziciji i brzini čestice. U konkretnom slučaju, ako se dogodi situacija da čestica izračuna kao sljedeću poziciju vrijednost koja se nalazi izvan domene, čestica biva reflektirana od ruba nazad u domenu.

6.4. Opis programskog sustava

6.4.1. Odabir programskog jezika i razvojnog okruženja

S obzirom na težnju da se pri programskom ostvarenju dosegne određena razina apstrakcije, odabir jezika svakako je iz skupine viših programskih jezika i to onih objektno-orijentiranih. Programski jezik Java objektno je orijentirani programski jezik, nezavisan od platforme te ima pojednostavljenu kontrolu memorije (automatsko čišćenje resursa koji se više ne koriste, tzv. *garbage collector*), stoga je odabran za programsko ostvarenje ovog sustava.

Za razvoj sustava korištena je programska razvojna okolina (IDE – Integrated Development Environment) Eclipse koja je besplatna za korištenje i otvorenog koda.

Za ostvarenje grafičkog korisničkog sučelja korištena je tehnologija JavaFX. JavaFX sadrži brojne grafičke pakete kojima je moguće ostvariti bogata korisnička sučelja za prikaz raznih medija kao što su video, animacije i audio sadržaj, ugraditi web stranice unutar JavaFX aplikacije, urediti komponente tehnologijom kao što je CSS, dizajnirati grafičko korisničko sučelje FXML-om i još mnogo toga. Također pruža mogućnost kombiniranja JavaFX i Swing komponenata.
U konkretnom programskom ostvarenju korisnima su se pokazale sljedeće tehnologije:

- Canvas API – omogućuje crtanje direktno na prostor JavaFX scene koja se sastoji od samo jednog grafičkog elementa (node). Canvas API korišten je za iscrtavanje vizualne reprezentacije funkcije, agenata i njihovog kretanja po prostoru, iscrtavanje mape istraženog prostora te za crtanje labirinta.
- Ugrađene kontrole korisničkog sučelja i CSS – pruža sve glavne komponente korisničkog sučelja i omogućuje uređivanje komponenata pomoću CSS-a. Neke od komponenata bilo je potrebno istaknuti, stoga se CSS pokazao kao jednostavno i korisno rješenje.

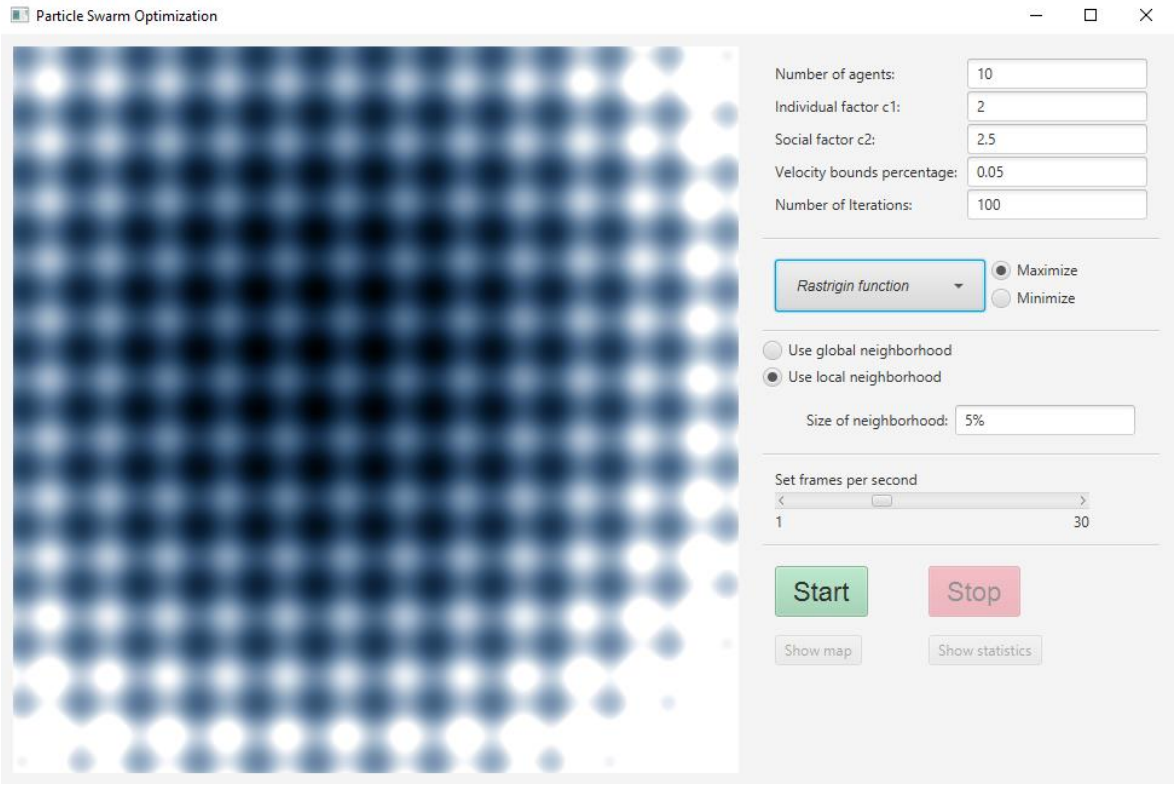
6.4.2. Opis osnovnih razreda

Od ostvarenih klasa treba istaknuti nekoliko bitnijih:

- *Particle* – ostvaruje pojedinu česticu populacije
- *ParticleSwarmOptimization* – ostvaruje algoritam roja čestica te pamti populaciju jedinke, susjedstvo i problem koji se optimizira.
- *Neighborhood* – sučelje koje nasljeđuju razredi za ostvarivanje odabranog susjedstva. Ovo sučelje omogućuje inverziju ovisnosti između algoritma roja čestica i ostvarenja susjedstva za konkretni problem koji algoritam rješava.
- *LocalNeighborhood* – nasljeđuje sučelje *Neighborhood*, te ostvaruje lokalno susjedstvo
- *GlobalNeighborhood* - nasljeđuje sučelje *Neighborhood*, te ostvaruje globalno susjedstvo
- *DomainFunction* – ostvaruje primjerak funkcije koja je odabrana za optimiziranje preko korisničkog sučelja. Čuva referencu na objekt tipa *IFunction*, a metodom *initFunction* stvara se odabrana funkcija.
- *SimulationFrame* – ostvaruje komponentu koja grafički prikazuje domenu i kretanje agenata po njoj
- *MainFrame* – ostvaruje korisničko sučelje kroz koje se pokreće algoritam, podešavaju parametri te upravlja izvođenjem.

6.4.3. Korisničko sučelje

Programsko rješenje uključuje grafičko korisničko sučelje kojim korisnik može upravljati izvođenjem algoritma. Korisničko sučelje prikazano je na slici 6.2.



Slika 6.2. Korisničko sučelje

Korisničko sučelje sastoji se od dva osnovna i nekoliko dodatnih dijelova. Glavni prozor korisničkog sučelja sastavljen je od:

- Kontrolnog dijela za unos i mijenjanje parametara algoritma te opcija za podešavanje i upravljanje simulacijom
- Prikaza domene i kretanja agenata tijekom izvođenja algoritma

Kontrolni dio korisničkog sučelja nalazi se na desnoj strani prozora i sadrži niz parametara koje je za potrebe testiranja algoritma moguće mijenjati. Za svaki su parametar zadane početne vrijednosti, a korisnik ih po želji mijenja. U slučaju da je za neki od prostora za unos parametara unesena nedozvoljena vrijednost ili prostor ostavljen prozor, ispisuje se odgovarajuća poruka i upisivanje se treba ponoviti. Parametri i opcije odvojene su po kontekstu.

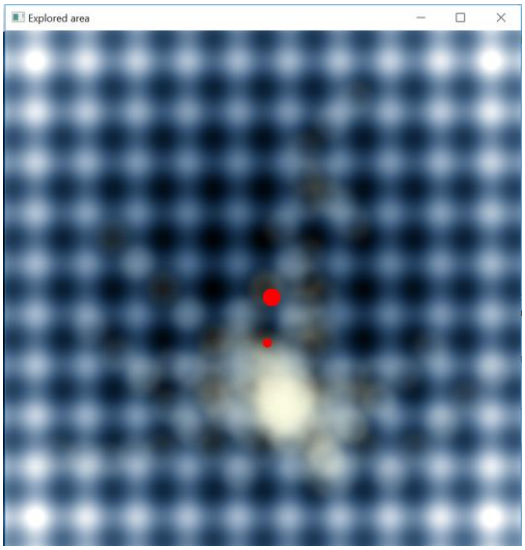
1. Parametri algoritma:
    a. Broj agenata (veličina populacije)
    Za rješavanje problema funkcije broj agenata treba biti veći od 1 kako bi se postiglo kolektivno ponašanje. Gornja granica veličine populacije je za konkretan problem 100 jer zbog odabira problema sve populacije s više agenata pokazuju približno iste rezultate.
    b. Individualni faktor $c1$
    c. Socijalni faktor c2

    d. Maksimalna brzina agenta
       Postotkom veličine domene određuje se maksimalna brzina agenata, odnosno najveća udaljenost koju u jednom koraku agent može prijeći.
    e. Maksimalan broj iteracija
2. Odabir funkcije:
    a. Odabir jedne od ponuđenih ispitnih funkcija
       Nakon odabira neke od ispitnih funkcija, ona se iscrtava na prostoru za vizualni prikaz algoritma (lijevo) u rasponu od 0 do 500 po x i y koordinati.
    b. Odabir vrste optimizacije (minimum ili maksimum funkcije)
3. Opcije susjedstva:
    a. Odabirom globalnog susjedstva algoritam se izvodi u svom osnovnom obliku.
    b. Odabirom lokalnog susjedstva omogućuje se unos u prostor za odabir veličine susjedstva u obliku postotka veličine populacije. Algoritam se potom modificira po ranije objašnjenim pravilima lokalnog susjedstva.
4. *Frames per second*
   Ovaj parametar određuje brzinu izvođenja simulacije. Koristan je za praćenje rezultata pojedinih iteracija u slučaju da je postavljen na dovoljno malu vrijednost te brzog izvođenja velikog broja iteracija ako je postavljen na dovoljno veliku vrijednost.
5. Gumbi
    a. *Start* pokreće simulaciju izvođenja algoritma.
    b. *Stop* zaustavlja algoritam prije nego što su se završile sve iteracije.
    c. *Show map* otvara novi prozor u kojem je grafički i dinamički prikazan istraženi prostor agenata trenutnoj simulaciji algoritma.
    d. *Show statistics* otvara novi prozor u kojem je prikazana statistika o algoritmu koji se trenutno izvodi

Na lijevoj strani prozora nalazi se okvir za vizualni prikaz simulacije algoritma. Pri odabiru funkcije mijenja se i prikaz okvira. Iscrtava se odabrana funkcija na način da su vrijednosti prikazane nijansom boje. Svjetlina područja proporcionalna je s vrijednošću funkcije na određenoj poziciji; svijetla područja predstavljaju veće vrijednost, a tamna područja manje vrijednosti. Pritiskom na tipku *Start* u okviru se počinju iscrtavati agenti i njihovo kretanje.

Mapa istraženog prostora:

Pritiskom na gumb *Show map* otvara se dodatni prozor pored glavnog. Ovaj prozor sadrži prikaz funkcije koja se optimizira te bojom označena područja kojima su agenti prošli. Boja sadrži element prozirnosti što omogućava razlikovanje u područjima gdje su agenti prošli više puta od onih kojima su prošli manje puta. Također, zbog boljeg uvida u problem, označen je globali optimum funkcije većom crvenom točkom, najbolje pronađeno rješenje populacije dinamički se mijenja tijekom izvođenja algoritma i označen je manjom crvenom točkom. Slika 6.3. prikazuje mapu istraženog prostora u trenutku kada najbolje rješenje populacije još uvijek nije doseglo globalni optimum. Na slici su vidljivi svi navedeni elementi ovog prikaza.



Slika 6.3. Prikaz istraženog prostora

6.4.4. Ispitne funkcije

Za prikaz zapisa funkcija u izborniku na grafičkom korisničkom sučelju korištena su imena funkcija umjesto njihovih formula te pojednostavljene verzije dodatnih funkcija kako bi njihov prikaz bio što jednostavniji. U programskom ostvarenju sve su funkcije zbog potreba iscrtavanja na intervalu [0, *problem_size*] skalirane i transponirane. Na primjer, funkcija $f(x, y) = sin(x/30.0) - cos(y/30.0)$ u korisničkom je sučelju ponuđena kao $f(x, y) = sin(x) - cos(y)$.
Skaliranje funkcije obavlja se množenjem varijabli x i y s faktorom skaliranja *scale.* Faktor skaliranja računa se prema izrazu

$$scale_i = searchDomain_i / canvasSize_i$$

gdje je $searchDomain$ širina, odnosno visina prostora pretraživanja, a $canvasSize$ širina, odnosno visina prozora za crtanje.
Pomak funkcije po koordinatnim osima ostvaruje se zbrajanjem skalirane varijable s odabranim odmakom Δx, odnosno Δy. Ukupna transformacija po koordinatama tada iznosi:

$$x' = scale_x * x + \Delta x$$
$$y' = scale_y * y + \Delta y$$

U prozoru za prikaz statistike prikazana je ostvarena funkcija $f(x', y')$ kako bi se vrijednosti mogle pravilno analizirati.

U programskom rješenju ponuđeno je osam ispitnih funkcija:

1. $f(x, y) = sin(x/30.0) - cos(y/30.0)$
2. $f(x, y) = \sqrt{(x - 120)^2 + (y - 200)^2}$
3. Rosenbrockova funkcija: $f(x', y') = 100\left(y' - x'^2\right)^2 + (1 - x')^2$

$$scale_x = scale_y = \frac{4.17}{canvasSize}$$

$$\Delta x = -2$$
$$\Delta y = -1$$

4. Bukinova funkcija: $f(x', y') = 100\sqrt{|y' - 0.01x'^2|} + 0.01|x' + 10|$

$$scale_x = \frac{10}{canvasSize}$$
$$scale_y = \frac{6}{canvasSize}$$
$$\Delta x = 5$$
$$\Delta y = -3$$

5. Rastriginova funkcija: $f(x', y') = 20 + x'^2 - 10\cos(2\pi x') + y'^2 - 10\cos(2\pi y')$

$$scale_x = scale_y = \frac{10.24}{canvasSize}$$
$$\Delta x = \Delta y = -5.12$$

6. Himmelblauova funkcija: $f(x', y') = (x'^2 + y' - 11)^2 + (x' + y'^2 - 7)^2$

$$scale_x = scale_y = \frac{8}{canvasSize}$$
$$\Delta x = \Delta y = -4$$

7. Bealeova funkcija:
$f(x', y') = (1.5 - x' - x'y')^2 + (2.25 - x' + x'y'^2)^2 + (2.625 - x' + x'y'^3)^2$

$$scale_x = \frac{8}{canvasSize}$$
$$scale_y = \frac{6}{canvasSize}$$
$$\Delta x = -4$$
$$\Delta y = -3$$

8. Easomova funkcija: $f(x', y') = -\cos(x')\cos(y')\exp(-((x' - \pi)^2 + (y' - \pi)^2))$

$$scale_x = scale_y = \frac{20}{canvasSize}$$
$$\Delta x = \Delta y = -10$$

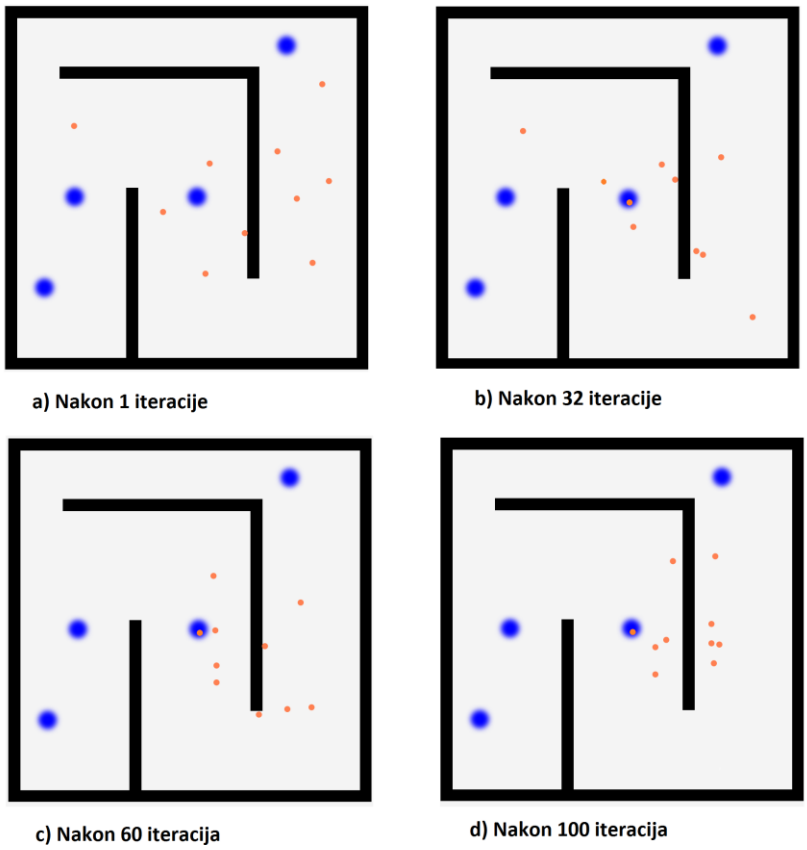
6.5. Rezultati ispitivanja

Ispitivanje rezultata algoritma provodi se u dva dijela. U prvom dijelu ispituje se učinkovitost algoritma pri pronalasku jednog ili više ciljeva u labirintu. Drugi dio ispitivanja provodi se na testnim funkcijama. Mijenjajući jedan parametar, dok ostali trebaju biti fiksni, ispituje se utjecaj tog parametra na ponašanje agenata i na efikasnost algoritma.

6.5.1. Testiranje algoritma roja čestica pri rješavanju problema labirinta

Postavke i parametri:

- Individualni faktor c1: 2.0
- Socijalni faktor c2: 2.5
- Maksimalna brzina agenata: 5% veličine domene
- Globalno susjedstvo
- Broj agenata: 10
- Maksimalan broj iteracija: 100



Slika 6.4. Rezultati ispitivanja algoritma pri rješavanju labirinta

Slika 6.4. prikazuje napredovanje algoritma u rješavanju algoritma. U 32. iteraciji jedna od čestica populacije pronalazi optimum, odnosno ciljno stanje algoritma. U idućim se iteracijama može primijetiti premala konvergencija populacije prema pronađenom globalnom optimumu. Čestice ostaju zaglavljene u neoznačenom prostoru. Osim nedostatka informacija u hodnicima labirinta, konvergenciju čestica prema izlazu sprječavaju i prepreke koje ih navode na promjenu smjera.

Može se zaključiti da je pronađeni optimum rezultat nasumičnog faktora koji vrši utjecaj na kretanje čestica u početnim iteracijama algoritma. Konvergencija populacije je spora i neefikasna zbog nedostatka informacija kojima čestice raspolažu i smislene heuristike. Osnovna izvedba algoritma, dakle, ne pokazuje se kao efikasna za rješavanje problema labirinta.

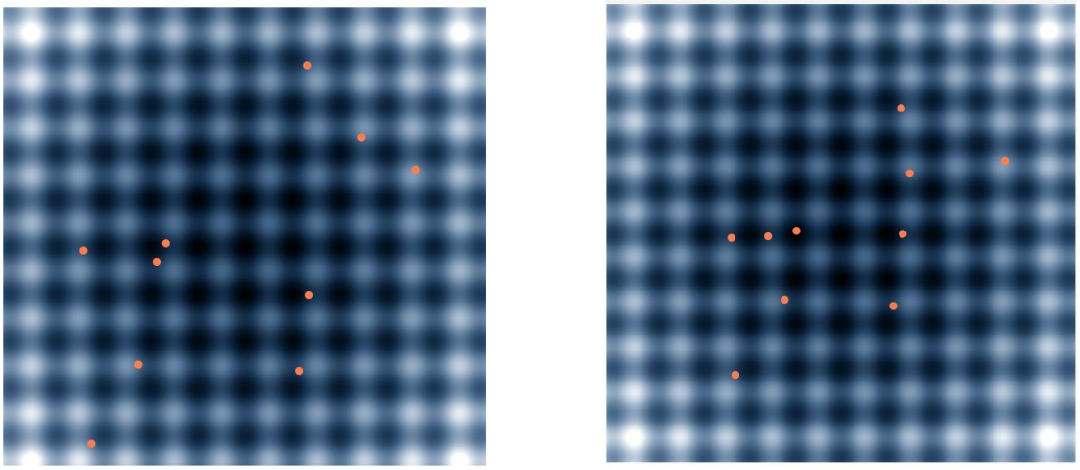
6.5.2. Utjecaj broja čestica na preciznost optimizacije funkcije uz konstantne parametre

Konstantne postavke i parametri:

- Testna funkcija: Rastriginova funkcija
  - Minimizacija
  - Globalni minimum: $f(300, 300) = 0$
- Individualni faktor c1: 2.0
- Socijalni faktor c2: 2.5
- Maksimalna brzina agenata: 5% veličine domene
- Globalno susjedstvo

Tablica 6.1. Rezultati ispitivanja za broj_agenata=1

Iteracija	Globalno najbolje rješenje	Najbolja pozicija	Prosječno rješenje
1	17.7051	(202, 298)	32.7781
3	6.1096	(238, 291)	19.5213
10	2.6379	(359, 245)	5.3683
20	1.0578	(358, 299)	3.6913
35	1.0234	(359, 300)	1.7012
50	1.0001	(358, 300)	1.3689
100	1.0001	(358, 300)	1.0323



a) Nakon 1 iteracije

b) Nakon 3 iteracije

c) Nakon 10 iteracija

d) Nakon 30 iteracija

e) Nakon 50 iteracija

f) Nakon 100 iteracija

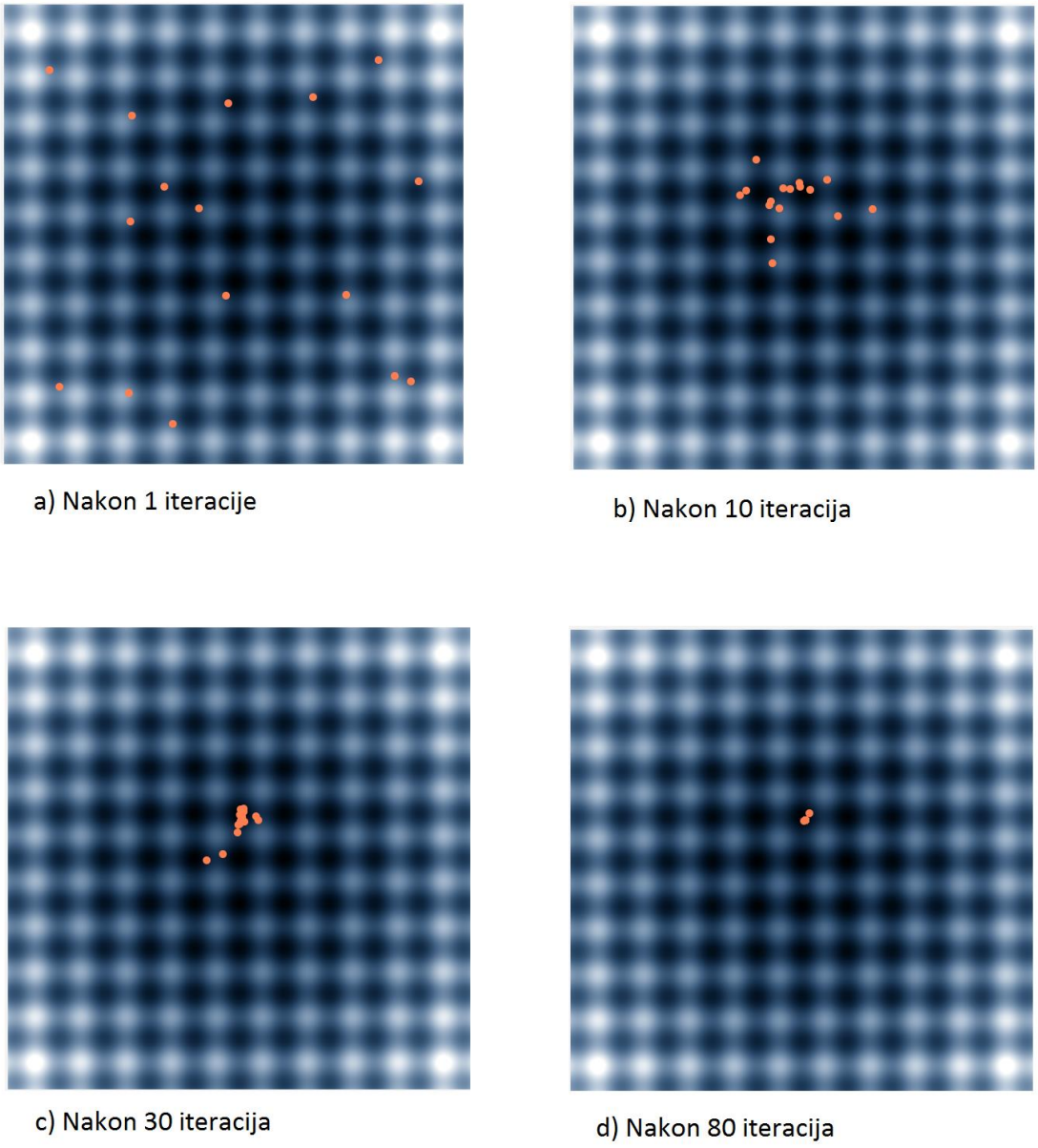
Slika 6.5. Rezultati ispitivanja za broj_agenata=1

Tablica 6.1. prikazuje najbolju pronađenu globalnu vrijednost i njezinu poziciju u pojedinim iteracijama te prosječnu vrijednost svih agenata određene iteracije. Već nakon 20 iteracija algoritam pronalazi dobru vrijednost kao globalni optimum – oko 1.06, a nakon 50 iteracija oko 1.0, te agenti počinju konvergirati prema njoj. Do 100-te iteracije prosječna vrijednost agenata sve se više približava 1.0 što znači da agenti istražuju sve manji i manji prostor oko pronađenog globalnog optimuma i da daljnjeg pretraživanja prostora neće ni biti.
Zaključuje se da su agenti „zaglavili“ u lokalnom optimumu te da je 10 agenata premalo za rješavanje konkretnog problema sa zadanim parametrima.

Na slici 6.5. vizualno je prikazano kretanje agenata kroz prostor rješenja u odabranim iteracijama algoritma. Vidljivo je da vrijednost 1.0 na poziciji (358, 300) uistinu jeste jedan od lokalnih optimuma.

Tablica 6.2. Rezultati ispitivanja za broj_agenata=15

Iteracija	**Globalno najbolje rješenje**	**Najbolja pozicija**	**Prosječno rješenje**
1	14.5989	(182, 256)	29.7238
3	6.5763	(188, 239)	19.9831
10	1.2198	(301, 240)	8.9267
25	1.0001	(300, 242)	2.9049
50	1.0001	(300, 242)	1.2430
100	1.0001	(300, 242)	1.0580



Slika 6.6. Rezultati ispitivanja za broj_agenata=15

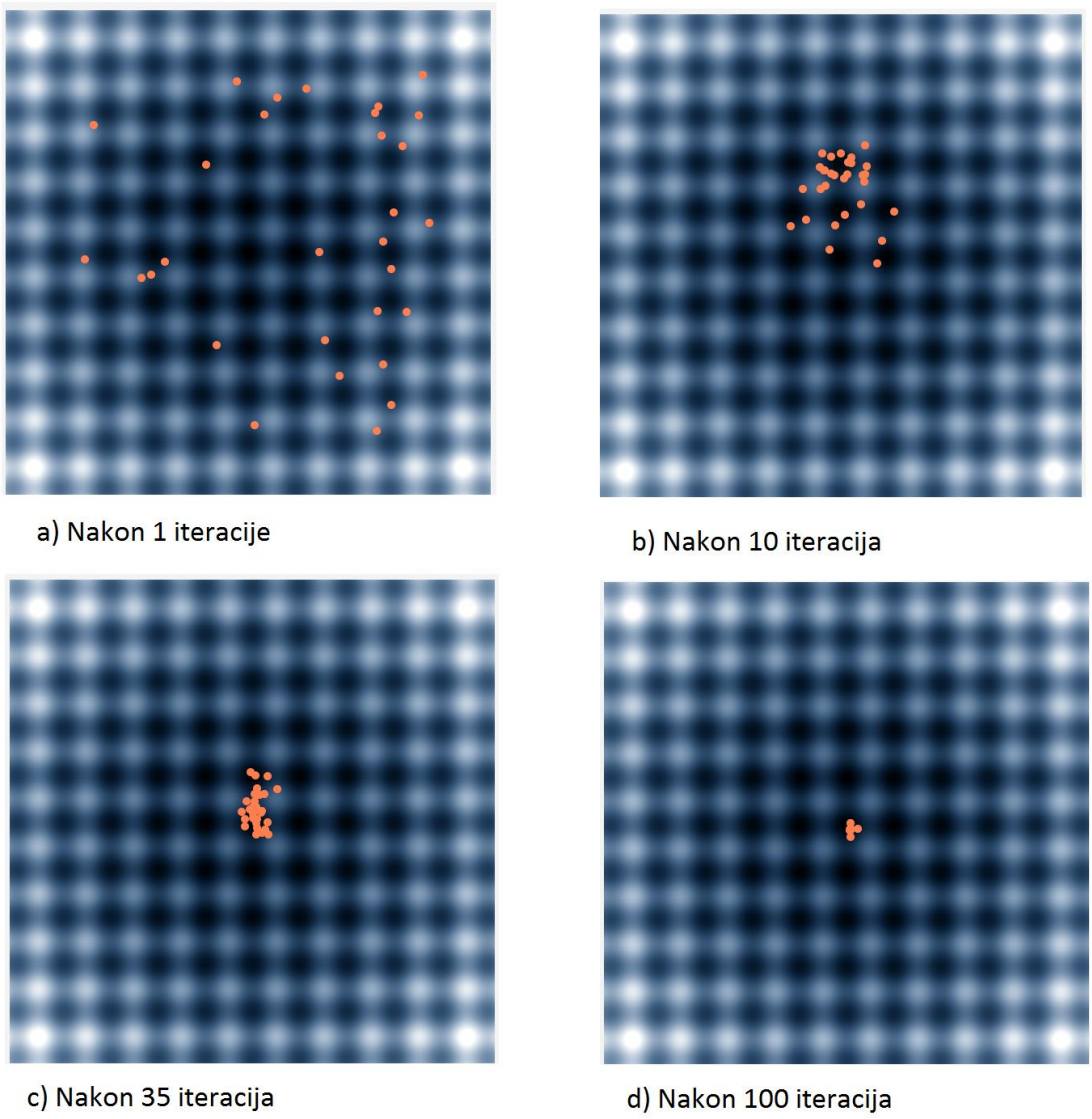
Tablica 6.2. prikazuje najbolju globalnu vrijednost, njezinu poziciju te prosječnu vrijednost agenata pojedine iteracije s veličinom populacije od 15 agenata. Algoritam već oko 25-te iteracije pronalazi optimum s vrijednošću 1.0 i zadržava tu vrijednost kao globalni optimum sve do 100-te iteracije, što znači da su agenti ponovno „zaglavili“ u lokalnom optimumu te da je 15 nedovoljan broj agenata za optimizaciju problema.

Uspoređujući s prethodnim slučajem s 10 agenata, optimum je pronađen ranije zbog mogućnosti populacije da brže pretraži prostor rješenja. Odstupanje prosječnog

rješenja agenata od najboljeg globalnog u 100-toj iteraciji predstavlja utjecaj individualnog faktora agenata koji potiče pretraživanje prostora.

Tablica 6.3. Rezultati ispitivanja za broj_agenata=30

Iteracija	Globalno najbolje rješenje	Najbolja pozicija	Prosječno rješenje
1	10.8484	(469, 300)	29.8456
3	5.4559	(243, 186)	21.5823
10	1.3222	(302, 243)	9.8231
33	0.0000	(300, 300)	2.2371
100	0.0000	(300, 300)	0.0116



Slika 6.7. Rezultati ispitivanja za broj_agenata=30

Tablica 6.3. prikazuje najbolju globalnu vrijednost, njezinu poziciju te prosječnu vrijednost agenata pojedine iteracije s veličinom populacije od 30 agenata.

U 33. iteraciji algoritam pronalazi globalni optimum s vrijednošću 0.0 na poziciji (300, 300), što i je globalni optimum funkcije. Promotri li se prosječno rješenje agenata po iteracijama, može se uočiti da nakon 33. iteracije agenti počinju konvergirati prema pronađenoj najboljoj poziciji, te do 100. iteracije gotovo svi imaju vrijednost 0.0.

Konvergencija prema globalnom optimumu funkcije po iteracijama je prikazana na slici 6.7.

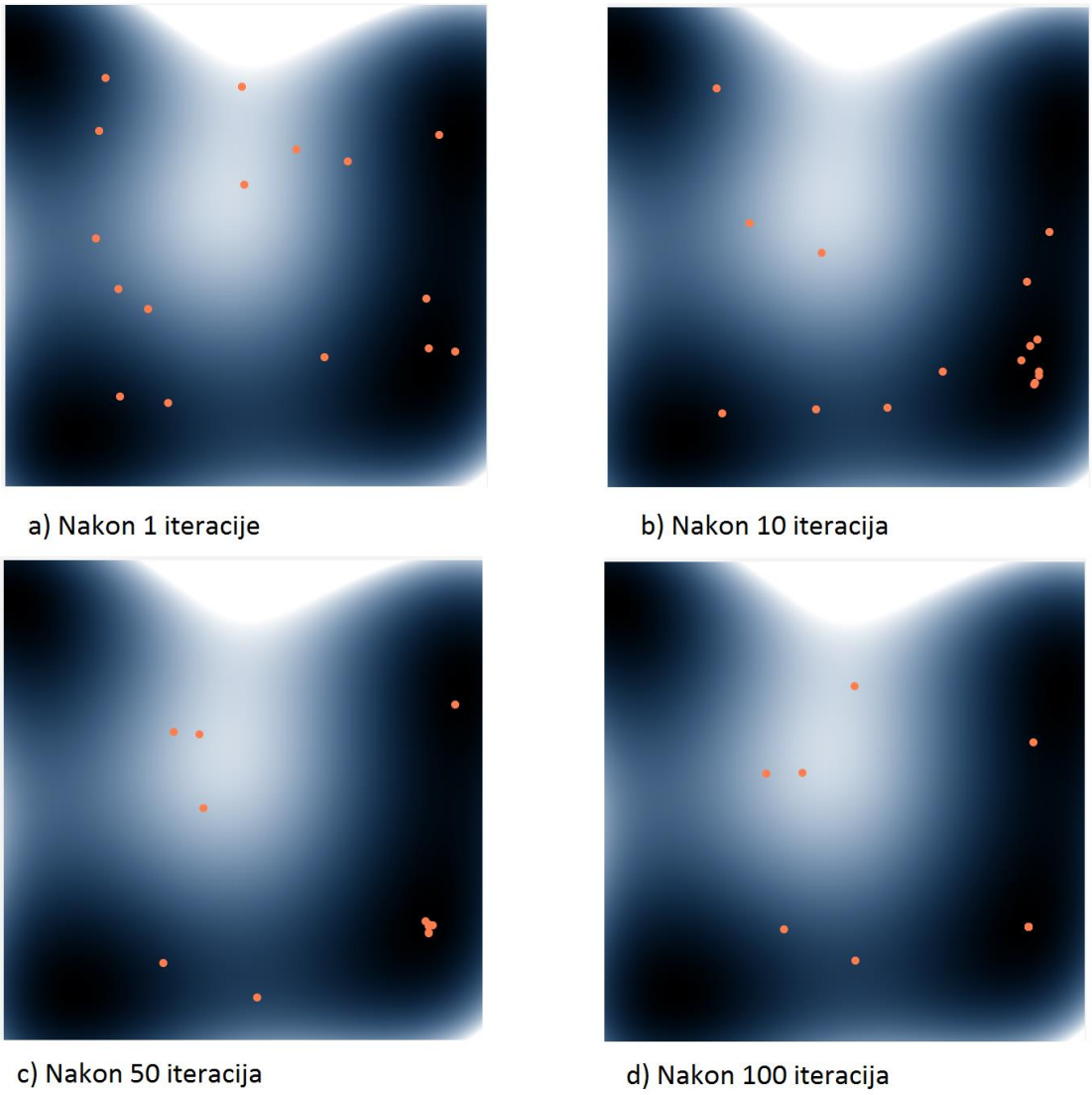
6.5.3. Utjecaj individualnog i socijalnog faktora na preciznost optimizacije funkcije uz konstantne parametre

Konstantne postavke i parametri:

- Testna funkcija: Himmelblauova funkcija
  - Minimizacija
  - Globalni minimum: $f(525, 450) = 0$
- Maksimalna brzina agenata: 5% veličine domene
- Globalno susjedstvo
- Broj agenata: 16

Tablica 6.4. Rezultati ispitivanja za individualni faktor c1=3.0 i socijalni faktor c2=1.3

Iteracija	Globalno najbolje rješenje	Najbolja pozicija	Prosječno rješenje
1	3.1964	(544, 415)	73.9590
5	0.02669	(525, 447)	48.5165
14	0.0000	(525, 450)	14.1695
50	0.0000	(525, 450)	12.3354
100	0.0000	(525, 450)	12.3310



Slika 6.8. Rezultati ispitivanja za individualni faktor c1=3.0 i socijalni faktor c2=1.3

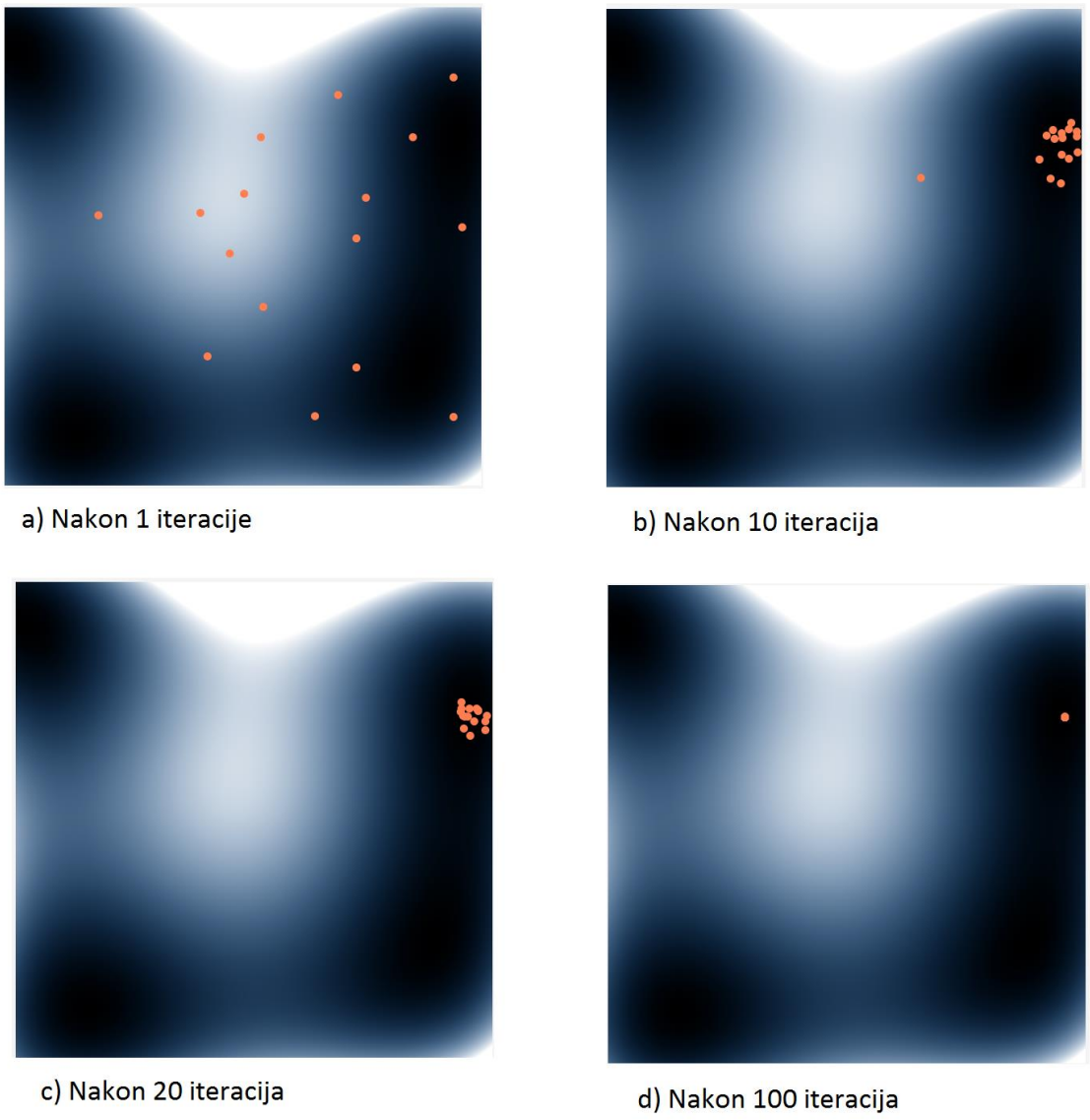
Tablica 6.4. prikazuje najbolju globalnu vrijednost, njezinu poziciju te prosječnu vrijednost agenata pojedine iteracije u slučaju kad je individualni faktor postavljen na relativno veliku vrijednost u odnosu na socijalni faktor.
Usporedbom najboljeg rješenja i prosječnog rješenja populacije po iteracijama, može se uočiti kako u njima postoji značajno odstupanje. Unatoč tome što je već u 14. iteraciji pronađen globalni optimum, agenti ne konvergiraju prema njemu nego i dalje pretražuju prostor rješenja.

Ovakvo ponašanje algoritma upravo je i očekivano jer je smanjenjem socijalnog faktora smanjen i utjecaj globalno najboljeg rješenja na jedinke populacije.

Slika 6.8. vizualno prikazuje pozicije agenata u pojedinim iteracijama.

Tablica 6.5. Rezultati ispitivanja za individualni faktor c1=1.1 i socijalni faktor c2=3.2

Iteracija	Globalno najbolje rješenje	Najbolja pozicija	Prosječno rješenje
1	13.2664	(571, 271)	91.6831
5	0.4120	(570, 174)	49.9764
10	0.0307	(567, 162)	11.0782
17	0.0073	(569, 163)	0.4380
50	0.0006	(569, 161)	0.0076
100	0.0006	(569, 161)	0.0006



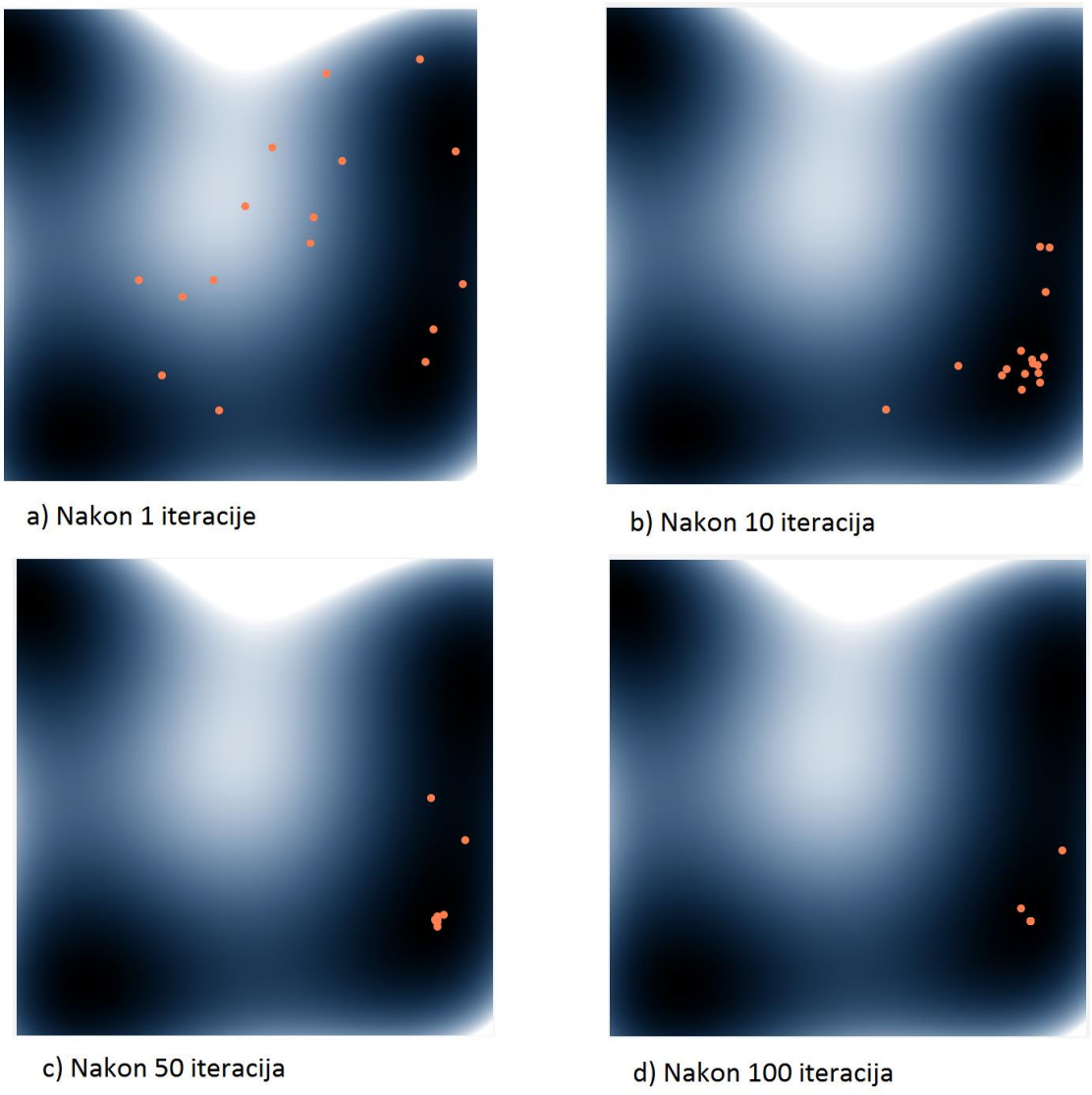
Slika 6.9. Rezultati ispitivanja za individualni faktor c1=1.1 i socijalni faktor c2=3.2

Tablica 6.5. prikazuje najbolju globalnu vrijednost, njezinu poziciju te prosječnu vrijednost agenata pojedine iteracije u slučaju kad je socijalni faktor postavljen na relativno veliku vrijednost u odnosu na individualni faktor.
S obzirom na postavljene vrijednosti faktora i promatranje prethodnog suprotnog slučaja, očekivano ponašanje algoritma je takvo da agenti u ranoj fazi pretraživanja počinju konvergirati prema pronađenom optimumu.
Usporedbom najboljeg globalnog i prosječnog rješenja po iteraciji iz tablice 6.5. uočavaju se približno jednake vrijednosti već nakon 17. iteracije. Također, na slici 6.9. jasno se može uočiti rana konvergencija agenata prema globalnom optimumu.

Unatoč tome što je u ovom slučaju pronađeno približno najbolje rješenje (0.0006≈ 0) u malo iteracija, ovakvo ponašanje algoritma nije poželjno. Česte su situacije gdje bi lokalni optimum bio prepoznat kao najbolji, te bi cijela populacija konvergirala prema njemu, propuštajući pritom priliku da pronađe globalni optimum.

Tablica 6.6. Rezultati ispitivanja za individualni faktor c1=1.7 i socijalni faktor c2=2.5

Iteracija	Globalno najbolje rješenje	Najbolja pozicija	Prosječno rješenje
1	0.2750	(530, 440)	91.5411
5	0.0494	(522, 452)	31.9193
14	0.0066	(526, 450)	3.5884
31	0.0000	(525, 450)	1.2720
50	0.0000	(525, 450)	0.4360
100	0.0000	(525, 450)	0.0233



Slika 6.10. Rezultati ispitivanja za individualni faktor c1=1.7 i socijalni faktor c2=2.5

Tablica 6.6. prikazuje najbolju globalnu vrijednost, njezinu poziciju te prosječnu vrijednost agenata pojedine iteracije u slučaju kada je socijalni faktor postavljen na nešto (ne značajno) veću vrijednost u odnosu na individualni faktor.
Od prve iteracije algoritam započinje s pronalaskom dobrih rješenja. Promatrajući promjenu prosječnog rješenja agenata po iteracijama, uočava se blaga konvergencija ali i odstupanje od globalnog optimuma.

Ovaj slučaj opisuje uravnoteženi utjecaj oba faktora na agente. Primjerice, odstupanje prosječnog od globalno najboljeg rješenja koje se može primijetiti po iteracijama rezultat je utjecaja individualnog faktora koji sprječava konvergenciju i potiče na

istraživanje prostora rješenja. Ipak, dovoljna sličnost globalno najboljeg i prosječnog rješenja u kasnijim iteracijama rezultat je utjecaja socijalnog faktora koji određuje utjecaj globalnog optimuma na kretanje jedinki populacije.

Na slici 6.10. vizualno su prikazane pozicije agenata u određenim iteracijama.

Ovakvim odabirom parametara ostvareno je željeno ponašanje algoritma. Prostor rješenja dovoljno je istražen, stoga je smanjena šansa da algoritam neće pronaći globalno najbolje rješenje. Također, nakon dovoljnog broja iteracija i faze istraživanja prostora, sve će jedinke konvergirati u pronađeni optimum.

7. Zaključak

Prirodom inspirirani algoritmi koriste se na širokom spektru optimizacijskih problema i područje su s velikim potencijalom koje sve više dolazi do izražaja. Jedan takav algoritam je optimizacija rojem čestica.

U ovom radu ostvareno je grafičko sučelje koje prikazuje kretanje roja čestica pri optimizaciji problema labirinta i dvodimenzionalnih funkcija, na kojima je rad algoritma i analiziran. Pokazalo se da za rješavanje problema labirinta algoritam nije učinkovit zbog nasumičnosti domene i nedostatka smislene heuristike. Nasuprot tome, pri rješavanju problema dvodimenzionalnih funkcija algoritam se izborom povoljnih parametara pokazao vrlo efikasnim te se kvalitetno mogao istražiti utjecaj parametara na učinkovitost algoritma.

Moguće daljnjeg istraživanje podrazumijeva proširenje problema na višedimenzionalne funkcije, izbor parametara u svrhu boljih rezultata, te modeliranje heuristike koja bi poboljšala pretragu na problemu labirinta.

Simulacija omogućava upravljanje izvođenjem algoritma i lijep prikaz načina na koji algoritam funkcionira.

Literatura

[1] Marko Čupić. *Prirodom inspirirani optimizacijski algoritmi. Metaheuristike.*, 2013.

[2] Jason Brownlee. *Clever Algorithms: Nature-Inspired Programming Recipes*, 2011.

[3] K.N. Krishnanand, D.Ghose. *Glowworm Swarm optimization for simultaneous capture of multiple local optima of multimodal functions*. Springer 2008.

[4] K.N. Krishnanand, D. Ghose, *Detection of multiple source locations using a glowworm metaphof with applications to collective robotics*. IEEE Swarm Intelligence Symposium, Pasadena, California, USA, 2005.

[5] Hamed Shah-Hosseini. *The intelligent water drops algorithm: a nature-inspired swarm optimization algorithm*, Inderscience Enterprises Ltd., 2009.

[6] Wikipedia. *Glowworm swarm optimization*. URL https://en.wikipedia.org/wiki/Glowworm_swarm_optimization

[7] Wikipedia. *Intelligent Water Drops algorithm*. URL https://en.wikipedia.org/wiki/Intelligent_Water_Drops_algorithm

[8] Wikipedia. *Test functions for optimization*. URL https://en.wikipedia.org/wiki/Test_functions_for_optimization