ELEKTROTEHNIŠKI VESTNIK 89(1-2): 46-52, 2022
IZVIRNI ZNANSTVENI ČLANEK

Primerjava evolucijskih algoritmov implementiranih v različnih programskih jezikih

Miha Ravber, Matej Moravec, Marjan Mernik

Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Koroška cesta 46, 2000 Maribor, Slovenija
E-pošta: miha.ravber@um.si

Povzetek. Uspešnost evolucijskih algoritmov se oceni na podlagi primerjave z obstoječimi algoritmi. Primerjava evolucijskih algoritmov v literaturi igra ključno vlogo, da lahko definiramo, kateri algoritmi so boljši. V večini primerov se avtorji z obstoječimi algoritmi primerjajo tako, da se primerjajo z rezultati iz člankov. Vendar ni zagotovljeno, da so bili eksperimenti izvedeni v skladu s smernicami za pošteno primerjavo. Ker je pravilna izvedba eksperimenta ključna, da lahko z gotovostjo potrdimo superiornost evolucijskih algoritmov, je priporočljivo, da se eksperimenti v celoti ponovijo. Le tako smo lahko prepričani, da je naš evolucijski algoritem primerljiv oz. boljši od obstoječih. Pri tem pa se lahko pojavi težava, če avtor algoritma, s katerim se želimo primerjati, ni podal izvorne kode algoritma ali pa ga je treba implementirati v drugem programskem jeziku. Pri takem pristopu so rezultati lahko zavajajoči. Do odstopanja lahko pride zaradi človeškega faktorja, ker avtor nenamenoma napačno implementira obstoječi algoritem. Drugi faktor, na katerega pogosto pozabimo, pa so razlike med programskimi jeziki, v katerih algoritem implementiramo. Ta problem smo izpostavili z analizo in primerjavo uspešnosti algoritma GWO, implementiranega v šestih programskih jezikih.

Ključne besede: evolucijski algoritmi, algoritem GWO, primerjava algoritmov, uspešnost algoritmov

Comparison of evolutionary algorithms implemented in different programming languages

The performance of the evolutionary algorithms is assessed by comparing them with the existing algorithms. Comparison of the evolutionary algorithms in the literature plays a key role as it illustrates which algorithms are better. It is a common practice to simply take the results from the papers of the algorithms you are comparing your proposed algorithm to. However, it is not guaranteed that the experiments are performed in accordance with the guidelines for a fair comparison. Since a correct execution of an experiment is crucial to be able to state with a certainty the superiority of evolutionary algorithms, it is recommended that the experiment is repeated. This, however, can be a challenge if the source code of the algorithm is not available or it needs to be implemented in another programming language. Such approach can lead to misleading results due to the unintentional mistakes made in implementing the algorithm. Another factor that is often overlooked are the differences between the programming languages in which the algorithm is implemented. The problem is highlighted by analyzing and comparing the performance of the GWO algorithm implemented in six programming languages.

Keywords: evolutionary algorithms, grey wolf optimizer, comparison of algorithms, algorithms performance

1 UVOD

Evolucijski algoritmi so se izkazali za zelo uspešne pri reševanju optimizacijskih problemov. Posledično lahko v literaturi zasledimo vse več novih in izboljšanih evolucijskih algoritmov. Avtorji novih evolucijskih algoritmov morajo te primerjati z obstoječimi najsodobnejšimi algoritmi z uporabo primerjalnih analiz (angl. Benchmarking). Primerjava evolucijskih algoritmov je zelo zapletena naloga, ki vključuje veliko dejavnikov, ki jih je treba upoštevati, da se zagotovi poštena in nepristranska primerjava. Pri primerjavi moramo biti pozorni na to, kateri optimizacijski problemi bodo vključeni v primerjalno analizo, s katerimi algoritmi se bomo primerjali, kako nastaviti kontrolne parametre algoritmov, katero statistično metodo uporabiti in kako nastaviti zaustavitveni kriterij. V literaturi lahko zasledimo veliko člankov, ki poskušajo olajšati delo raziskovalcem tako, da podajo smernice za primerjavo evolucijskih algoritmov [1][2][3]. Torej, kadar želimo izvesti primerjavo z obstoječimi algoritmi, je dobro upoštevati te smernice, da zagotovimo čim bolj pošteno primerjavo.

Avtorji novih algoritmov pogosto obstoječe rezultate (grafe, tabele in slike) primerjajo s svojimi, pa vendar je treba, če v člankih, s katerimi se primerjajo, niso upoštevali smernic, eksperiment ponoviti tako, da sami zaženemo algoritme. Prav tako moramo celotni

Prejet 17. februar, 2022
Odobren 14. marec, 2022

eksperiment ponoviti, če smo pri izvedbi uporabili drugačne parametre. Toda izvorna koda algoritmov ni zmeraj na voljo ali pa algoritmi niso pravilno implementirani.

V literaturi lahko zasledimo veliko člankov o pravilni primerjavi evolucijskih algoritmov in tem, da je treba nov algoritem primerjati z najsodobnejšimi (angl. state-of-the-art) algoritmi [1][2][3]. Vendar je redko omenjeno, da morajo biti za pošteno primerjavo algoritmi implementirani tudi v istem programskem jeziku. V sorodnih delih so primerjali vpliv implementacije in programskih jezikov na hitrost izvajanja algoritmov [7][8][9][10][11], v tem članku pa ne bomo primerjali hitrosti izvajanja ali učinkovitosti algoritmov, temveč njihovo uspešnost. Primerjali so tudi razliko med različicami istega algoritma. Za posamezne algoritme, kot je CMA-ES [4], obstaja več različic, ki se velikokrat sploh ne omenjajo, vendar je med njimi signifikantna razlika [5]. Mi smo prikazali signifikantno razliko, tudi kadar gre za eno samo različico algoritma.

Cilj naše študije je torej bil, da pri primerjavi evolucijskih algoritmov poudarimo problem nepravilno implementiranih algoritmov in razlike med implementacijami v različnih programskih jezikih.

Za primerjavo smo izbrali evolucijski algoritem GWO (angl. Grey Wolf Optimizer) [17], ker je ta zelo popularen in ker je na avtorjevi strani na voljo tudi izvorna koda algoritma GWO v različnih programskih jezikih [13]. Avtor algoritma GWO ni avtor vseh različic, ki so na voljo. Vendar lahko sklepamo, da je algoritem implementiran pravilno, če je izvorna koda objavljena na njegovi spletni strani. Z avtorjeve strani smo prenesli algoritem GWO, implementiran v šestih programskih jezikih: Java, C, C++, Python, R in MATLAB. Ker algoritma nismo implementirali sami, lahko izločimo svoje implementacijske napake. Ko smo prenesli vse različice, smo jim nastavili iste parametre. Empirične študije smo izvedeli s svojim ogrodjem EARS (angl. Evolutionary Algorithms Rating System) [14]. V ogrodju se algoritmi primerjajo z uporabo šahovskega sistema rangiranja za primerjavo evolucijskih algoritmov (angl. Chess Rating System for Evolutionary Algorithms - CRS4EAs) [6]. CRS4EAs vrne lestvico, ki vsebuje vse sodelujoče algoritme, kar nam omogoča enostavno odkrivanje statistično značilnih razlik med algoritmi.

V naslednjem poglavju na kratko opišemo delovanje algoritma GWO. V 3. poglavju opišemo metodologijo primerjave, ki smo jo uporabili v eksperimentih. Eksperimentalne rezultate in analizo izvornih kod predstavimo v poglavju 4. Sledi 5. poglavje, kjer podamo zaključne misli in priporočila za raziskovalce.

2 ALGORITEM GREY WOLF OPTIMIZER

Algoritem GWO je evolucijski algoritem, ki so ga predlagali avtor Seyedali Mirjalili in drugi [17]. Navdih so dobili tako, da so posnemali obnašanje črede sivih volkov pri lovljenju plena. V matematičnem modelu evolucijskega algoritma GWO je najboljša rešitev predstavljena kot volk alfa (α). Druga in tretja najboljša rešitev sta predstavljeni kot volkova beta (β) in delta (δ). Vse preostale rešitve so tretirane kot volkovi omega (ω). Lovljenje plena oz. iskanje optimalne rešitve pri evolucijskem algoritmu GWO deluje tako, da volkovi omega sledijo volkovom alfa, beta in delta ter tako poskušajo najti globalni optimum. Za iskanje optimalne rešitve so implementirani trije glavni koraki pri lovljenju plena: iskanje plena, obkrožanje plena in napad na plen.

Psevdokoda na sliki 1 prikazuje delovanje evolucijskega algoritma GWO pri reševanju problema. Proces iskanja najboljše rešitve se začne z generiranjem naključne populacije agentov možnih rešitev – sivih volkov. Za vsako možno rešitev se izračuna funkcija uspešnosti in shranijo trije najboljši agenti (alfa, beta, delta). Evolucijski algoritem nato skozi več iteracij na podlagi pozicij treh najboljših agentov posodablja pozicije preostalih iskalnih agentov (angl. search agent) in oceni morebitno lokacijo plena (možne rešitve) ter izračuna oddaljenost od plena (rešitve). V ta namen so uporabljene naslednje enačbe.

$$\overrightarrow{D_\alpha} = |\overrightarrow{C_1} \cdot \overrightarrow{X_\alpha} - \vec{X}|, \overrightarrow{D_\beta} = |\overrightarrow{C_2} \cdot \overrightarrow{X_\beta} - \vec{X}|, \overrightarrow{D_\delta} = |\overrightarrow{C_3} \cdot \overrightarrow{X_\delta} - \vec{X}| \quad (1)$$

$$\overrightarrow{X_1} = \overrightarrow{X_\alpha} - \overrightarrow{A_1} \cdot (\overrightarrow{D_\alpha}), \overrightarrow{X_2} = \overrightarrow{X_\beta} - \overrightarrow{A_2} \cdot (\overrightarrow{D_\beta}), \overrightarrow{X_3} = \overrightarrow{X_\delta} - \overrightarrow{A_3} \cdot (\overrightarrow{D_\delta}) \quad (2)$$

$$\vec{X}(t+1) = \frac{(\overrightarrow{X_1} + \overrightarrow{X_2} + \overrightarrow{X_3})}{3} \quad (3)$$

Algoritem GWO pri iskanju najboljše rešitve uporablja parametre a, $\vec{A}$ in $\vec{C}$. Parametra a in $\vec{A}$ pri iskanju najboljše rešitve predstavljata naključno oz. prilagodljivo vrednost, ki se uporabita pri iskanju plena (eksploracija) in pri napadu na plen (eksploatacija). Parameter a se zmanjšuje od 2 proti 0 z namenom poudarjanja eksploracije in eksploatacije [12]. V primeru, ko je vrednost $\overrightarrow{|A|}$ večja od 1, se od plena oz. možne rešitve oddaljujemo. Kadarkoli pa je vrednost $\overrightarrow{|A|}$ manjša od 1, se k plenu oz. možni rešitvi približujemo. Na tak način evolucijski algoritem GWO skozi več generacij poišče najboljšo rešitev.

```
1: inicializacija populacije iskalnih agentov X_i(i = 1, 2, ..., n)
2: inicializacija a, A in C
3: izračun funkcije uspešnosti za vsakega agenta
4: X_α = najboljši iskalni agent
5: X_β = drugi najboljši iskalni agent
6: X_δ = tretji najboljši iskalni agent
7: while t < največje število iteracij do
8:     for vsak iskalni agent do
9:         posodobi pozicijo trenutnega iskalnega agenta po enačbi (3)
10:    end for
11:    posodobi a, A in C
12:    izračun funkcije uspešnosti za vse iskalne agente
13:    posodobi X_α, X_β in X_δ
14:    t = t + 1
15: end while
16: return X_α
```

Slika 1: Psevdokoda evolucijskega algoritma GWO.

3 METODOLOGIJA PRIMERJAVE

Za primerjavo evolucijskega algoritma GWO, implementiranega v različnih programskih jezikih, smo uporabili šahovski sistem rangiranja. Ta uporablja določene enačbe in pravila, ki vplivajo na igralčev rating. Rating posameznega igralca se spremeni vedno, ko se udeleži kakega organiziranega turnirja in tam doseže določeno število zmag, porazov in remijev. V splošnem velja, da je rating pozitivno celo število, ki predstavlja uspešnost igralca. Višji je rating igralca, uspešnejši je igralec. Šahovski sistem rangiranja se je prvotno uporabljal le za rangiranje igralcev pri strateški igri šah. Podobno pa lahko uporabo šahovskega sistema rangiranja preslikamo na področje primerjave evolucijskih algoritmov. Tako v prispevku med seboj primerjamo implementacije evolucijskega algoritma GWO v različnih programskih jezikih. Rezultati posameznih implementacij se na turnirju med seboj primerjajo in vsaki implementaciji je dodeljen rating glede na število zmag, porazov in remijev. Dodeljeni rating predstavlja uspešnost posamezne implementacije evolucijskega algoritma GWO in služi za razvrstitev na lestvici, kjer so implementacije rangirane od najboljše oz. tiste z najvišjim ratingom do najslabše oz. tiste z najnižjim ratingom.

V splošnem obstaja več šahovskih sistemov rangiranja. Za primerjavo implementacij evolucijskega algoritma GWO v različnih programskih jezikih smo v našem primeru uporabili odprtokodno ogrodje EARS, ki uporablja sistem Glicko-2. Algoritmi so pri uporabi ogrodja EARS predstavljeni kot šahisti. Na turnirju tekmujejo v dvojicah, kjer vsak algoritem (igralec) odigra igro z vsemi drugimi algoritmi. Igra je za vsak problem odigrana N-krat. V našem prispevku je vsak algoritem odigral 30 iger z vsemi drugimi algoritmi za vsak posamezni problem. Algoritma tekmujeta pod istimi pogoji (isti zaustavitveni pogoj) in pri tem želi vsak najti čim boljšo rešitev za dani problem. Rešitvi se nato primerjata in določi se izid igre. Ta je lahko zmaga enega algoritma in poraz drugega ali pa je izid neodločen. Po končanem turnirju se na podlagi rezultatov izračunajo rating R, odklon ratinga RD (angl. Rating Deviation) in interval ratinga RI (angl. Rating Interval). Končni rezultat turnirja predstavlja lestvica vseh algoritmov, ki so na lestvici razvrščeni glede na njihov rating, pri čemer višji rating pomeni boljši rezultat in pomeni igralčevo absolutno moč. Odklon ratinga pove, kako zanesljiv je igralčev rating. Intervali ratinga oz. intervali zanesljivosti so izračunani glede na rating R in odklon ratinga RD. Glede na interval ratinga [R - 2RD, R + 2RD] lahko s 95 % gotovostjo trdimo, da igralčev rating R leži na tem intervalu. Statistični pomen je določen glede na to, ali se intervali ratinga prekrivajo. Kadar se dva intervala ne prekrivata, pomeni, da sta izvedbi primerjanih algoritmov signifikantno različni. V članku [18] so avtorji pokazali, da je metoda CRS4EAs primerljiva s Friedmanovim statističnim testom, ki mu sledi Nemenyijev statistični test. Hkrati CRS4EAs daje zanesljive rezultate že pri manjšem številu neodvisnih zagonov. V članku [19] so avtorji pokazali, da je CRS4EAs primerljiva tudi s pristopom pDSC (Practical Deep Statistical Comparison).

Za primerjavo uspešnosti delovanja implementacij evolucijskega algoritma GWO v različnih programskih jezikih, smo uporabili enomodalne (angl. unimodal) optimizacijske probleme, ki imajo samo en optimum in so prikazani v Tabela 1. Ti problemi so: Ackley (f_1),

Tabela 1: Enomodalni optimizacijski problemi.

Funkcija uspešnosti	d	Območje	f_{min}
$f_1(x) = -a\exp(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}) - \exp(\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)) + a + \exp(1)$	100	$[-32,768, 32,768]$	0
$f_2(x) = (x_1 - 1)^2 + \sum_{i=2}^{d}(2x_i^2 - x_{i-1})^2$	100	$[-10.0, 10.0]$	0
$f_3(x) = \sum_{i=1}^{d}\frac{x_i^2}{4000} - \prod_{i=1}^{d}\cos(\frac{x_i}{\sqrt{i}}) + 1$	100	$[-600, 600]$	0
$f_4(x) = 10d + \sum_{i=1}^{d}[x_i^2 - 10\cos(2\pi x_i)]$	100	$[-5, 12, 5, 12]$	0
$f_5(x) = \sum_{i=1}^{d-1}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	100	$[-5, 10]$	0
$f_6(x) = 1 - \cos(2\pi\|\|x\|\|)) + 0,1\|\|x\|\|$	100	$[-100, 100]$	0
$f_7(x) = -\sum_{i=1}^{n}[x_i\sin(\sqrt{\|x_i\|})]$	100	$[-500, 500]$	-418,98d*
$f_8(x) = \sum_{i=1}^{d}(x_i - 1)^2 - \sum_{i=2}^{d} x_i x_{i-1}$	100	$[-100^2, 100^2]$	$\frac{-d(d+4)(d-1)}{6}$
$f_9(x) = \sum_{i=1}^{d} x_i^2 + (\sum_{i=1}^{d} 0,5ix_i)^2 + (\sum_{i=1}^{d} 0,5ix_i)^4$	100	$[-5, 10]$	0
$f_{10}(x) = \sum_{i=1}^{d/4}[(x_{4i-3} + 10x_{4i-2})^2 + 5(x_{4i-1} - x_{4i})^2 + (x_{4i-2} - 2x4_{4i-1})^4 + 10(x_{4i-3} - x_{4i})^4]$	100	$[-4.0, 5.0]$	0

* $f_{min} = -418,98288727243379980791360139 8d$

DixonPrice (f_2), Griewank (f_3), Rastrigin (f_4), Rosenbrock (f_5), Salomon (f_6), Schwefel (f_7), Trid (f_8), Zakharov (f_9) in Powell (f_{10}). Vsem smo določili isto dimenzijo d=100. Za vsak optimizacijski problem je podano tudi območje iskanja in globalni optimum f_{min}.

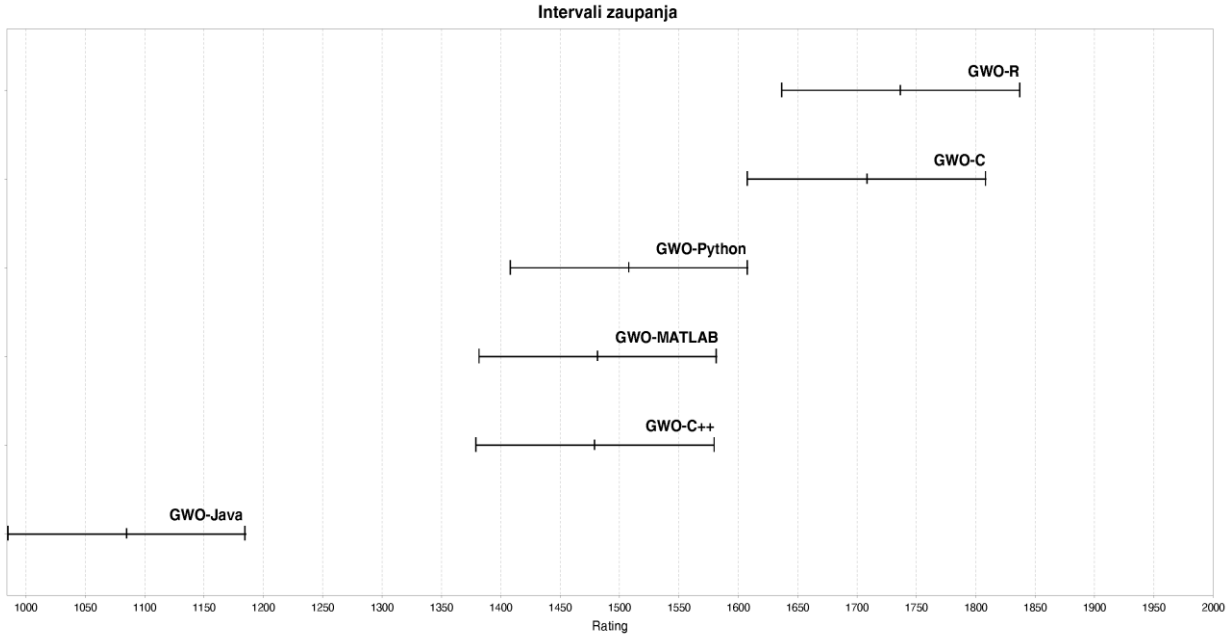
4 EKSPERIMENTALNI REZULTATI

V tem poglavju predstavimo rezultate primerjave algoritma GWO, implementiranega v različnih programskih jezikih. Da bi zagotovili pošteno primerjavo, moramo vsem implementacijam algoritma določiti iste parametre. Edini nastavljivi parameter v algoritmu GWO je velikost populacije (število iskalnih agentov). To smo nastavili na vrednost 30, ki jo je uporabil tudi avtor v izvorni kodi algoritma GWO, napisanega v programskem jeziku MATLAB. Zaustavitveni pogoj je bil maksimalno število iteracij, ki smo ga nastavili na 500. Za vsak optimizacijski problem smo algoritem izvedli 30-krat.

Rezultat primerjave so intervali zaupanja, ki jih lahko vidimo na sliki 2. Implementacije so razvrščene glede na rating, kjer je najboljša zgoraj desno in najslabša spodaj levo. Torej, najbolje se je odrezala implementacija v jeziku R (GWO-R) in najslabše implementacija v jeziku Java (GWO-Java). Vendar pri primerjavi ne moremo primerjati samo ratinga, ampak tudi interval zaupanja. Če se intervala dveh algoritmov ne prekrivata, je prišlo do statistično značilne razlike v uspešnosti. Bolj se intervala prekrivata, manjša je razlika v uspešnosti. Ker imamo tu opravka z nedeterminističnimi algoritmi, bi to tudi pomenilo, da bi se lahko s ponovnim zagonom eksperimenta vrstni red takšnih algoritmov spremenil. Če torej pogledamo intervale zaupanja, lahko trdimo, da sta GWO-R in GWO-C statistično značilno boljša od vseh preostalih implementacij. Ker se njuna intervala v večini prekrivata, je njuna uspešnost zelo podobna in lahko pričakujemo tudi, da bi se njun vrsti red ob ponovnem zagonu eksperimenta lahko zamenjal. Podobno velja za implementacije GWO-Python, GWO-MATLAB in GWO-C++, kjer se njihovi intervali v večini prekrivajo. Omenjene implementacije so statistično značilno boljše od GWO-Java. S slike je torej jasno razvidno, da je med implementacijami prišlo do signifikantnih razlik. Pri tem ponovno poudarimo, da avtorji pri uporabi algoritma GWO običajno ne navajajo, katero implementacijo so izbrali, zato takšni eksperimenti niso ponovljivi [3].

4.1 Analiza izvorne kode

Da bi ugotovili, zakaj je prišlo do takšnih razlik med implementacijami, smo analizirali izvorno kodo vseh implementacij. Za osnovo smo vzeli implementacijo v jeziku MATLAB, ker jo je sprogramiral avtor algoritma GWO. Primerjavo smo izvedli tako, da smo vzporedno zagnali implementacijo v MATLAB-u in implementacijo v programskem jeziku, s katerim jo primerjamo. Ker evolucijski algoritmi spadajo pod stohastično optimizacijo, moramo pred primerjavo izločiti naključje. To lahko izvedemo tako, da uporabimo isti psevdonaključni generator števil, ki je na voljo v vseh programskih jezikih, kot je Mersenne Twister [15]. Nato pri zagonu vsake implementacije uporabimo isto seme. Vendar zaradi zaokrožitvene napake tudi s tem ne moremo zagotoviti povsem enakih števil na različnih platformah, zato smo se raje odločili, da vnaprej zgeneriramo naključna števila, ki jih jemljemo iz krožne vrste (angl. round-robin). Ko smo izločili naključje, smo lahko zagnali kodo in po njej korakali vrstico za vrstico. Za implementaciji v programskih jezikih Python in C++ nismo opazili razlik v primerjavi z implementacijo v



Slika 2: Intervali zaupanja algoritma GWO, implementiranega v različnih programskih jezikih, z nespremenjeno izvorno kodo.

programskem jeziku MATLAB. Razlike v implementaciji so se kot pričakovano pokazale pri implementacijah, ki so statistično značilno različne (GWO-C, GWO-R in GWO-Java). V naslednjih poglavjih je podrobnejši opis razlik implementacij v teh jezikih.

4.1.1 Analiza izvorne kode GWO-Java

Zaradi vzporednega zagona implementacij smo lahko hitro opazili zaokrožitvene napake in vsako manjše odstopanje v kodi. V GWO-Java se je razlika pokazala že na začetku, saj se tu iteracija začne z 1, medtem ko se v GWO-MATLAB začne z 0. Po ponovnem zagonu smo opazili napako pri popravljanju meja pozicije iskalnih agentov. Kadar gre posameznik zunaj meja iskalnega prostora, ki je definiran za posamezni problem (Tabela 1), ga moramo vrniti znotraj meja, saj je takšna rešitev neveljavna. Običajni pristop, ki se uporablja, je projekcija [20], kjer se vrednosti, ki gredo čez mejo, postavijo na mejno vrednost. Na primer, če je iskalni prostor omejen z [-100, 100] in gre vrednost čez 100, jo nastavimo na 100, in če gre vrednost pod -100, jo nastavimo na -100. Tako tudi deluje popravljanje meja v GWO-MATLAB. Če gre v implementaciji GWO-Java vrednost zunaj meja, se zgenerira nova naključna vrednost znotraj meja. Vendar je v tem odseku kode, ki skrbi za generiranje novih števil, napaka, zaradi katere se lahko generirajo števila zunaj mej. Ta napaka je tudi glavni razlog, zakaj se je GWO-Java odrezal tako slabo. Implementacija se razlikuje tudi v tem, kdaj se metoda za popravljanje meja uporabi. Meje moramo preveriti in popraviti, preden ovrednotimo iskalnega agenta (Slika 1, vrstica 12), v GWO-Java pa se metoda za popravljanje meja uporabi tudi nad vektorji $\overrightarrow{X_1}, \overrightarrow{X_2}\ in\ \overrightarrow{X_3}$ (enačba (2)). Zadnja razlika, ki smo jo odkrili, je, da se v GWO-Java iskalni agenti alfa, beta in delta v vsaki generaciji zamenjajo z agenti iz trenutne populacije, čeprav so lahko slabši. Avtor je to napako delno omilil tako, da je uvedel elitizem, torej je v populacijo zmeraj vključil agenta alfa.

4.1.2 Analiza izvorne kode GWO-C

V implementaciji GWO-C smo najprej opazili, da ne uporabljajo privzetega psevdonaključnega generatorja števil. Algoritem smo poskusili zagnati s privzetim generatorjem števil in generatorjem števil Mersenne Twister, vendar ni bilo večjih razlik. Tudi v GWO-C so uporabili drugo metodo za popravljanje meja. Če gre agent zunaj meja, se nova vrednost izračuna iz povprečja agentov alfa, beta in delta. Opazili smo tudi nekaj napak. Implementacija na začetku ne nastavi agentov alfa, beta in delta, kadar imajo agenti v populaciji previsoko vrednost funkcije uspešnost (angl. fitness function). Pri izračunu parametra a pride do zaokrožitvene napake, ker se v enačbi uporablja napačen podatkovni tip. Največja napaka je pri izračunu vektorjev $\overrightarrow{X_1}, \overrightarrow{X_2}\ in\ \overrightarrow{X_3}$ (enačba (2)). V kodi so pri branju pozicij agentov alfa, beta in delta uporabili indeks agenta namesto indeks dimenzije.

4.1.3 Analiza izvorne kode GWO-R

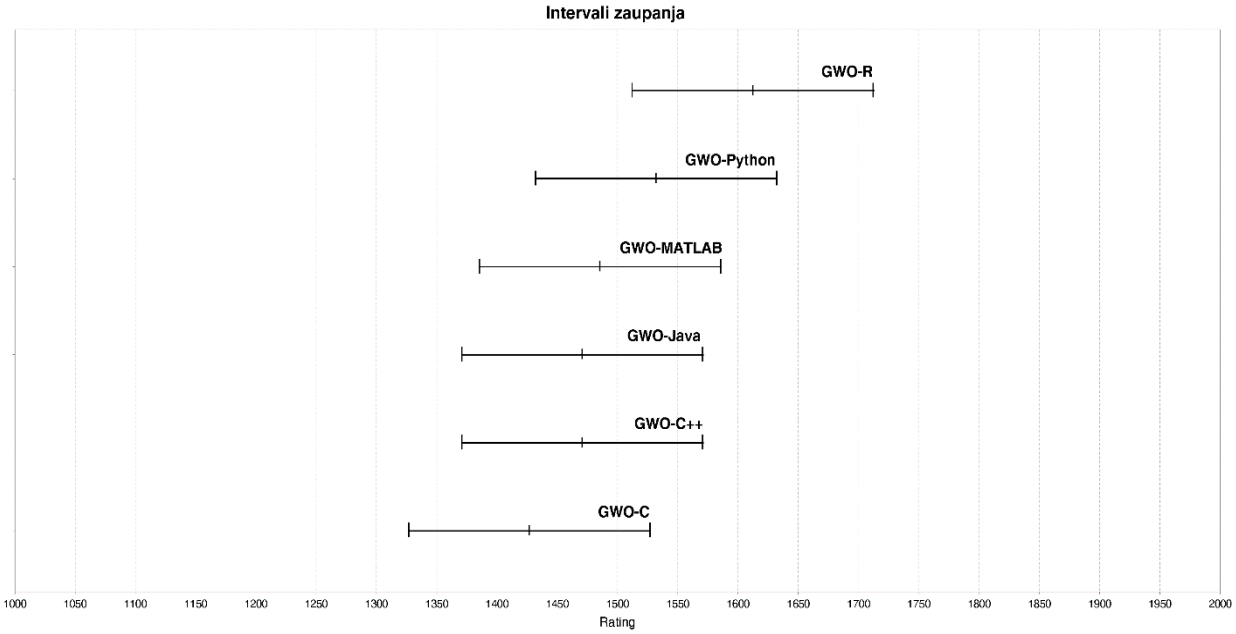
V implementaciji GWO-R smo odkrili samo dve razliki. Prva je zelo zanemarljiva: kot v GWO-Java se tudi v GWO-R iteracija začne z 1 namesto 0. Druga razlika pa je veliko bolj signifikantna. Logika za posodabljanje agentov alfa, beta in delta se izvaja znotraj zanke (Slika 1, vrstice od 8 do 10). V originalni implementaciji se ti agenti posodobijo samo enkrat na iteracijo, v tem primeru pa se posodobijo za vsako dimenzijo v iteraciji.

4.2 Primerjava popravljenih implementacij

Ko smo odpravili vse razlike in napake v različnih implementacijah, smo ponovno izvedli primerjavo. Intervali zaupanja popravljenih implementacij so prikazani na sliki 3. Iz rezultatov je razvidno, da med implementacijami ni več statistično značilnih razlik, seveda pa rezultati niso popolnoma enaki. Razlogi za to so stohastična narava evolucijskih algoritmov in zaokrožitvene napake pri izračunih.

5 SKLEP

Pravilnost implementacije in programski jezik imata lahko velik vpliv na uspešnost evolucijskih algoritmov, torej je ključnega pomena, da je algoritem implementiran pravilno in v istem programskem jeziku, da lahko pravilno reproduciramo eksperimente. V članku smo prikazali veliko odstopanj med rezultati iste različice algoritma GWO, implementiranega v različnih programskih jezikih. Te razlike smo demonstrirali na implementacijah, vzetih iz zaupanja vrednega vira, to je s spletne strani avtorja. S tem smo še dodatno podkrepili resnost te težave. V rezultatih eksperimenta je med implementacijami prišlo do statistično značilnih razlik. Podrobna analiza izvorne kode je pokazala, da so avtorji storili napake pri implementaciji ali pa v članku niso upoštevali navodil. Do razlik je prihajalo tudi zaradi drugačnega delovanja zaokrožitve med programskimi jeziki. Po popravkih med implementacijami ni bilo več statistično značilnih razlik. Ključni cilj tega prispevka je pritegniti pozornost raziskovalcev in jim pokazati, da lahko tudi z avtorjeve uradne strani prenesemo napačno implementiran algoritem in da morajo upoštevati razlike med programskimi jeziki. Zato raziskovalcem svetujemo, da zmeraj preverijo, ali izvorna koda deluje pravilno. Prav tako je pomembno, da raziskovalci navedejo vir izvorne kode, da lahko preverimo delovanje algoritma in pravilno ponovimo eksperiment. Rezultati se seveda zaradi občutljivosti evolucijskih algoritmov in zaokrožitvenih napak ne bodo nikoli popolnoma ujemali, a bomo izločili vsaj napake in razlike v izvorni kodi.



Slika 3: Intervali zaupanja algoritma GWO, implementiranega v različnih programskih jezikih, s popravljeno izvorno kodo.

ZAHVALA

Raziskavo je omogočila Javna agencija za raziskovalno dejavnost Republike Slovenije (ARRS) v okviru programa P2-0041 – Računalniški sistemi, metodologije in inteligentne storitve.

LITERATURA

[1] Bartz-Beielstein Thomas, Carola Doerr, Daan van den Berg, Jakob Bossek, Sowmya Chandrasekaran, Tome Eftimov, Andreas Fischbach. "Benchmarking in optimization: Best practice and open issues." *arXiv preprint arXiv:2007.03488* (2020).

[2] LaTorre Antonio, Daniel Molina, Eneko Osaba, Javier Del Ser, Francisco Herrera. "Fairness in bio-inspired optimization research: A prescription of methodological guidelines for comparing meta-heuristics." *arXiv preprint arXiv:2004.09969* (2020).

[3] López-Ibáñez Manuel, Juergen Branke, Luís Paquete. "Reproducibility in evolutionary computation." *ACM Transactions on Evolutionary Learning and Optimization* 1, no. 4 (2021): 1-21.

[4] Hansen Nikolaus, Andreas Ostermeier. "Completely derandomized self-adaptation in evolution strategies." *Evolutionary computation* 9.2 (2001): 159-195.

[5] Biedrzycki, Rafał. "Comparison with State-of-the-Art: Traps and Pitfalls." In *2021 IEEE Congress on Evolutionary Computation (CEC)*, str. 863-870. IEEE, 2021.

[6] Veček Niki, Marjan Mernik, Matej Črepinšek. "A chess rating system for evolutionary algorithms: A new method for the comparison and ranking of evolutionary algorithms." *Information Sciences* 277 (2014): 656-679.

[7] Alba Enrique, Edgardo Ferretti, Juan M. Molina. "The influence of data implementation in the performance of evolutionary algorithms." *International Conference on Computer Aided Systems Theory*. Springer, Berlin, Heidelberg, 2007.

[8] Merelo J. J., Gustavo Romero, Maribel García Arenas, Pedro A. Castillo, Antonio Miguel Mora, Juan Luis Jiménez Laredo. "Implementation matters: Programming best practices for evolutionary algorithms." *International Work-Conference on Artificial Neural Networks*. Springer, Berlin, Heidelberg, 2011.

[9] Merelo-Guervós, Juan Julián, Israel Blancas-Alvarez, Pedro A. Castillo, Gustavo Romero, Pablo García-Sánchez, Victor M. Rivas, Mario García-Valdez, Amaury Hernández-Águila, Mario Román. "Ranking the Performance of Compiled and Interpreted Languages in Genetic Algorithms." *International Conference on Evolutionary Computation Theory and Applications*. Vol. 2. SCITEPRESS, 2016.

[10] Merelo-Guervós, Juan-Julian, Israel Blancas-Álvarez, Pedro A. Castillo, Gustavo Romero, Victor M. Rivas, Mario García-Valdez, Amaury Hernández-Águila, Mario Román. "A comparison of implementations of basic evolutionary algorithm operations in different languages." *2016 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2016.

[11] Merelo Juan-J., Pablo García-Sánchez, Mario García-Valdez, Israel Blancas. "There is no fast lunch: an examination of the running speed of evolutionary algorithms in several languages." *arXiv preprint arXiv:1511.01088* (2015).

[12] Črepinšek Matej, Shih-Hsi Liu, Marjan Mernik. "Exploration and exploitation in evolutionary algorithms: A survey." *ACM computing surveys (CSUR)* 45.3 (2013): 1-33.

[13] Izvorna koda algoritma GWO dostopna na avtorjevi spletni strani, https://seyedalimirjalili.com/gwo (2. 2. 2022).

[14] Ogrodje EARS dostopno na GitHub repozitoriju, https://github.com/UM-LPM/EARS (2. 2. 2022).

[15] Matsumoto Makoto, Takuji Nishimura. "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator." *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 8.1 (1998): 3-30.

[16] Faris Hossam, Ibrahim Aljarah, Seyedali Mirjalili, Pedro A. Castillo, Juan Julián Merelo Guervós. "EvoloPy: An Open-source Nature-inspired Optimization Framework in Python." In IJCCI (ECTA), pp. 171-177. 2016.

[17] Mirjalili, Seyedali, Seyed Mohammad Mirjalili, Andrew Lewis. "Grey wolf optimizer." *Advances in engineering software* 69, str. 46-61, 2014.

[18] Veček Niki, Matej Črepinšek, Marjan Mernik. "On the influence of the number of algorithms, problems, and independent runs in the comparison of evolutionary algorithms." *Applied Soft Computing* 54 (2017): 23-45.

[19] Eftimov Tome, Peter Korošec. "Identifying practical significance through statistical comparison of meta-heuristic stochastic optimization algorithms." *Applied Soft Computing* 85 (2019): 105862.

[20] Biedrzycki, Rafał. "Handling bound constraints in CMA-ES: An experimental study." *Swarm and Evolutionary Computation* 52 (2020): 100627.

Miha Ravber je leta 2012 diplomiral, leta 2015 magistriral in leta 2018 doktoriral na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru, in sicer s področja računalništva in informacijskih tehnologij. Od leta 2015 je zaposlen v Laboratoriju za programirne metodologije. Njegovo raziskovalno področje obsega evolucijsko računanje, eno- in večkriterijske optimizacije in algoritme po vzoru iz narave.

Matej Moravec je leta 2017 diplomiral in leta 2019 magistriral na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru, in sicer s področja računalništva in informacijskih tehnologij. Od leta 2019 je zaposlen v Laboratoriju za programirne metodologije.

Marjan Mernik je leta 1994 magistriral, leta 1998 pa doktoriral na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru, in sicer s področja računalništva. Je redni profesor in vodja Laboratorija za programirne metodologije. V letih 2007 do 2018 je bil gostujoči profesor na univerzi University of Alabama at Birmingham. Njegovo raziskovalno področje so programski jeziki, domensko specifični (modelirni) jeziki in evolucijski algoritmi. Je glavni in odgovorni urednik revije Journal of Computer Languages in odgovorni urednik revij Applied Soft Computing, Information Sciences in Swarm and Evolutionary Computation. V letih 2017 in 2018 je bil visokocitirani raziskovalec.