ELEKTROTEHNIŠKI VESTNIK 78(5): 275-280, 2011
EXISTING SEPARATE ENGLISH EDITION

Izbor v diferencialni evoluciji

Iztok Fajfar, Janez Puhan, Sašo Tomažič in Árpád Bűrmen

Univerza v Ljubljani, Fakulteta za elektrotehniko, Tržaška 25, 1000 Ljubljana, Slovenija
e-pošta: iztok.fajfar@fe.uni-lj.si

Povzetek. Diferencialna evolucija je enostaven algoritem za globalno optimizacijo. V grobem je sestavljen iz treh operacij: mutacije, križanja in izbora. Medtem ko obstaja mnogo znanstvenih člankov, ki se ukvarjajo s prvima dvema, je tretja operacija deležna komaj kakšne pozornosti, niti ni zanjo predvidenega mesta v originalnem načinu poimenovanja različic algoritma. V prispevku smo pokazali, da uporaba različnih postopkov izbora v kombinaciji z naključno perturbacijo vektorjev populacije vodi k opazno boljšemu delovanju na večrazsežnostnih problemih

Ključne besede: globalna optimizacija, direktni iskalni postopki, diferencialna evolucija, hevristični postopki

On Selection in Differential evolution

Differential evolution is a simple algorithm for global optimization. Basically it consists of three operations: mutation, crossover and selection. Despite many research papers dealing with the first two, hardly any attention has been paid to the third one nor is there a place for this operation in the algorithm basic naming scheme. In the paper we show that employing different selection strategies combined with some random perturbation of population vectors notably improves performance in high-dimensional problems.

1 UVOD

Diferencialna evolucija (DE) je enostaven, a kljub temu učinkovit algoritem za globalno optimizacijo. Razvila sta ga Storn in Price [1]. V zadnjem desetletju postaja algoritem zaradi svoje izjemne enostavnosti in dobrih konvergenčnih lastnosti čedalje bolj priljubljen v raziskovalnih in inženirskih krogih. Algoritem DE pripada razredu evolucijskih algoritmov (EA), katerih obnašanje posnema biološke procese genetskega dedovanja in preživetja najsposobnejših. Ena bistvenih prednosti EA pred ostalimi numeričnimi optimizacijskimi postopki je, da ne zahteva, da je kriterijska funkcija niti odvedljiva niti zvezna. Zato so ti algoritmi primerni za reševanje široke palete problemov.

DE se začne z generacijo *NP* naključno izbranih *D*-razsežnostnih parametrskih vektorjev. Nove vektorje dobimo tako, da izbranemu vektorju prištejemo uteženo razliko dveh drugih vektorjev. Tej operaciji pravimo *mutacija*. Parametre mutiranega vektorja nato mešamo s parametri še enega vektorja, ki ga imenujemo *ciljni vektor*. Rezultat mešanja parametrov je *poskusni vektor*, operacija pa se v krogih EA običajno imenuje *križanje*. Na koncu primerjamo poskusni vektor s ciljnim, in če ta predstavlja slabšo rešitev, ga nadomestimo s poskusnim. To zadnjo operacijo poimenujemo *izbor*. V vsaki generaciji nastopa vsak vektor enkrat kot ciljni vektor.

Obstaja veliko različic algoritma DE [2], med katerimi se najpogosteje uporablja različica *DE/rand/1/bin*. Zato to različico obravnavamo tudi v tem članku. Pred uporabo algoritma je potrebno določiti vrednosti treh parametrov, ki vplivajo na delovanje algoritma DE. Odločiti se moramo za velikost populacije *NP* ter za vrednosti dveh krmilnih parametrov – skalirnega faktorja *F* in faktorja križanja *CR*. Izbira teh parametrov je običajno odvisna od konkretnega problema, kar zahteva od uporabnika določene izkušnje. Raziskovalci so doslej poskušali ugnati problem z mnogimi adaptivnimi ali samoadaptivnimi strategijami za določanje krmilnih parametrov *F* in *CR* [3, 4, 5] in celo velikosti populacije *NP* [6, 7]. Drugi so proučevali in predlagali različne strategije mutacije in križanja [8, 9, 10]. Z izborom – tretjim operatorjem, ki nastopa v DE – ni bilo doslej narejenih nobenih izrecnih raziskav. Niti ni za ta operator določenega mesta v poimenovalnem vzorcu za različne različice algoritma DE (t.j. DE/x/y/z). V tem članku raziskujemo, kako vplivajo različne strategije izbora na obnašanje algoritma DE, še zlasti na njegovo zmožnost izogibanja lokalnim minimumom oz. stagnaciji. Poleg tega smo algoritmu dodali enostavno operacijo, ki bi jo v genetskih algoritmih poimenovali mutacija – z določeno verjetnostjo smo naključno spreminjali parametre vektorjev populacije. Ker je pojem mutacija v algoritmu DE že rezerviran, smo ta poseg poimenovali *naključna perturbacija*.

V naslednjem razdelku na kratko opišemo delovanje algoritma DE, v 3. razdelku pa predlagamo naključno perturbacijo vektorjev in različne strategije izbora.

Prejet 25. november, 2011
Odobren 8. december, 2011

Rezultate preizkusov na testnih funkcijah predstavimo v 4. razdelku.

2 KRATEK PREGLED DIFERENCIALNE EVOLUCIJE

Imejmo kriterijsko funkcijo $f: \mathbb{R}^n \to \mathbb{R}$, kjer imamo cilj najti minimum $\vec{a} \in \mathbb{R}^n$, tako da $\forall \vec{b} \in \mathbb{R}^n: f(\vec{a}) \leq f(\vec{b})$. V tem primeru poimenujemo $\vec{a}$ globalni minimum. V resničnih problemih je redko mogoče najti točen globalni minimum, zato moramo iz praktičnih razlogov sprejeti kandidata, ki ponuja razumno dobro rešitev.

Z namenom, da bi našla globalni minimum, uporablja diferencialna evolucija *NP* *D*-razsežnostnih parametrskih vektorjev $x_{i,G}$, i=1,2,...,*NP* kot populacijo v generaciji *G*. *NP* se iz generacije v generacijo ne spreminja. Začetno populacijo določimo naključno in tako – če niso znani nobeni podatki o kriterijski funkciji – da uniformno pokrije celoten parametrski prostor.

Med optimizacijskim postopkom ustvarimo nove parametrske vektorje tako, da prištejemo uteženo razliko dveh naključno izbranih vektorjev tretjemu vektorju: $v_{i,G+1}=x_{r1,G}+F\cdot(x_{r2,G}-x_{r3,G})$ s celoštevilskimi, med sabo različnimi, naključnimi indeksi $r_1,r_2,r_3 \in \{1,2,...,NP\}$, ki morajo biti prav tako vsi različni od i, ter z realnim konstantnim členom $F \in [0,2]$. Tej operaciji pravimo *mutacija*, dobljenemu vektorju pa *mutiran* vektor.

Parametre mutiranega vektorja nato mešamo s tretjim vektorjem, t.i. *ciljnim* vektorjem. Rezultat je *poskusni* vektor $u_{i,G+1}=(u_{1i,G+1},u_{2i,G+1},...,u_{Di,G+1})$, kjer

$$u_{ji,G+1} = \begin{cases} v_{ji,G+1} & \text{če} (randb(j) \leq CR) \text{ ali } j = rnbr(i) \\ x_{ji,G} & \text{če} (randb(j) > CR) \text{ in } j \neq rnbr(i) \end{cases}' \quad j = 1,2,\dots,D. \tag{1}$$

Pri tem je *randb*(*j*) *j*-ti izračun uniformnega naključnega generatorja z izhodom ∈ [0,1], *CR* je uporabniško določena konstanta ∈ [0,1] in *rnbr*(*i*) je naključni indeks ∈ {1,2,...,*D*}. Slednji zagotavlja, da dobi poskusni vektor vsaj en parameter mutiranega vektorja. Ta operacija mešanja parametrov se običajno imenuje *križanje*.

Na koncu se izvede še *izbor*, v katerem se odloči, ali naj poskusni vektor postane član generacije *G*+1. Vrednost kriterijske funkcije v poskusnem vektorju $u_{i,G+1}$ se primerja z njeno vrednostjo v ciljnem vektorju $x_{i,G}$ po principu pohlepa: ciljni vektor zamenjamo le, če dobimo s poskusnim vektorjem boljšo vrednost kriterijske funkcije. V nasprotnem primeru poskusni vektor zavržemo, ciljnega pa obdržimo.

3 SPREMEMBE ORIGINALNEGA ALGORITMA DE

Naše delo je osredotočeno na del algoritma po križanju, t.j. na trenutek, ko je poskusni vektor že oblikovan.

Navdih za spremembe smo dobili iz naslednjega opažanja. Kadar se stopnja križanja *CR* približuje 1, ne preživi prav dosti ciljnega vektorja v njegovem potomcu (poskusnem vektorju). V tem smislu je mogoče sklepati, da je smer iskanja v smeri od ciljnega proti poskusnemu vektorju prav tako dobra (ali slaba) kot katerakoli druga smer. Preizkusiti smo želeli hipotezo, da utegne obstajati kakšen drug (po možnosti boljši) kandidat za zamenjavo kot sam ciljni vektor.

V nadaljevanju bomo predlagali in preizkusili tri različna pravila izbora kandidata, ki bo tekmoval s poskusnim vektorjem. Kandidata označimo s $c_{i,G}$ in ga izberemo v skladu z enim od naslednjih treh kriterijev:

$$\begin{aligned} &\text{Cr1: } c_{i,G} = x_{n,G}, \exists n: \min_n \left(d\left(x_{n,G}, x_{i,G}\right)\right) \forall n: f\left(u_{i,G+1}\right) < f\left(x_{n,G}\right) \\ &\text{Cr2 :} c_{i,G} = x_{n,G}, \exists n: \min_n \left(d\left(x_{n,G}, u_{i,G+1}\right)\right) \forall n: f\left(u_{i,G+1}\right) < f\left(x_{n,G}\right) \\ &\text{Cr3: } c_{i,G} = x_{n,G}, \exists n: \begin{cases} n = i, f\left(u_{i,G+1}\right) < f\left(x_{i,G}\right) \\ \text{najmanjši } n \in \left\{1,2,\dots \frac{NP}{2}\right\}: f\left(u_{i,G+1}\right) < f\left(x_{n,G}\right), \text{sicer} \end{cases} \end{aligned} \tag{2}$$

kjer pomeni $d(\cdot,\cdot)$ Evklidovo razdaljo. Pripomniti velja, da tu še vedno obstajajo primeri, ko ni izbran noben kandidat. Takrat poskusni vektor zavržemo.

Pod pogojem Cr1 od vseh vektorjev, pri katerih ima kriterijska funkcija slabšo vrednost kot pri poskusnem vektorju, zamenjamo tistega, ki leži geometrično najbliže ciljnemu vektorju. Naj omenimo, da tako kot originalen algoritem tudi ta strategija vedno zamenja ciljni vektor, če je ta slabši od poskusnega. Sicer za zamenjavo išče kandidata, ki je ciljnemu vektorju čim bliže. Tako kot v originalnem algoritmu, obstaja tudi tu večja verjetnost, da zamenjamo ciljne vektorje s slabo vrednostjo kriterijske funkcije, medtem ko bodo tisti z boljšo vrednostjo preživeli. Poleg tega se bo nek bližnji vektor premaknil na mesto, kamor bi se premaknil ciljni vektor, če ne bi bil slabši od poskusnega. To pospeši kopičenje članov populacije okrog članov z relativno boljšo vrednostjo kriterijske funkcije. Po eni strani lahko to znatno pospeši konvergenco, po drugi pa obstaja nevarnost, da prezgodaj izgubimo potrebno raznolikost in zato ne najdemo globalnega minimuma.

Pristop s kriterijem Cr2 je precej različen v tem, da išče kandidata, ki je geometrično najbliže poskusnemu vektorju. V tem smislu se izvajajo zamenjave v prid

krajšim skokom, ki spodbujajo iskanje v sicer manj obetajočih območjih.

Zasnova kriterija Cr3 na prvi pogled ni tako očitna. Podobno kot pri originalnem algoritmu in Cr1 najprej preverimo, ali lahko zamenjamo ciljni vektor, t.j. ali je vrednost kriterijske funkcije v poskusnem vektorju boljša kot v ciljnem. Sicer zamenjamo prvega člana prve polovice populacije, čigar vrednost kriterijske funkcije je slabša od poskusnega vektorja. Osnovna ideja, ki se skriva za tem je, da imamo pol populacije, ki se razvija pod pogoji originalnega algoritma DE, drugo polovico pa pospešimo z dodatnimi menjavami. In tudi te dodatne menjave so nesimetrične in pogosteje prizadenejo člane z nižjimi indeksi. Tako smo želeli v originalni algoritem vpeljati čim manjšo spremembo, hkrati pa izvesti relativno močan dodaten pritisk na omejeno število članov populacije.

Preden začnemo z eksperimenti, dodajmo algoritmu še eno majhno modifikacijo. Zanimivo je opažanje, da čeprav sam algoritem spada v razred metahevrističnih in stohastičnih postopkov, v njem naključnost neposredno ne nastopa. Naključnost se pojavi pri izbiri vektorjev, ki nastopajo v gradnji mutiranega vektorja in pri križanju. Sami parametri vektorjev se spreminjajo naključno le posredno prek mutacije in križanja, dobljene vrednosti pa so omejene na linerane kombinacije parametrov, ki so že vsebovani v populaciji. Nekateri avtorji so že vpeljali dodatno naključnost bodisi neposredno z naključnim spreminjanjem vektorskih parametrov [11, 12] bodisi posredno z naključnim spreminjanjem krmilnih parametrov *F* in *CR* [13, 14].

V naši študiji smo se odločili, da enostavno mutiramo s konstantno verjetnostjo vsak posamezni parameter poskusnega vektorja neposredno pred postopkom izbora:

$$u_{ji,G+1} = \begin{cases} rand(j), \text{če}(randb(j) \leq 0.05) \\ u_{ji,G+1}, \text{sicer} \end{cases},$$

$$j = 1,2,\ldots,D, \qquad (3)$$

kjer je *rand*(*j*) klic naključnega generatorja, ki vrača uniformno porazdeljene vrednosti vzdolž celotne *j*-te osi parametrskega prostora. Konstantno verjetnost 0,05 smo določili empirično s pomočjo nekaj preliminarnih preizkusnih tekov algoritma. Ti preizkusi so pokazali tudi, da daje uniformna porazdelitev prek celotnega parametrskega prostora v našem primeru boljše rezultate kot normalna porazdelitev okrog obstoječih vrednosti parametrov, ki se pogosto uporablja v literaturi. To operacijo smo poimenovali *perturbacija*.

4 REZULTATI

4.1 Splošno delovanje

Z namenom, da bi dobili splošen in prvi vtis, kako na delovanje algoritma vplivajo trije predlagani kriteriji izbora in naključna perturbacija, smo izvedli enostaven preizkus. V ta namen smo izbrali 14 standardnih funkcij iz [15], 13 visoko- (D=30) in eno nizkorazsežnostno (D=4) funkcijo. Potem smo naključno izbrali tri parametre z intervalov $NP \in \{10,\ldots,100\}$, $F \in [0,1]$ in $CR \in [0,1]$ ter inicializirali naključno populacijo *NP* parametrskih vektorjev znotraj omejitev, ki so določene za vsako testno funkcijo. Nato smo izvedli osem optimizacijskih tekov v dolžini 150.000 izračunov kriterijskih funkcij (IKF) z enakimi vrednostmi parametrov in začetno populacijo vektorjev, toda vsakič z drugo shemo izbora ter s perturbacijo ali brez nje. To smo ponovili 5.000 krat za vsako testno funkcijo, vsakič z drugačnimi vrednostmi parametrov in začetno populacijo vektorjev. Rezultati so zbrani v tabeli 1.

Tabela 1: Primerjava različnih modifikacij algoritma z originalnim algoritmom

Postopek izbora	**Brez perturbacije**		**S perturbacijo**	
	50.000 IKF	***150.000 IKF***	***50.000 IKF***	***150.000 IKF***
Original	–	–	44,1/51,7	43,7/44,1
Cr1	53,5/43,4	45,1/48,0	69.9/26,9	63,1/29,0
Cr2	52,8/44,1	45,7/47,4	62,4/34,4	57,3/35,4
Cr3	61,4/34,6	53,0/37,1	75,2/20,6	68,5/20,5

Štirinajst parov števil v tabeli pomeni sedem različnih primerjav (vsako od modifikacij smo posebej primerjali z originalom) ob dveh različnih trenutkih teka algoritma: po 50.000 IKF in po 150.000 IKF. Števec predstavlja odstotek primerov, ko je dala boljše rezultate ustrezna modifikacija (ob natančnosti šestih značilnih mest), medtem ko imenovalec govori o odstotku primerov, ko smo dobili boljši rezultat z originalnim algoritmom. Vsota je na splošno manjša od 100, kajti v nekaj primerih sta obe različici dali enak rezultat. Štetje smo izvedli prek vseh primerov, ne glede na vrednosti krmilnih parametrov ali testno funkcijo. V resničnih problemih inženir pogosto ve zelo malo ali nič o kriterijski funkciji in posledično o najboljši nastavitvi krmilnih parametrov. Zato se zdi, da je povprečenje prek širokega spektra različnih funkcij in nastavitev krmilnih parametrov, izbranih po principu Monte Carlo, primerno merilo splošne učinkovitosti algoritma.

Pari števil v belo obarvanih celicah v tabeli pomenijo stanje po 50.000 IKF. Predvidevamo, da na tej stopnji algoritem na splošno še ne doseže polne konvergence. Kot posledica ti pari števil bolj kažejo na hitrost konvergence in ne toliko na zmožnost algoritma, da doseže globalni optimum.

Številke v sivo obarvanih celicah kažejo stanje po 150.000 IKF. Na tej stopnji predpostavljamo, da je primerov, ko je konvergenca dosežena, znatno več kot po 50.000 IKF. Zato menimo, da ti rezultati odražajo zmožnost algoritma, da najde dobro končno rešitev.

Iz tabele lahko izluščimo nekaj precej zanimivih ugotovitev. Če namesto ciljnega vektorja zamenjamo

kandidata, ki je najbliže ciljnemu vektorju (Cr1) ali poskusnemu vektorju (Cr2) in pri tem ne uporabimo perturbacije, dobimo le za spoznanje boljše rezultate po 50.000 IKF (1. stolpec, 2. oz. 3. vrstica) in za spoznanje slabše po 150.000 IKF (2. stolpec, 2. oz. 3. vrstica). Iz tega lahko sklepamo, da pogostejše menjave vektorjev v obeh primerih pospešijo konvergenco, kot smo pričakovali, vendar na splošno na koncu pogosteje obtičijo v lokalnem minimumu ali povzročijo stagnacijo. To pa se ni zgodilo pri strategiji izbora, ki uporablja kriterij Cr3. Ta strategija je po 50.000 IKF skoraj 2-krat boljša od originalne in je še vedno precej boljša tudi po 150.000 IKF (4. vrstica, 1. oz. 2. stolpec). Pomembno je opozoriti, da tudi pri tej spremembi algoritem izvede več zamenjav kot v originalni izvedbi, kar očitno pospeši konvergenco. Poglavitna razlika je v tem, da se te dodatne zamenjave izvajajo le na omejenem številu članov populacije, medtem ko so preostali še vedno izpostavljeni originalnemu postopku izbora. Tehnično gledano lahko govorimo o dveh različnih postopkih, ki tečeta vzporedno.

Primerjava originalnega postopka s perturbacijo in brez nje nam ne razkrije opazne razlike (1. vrstica, 3. oz. 4. stolpec). Kot smo lahko pričakovali, naključne perturbacije nekoliko upočasnijo konvergenco (1. vrstica, 3. stolpec), vendar na dolgi rok se ni nobena različica izkazala za boljšo (1. vrstica, 4. stolpec). Medtem ko je videti, da perturbacija na originalen algoritem nima pravega učinka, pa je zelo zanimivo, da opazno izboljša delovanje preostalih treh različic. V teh primerih perturbacija ne samo, da nadomesti prehitro izgubo raznolikosti populacije, ki bi bila lahko posledica prehitre konvergence, temveč tudi izboljša splošen učinek. Videti je, da se spremenjeni postopki izbora in naključne perturbacije medsebojno podpirajo. Vendar nam primerjave postopkov Cr1, Cr2 in Cr3 s perturbacijo po 50.000 oz. 150.000 IKF kažejo, da na dolgi rok v vseh treh primerih originalna metoda nekoliko pridobi v primerjavi s precej slabšim delovanjem v začetni fazi teka. Tako verjetno obstaja nekaj možnosti za dodatno izboljšavo s pomočjo uravnoteženja dejavnikov, ki vplivajo na hitrost konvergence oz. hitrost spreminjanja raznolikosti populacije.

4.2 Podrobnejša analiza

Poglejmo si zdaj nekoliko bliže kriterij izbora Cr3, kombiniran s perturbacijo vektorjev, kar nam je po dosedanjih opažanjih na splošno prineslo najvidnejšo izboljšavo. Da bi dobili podrobnejšo sliko, smo izvedli enake primerjave kot prej, le da tokrat za vsako testno funkcijo posebej. Rezultate povzema tabela 2. Tabela prikazuje primerjave originalne metode z originalno metodo s perturbacijo in s postopkom izbora Cr3 s perturbacijo ali brez nje. Vrednosti v običajni pisavi pomenijo stanje po 50.000 IKF, medtem ko pomenijo tiste v krepki pisavi stanje po 150.000 IKF.

Tabela 2: Primerjava vplivov različnih modifikacij po posameznih testnih funkcijah.

Testna funkcija	**Modifikacija algoritma**		
	Original s perturbacijo	***Cr3***	***Cr3 s perturbacijo***
f_1 (Kvadratična f.)	34,72/65,28 **31,60/68,40**	70,83/29,17 **68,75/31,25**	80,21/19,79 **76,39/23,61**
f_2 (Schwefel 2.22)	33,45/66,55 **31,71/68,29**	70,73/29,27 **66,90/33,10**	74,22/25,78 **70,03/29,97**
f_3 (Schwefel 1.2)	51,04/48,26 **54,51/45,49**	75,35/23,96 **70,49/29,51**	77,08/22,57 **78,47/21,53**
f_4 (Schwefel 2.21)	49,48/48,08 **48,43/49,83**	63,07/34,49 **59,58/39,72**	83,97/12,80 **79,79/18,82**
f_5 (Posplošena Rosenbrockova f.)	49,83/50,17 **52,96/47,04**	58,19/41,81 **56,10/43,90**	77,00/23,00 **73,87/26,13**
f_6 (Stopničasta f.)	31,36/24,39 **29,97/9,76**	31,36/27,87 **18,12/26,48**	47,04/6,62 **35,19/3,48**
f_7 (Kvartična f. s šumom)	46,50/53,50 **49,30/50,70**	70,28/29,72 **67,83/32,17**	77,97/22,03 **77,62/22,38**
f_8 (Posplošena f. Schwefel 2.26)	57,14/42,86 **58,54/29,62**	49,83/50,17 **36,59/58,19**	82,58/17,42 **78,75/11,15**
f_9 (Posplošena Rastriginova f.)	50,69/48,96 **49,65/43,40**	66,67/33,33 **57,64/39,24**	80,90/19,10 **79,17/15,28**
f_{10} (Ackley)	48,26/50,69 **48,96/33,33**	60,07/39,24 **43,75/41,67**	81,60/17,36 **68,40/15,63**
f_{11} (Posplošena Griewankova f.)	32,17/61,19 **31,47/36,71**	63,99/29,72 **42,31/29,02**	73,08/20,28 **53,50/17,83**
f_{12} (Posplošena kazenska f. 1)	43,36/56,29 **36,01/45,80**	68,53/31,12 **52,45/33,22**	81,47/18,53 **65,38/20,63**
f_{13} (Posplošena kazenska f. 2)	45,10/54,90 **42,66/43,01**	62,59/37,06 **50,35/37,06**	82,17/17,48 **72,03/15,03**
f_{15} (Kowalik)	44,41/52,45 **45,80/46,50**	48,25/47,55 **50,70/45,10**	52,80/45,80 **49,65/45,80**

Prvo in najpomembnejše, kar opazimo je, da kaže modifikacija v kombinaciji s perturbacijo opazno in konsistentno boljše rezultate v vseh primerih razen pri Kowalikovi testni funkciji, kjer ni zaslediti opazne razlike. Spet vidimo, da sama perturbacija praviloma ne izboljša originalne metode. Izstopajoči izjemi pri tem sta stopničasta funkcija in funkcija Schwefel 2.26.

Schweflova funkcija je nekoliko problematična, ker je globalni minimum geometrično precej oddaljen od naslednjih nekaj najboljših lokalnih minimumov. Originalni postopek izkazuje precej dobro konvergenco na začetku, kasneje pa mu pri iskanju globalnega minimuma pomagajo naključne perturbacije. Brez njih se originalni algoritem zatakne v katerem od lokalnih minimumov (1. stolpec, funkcija Schwefel 2.26). Zanimivo je, da se izkaže pri tej funkciji modifikacija brez perturbacije mnogo slabše kot originalen postopek. Tako je verjetno zato, ker ta postopek izdatno zamenjuje člane polovice populacije, s čimer dodatno sili populacijo v enega od lokalnih minimumov. Modificirani postopek s perturbacijo se v tem primeru izkaže precej bolje.

Podobno velja za stopničasto funkcijo. Tudi ta pomeni za originalni postopek nekaj težav, saj je sestavljena iz številnih platojev in nezveznosti. Vse točke v majhni soseščini imajo enako vrednost kriterijske funkcije, kar postopku precej otežuje gibanje

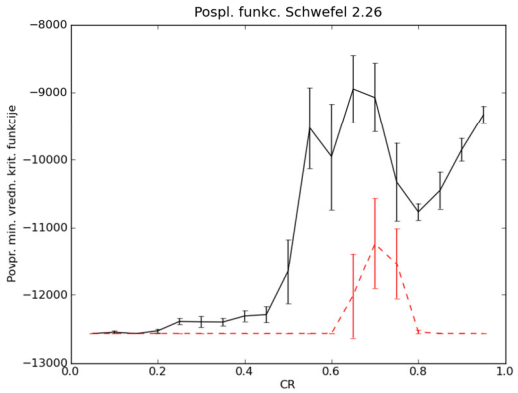
iz enega na drug plato. Zdi se, da perturbacije tu precej pomagajo.

4.3 Vpliv parametrov

Doslej se pri poskusih nismo osredotočali na dejansko izbiro krmilnih parametrov in velikosti populacije. Vrednosti smo izbirali popolnoma naključno v okviru predpisanih intervalov. V tem razdelku želimo raziskati, kako različne nastavitve parametrov vplivajo na delovanje algoritma s predlaganimi modifikacijami. Primerjali bomo originalen algoritem s tistim, ki uporablja postopek izbora Cr3 s perturbacijo. Čeprav se tu osredotočamo na eno samo testno funkcijo (posplošeno funkcijo Schwefel 2.26), moramo poudariti, da smo podobno obnašanje opazili tudi drugod.

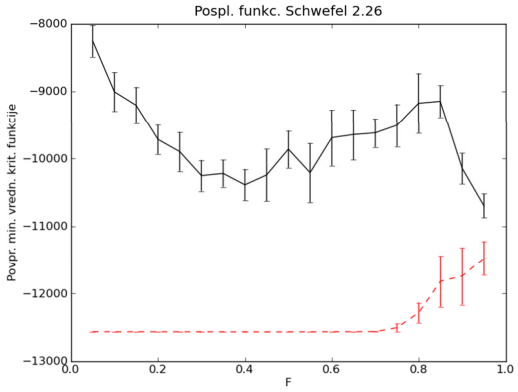
Za izhodišče smo izbrali vrednosti krmilnih parametrov, ki jih v literaturi najdemo najpogosteje, t.j. $F = 0{,}5$ in $CR = 0{,}9$. Eksperimentalno smo ugotovili, da dobimo pri teh vrednostih najboljše vrednosti kriterijske funkcije (ob predpostavljenih konstantnih 150.000 IKF) pri velikosti populacije $NP = 40$. Rezultate v tem razdelku smo dobili s spreminjanjem ene od omenjenih treh vrednosti, pri čemer smo preostali dve ohranili konstantni. Najboljše vrednosti kriterijskih funkcij so bile povprečene prek 25 neodvisnih tekov.

Slika 1 kaže, da originalnemu postopku nikjer ni uspelo doseči globalnega minimuma (ki leži pri –12569,5), razen pri najnižjih vrednostih *CR*. Zanimivo je, da z uporabo modifikacije algoritem DE najde minimum pri nižjih in višjih vrednostih *CR*, vendar ne tudi pri vrednostih okrog 0,7. Podobno obnašanje zasledimo na sliki 2, samo, da je tu izboljšanje nekoliko slabše le pri najvišjih vrednostih *F*.

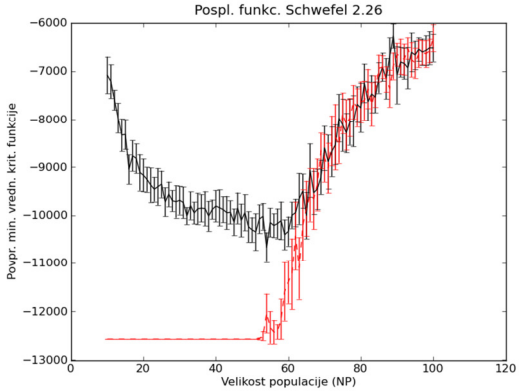


Slika 1: Vpliv krmilnega parametra *CR*. Neprekinjena črta pomeni rezultate, dobljene z originalnim algoritmom. Prekinjena črta prikazuje vrednosti dobljene s postopkom izbire Cr3 s perturbacijo.

Na sliki 3, ki prikazuje vpliv velikosti populacije, vidimo, da dobimo največjo izboljšavo pri manjših velikostih populacije. Velika populacija v DE navadno zagotavlja večjo verjetnost, da najdemo globalni minimum. Izvirni predlog za velikost populacije je bil $NP = 10D$ [16]. Kasneje so avtorji predlagali različne velikosti, vendar vse znatno večje od razsežnosti kriterijske funkcije *D*. S slike 3 lahko vidimo, da pri večjih populacijah naša modifikacija ne prinaša nikakršnih prednosti pred originalnim pristopom. To je nekako pričakovano, saj naj bi bila DE precej stabilna pri večjih populacijah. Težava je v tem, da nam stabilnost ne koristi preveč, če gre na račun nepraktično velikega števila IKF. Vidimo lahko, da je eden pomembnejših prispevkov našega pristopa v tem, da lahko uporabimo namesto ene velike več manjših vzporednih populacij. Tu obstaja potencialna možnost, da združimo sposobnost, da dejansko najdemo globalni minimum, in hitrost konvergence.



Slika 2: Vpliv krmilnega parametra *F*.



Slika 3: Vpliv velikosti populacije.

5 SKLEP

V prispevku smo raziskovali različne postopke izbora v algoritmu DE v kombinaciji z dodatno naključno perturbacijo parametrskih vektorjev. Prek eksperimentov z zbirko standardnih testnih funkcij smo ugotovili, da prinaša le ena od predlaganih treh modifikacij opazno boljše rezultate kot originalni algoritem. Nekakšno presenečenje je bilo, da sama

perturbacija ne izboljša originalnega algoritma, medtem ko izboljša delovanje vseh drugih.

Ko smo proučevali učinek primerjalne sheme Cr3 v kombinaciji z naključno perturbacijo, smo opazili znatno izboljšanje delovanja pri vseh večrazsežnostnih funkcijah. Opazili smo tudi, da je izboljšanje izrazitejše pri določenih vrednostih krmilnih parametrov in velikostih populacije: pri nižjih vrednostih *F* in *NP* ter pri nižjih kot tudi višjih vrednostih *CR*. Zlasti izjemno je bilo izboljšanje pri manjših velikostih populacije, kar bi bilo lahko uporabno pri implementaciji vzporednih algoritmov DE z več manjšimi populacijami.

Ena od prednosti pristopa, ki ga predlagamo v prispevku, je dejstvo, da se poseg v originalni algoritem ne vpleta v nobeno od drugih operacij. Zato ga je mogoče implementirati neodvisno in v kombinaciji s katerimkoli pristopom, predlaganim v literaturi.

Konec koncev je lepota originalnega algoritma DE prav v njegovi izjemni enostavnosti izvedbe. Z našo raziskavo se nismo želeli oddaljiti od te enostavnosti in pokazali smo, da je mogoče izboljšati delovanje algoritma samo z drugačnim postopkom izbora in dodatno naključno perturbacijo. Verjamemo, da je nadaljnje delo v tej smeri smiselno.

ZAHVALA

Raziskavo je omogočilo Ministrstvo za visoko šolstvo, znanost in tehnologijo Republike Slovenije v okviru programa P2-0246 – Algoritmi in optimizacijski postopki v telekomunikacijah.

LITERATURA

[1] R. Storn in K. Price, "Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces," J. Glob. Optim., zv. 11, str. 341–359, 1997.

[2] K. Price, "An introduction to differential evolution," v New ideas in optimization, D. Corne, M. Dorigo in F. Glover, edit. London (UK): McGraw-Hill Ltd., 1999, str. 79–108.

[3] J. Brest, S. Greiner, B. Bošković in M. Mernik, "Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems," IEEE Trans. Evol. Comput., zv. 10, str. 646–657, 2006.

[4] J. Liu in J. Lampinen, "A fuzzy differential evolution algorithm," Soft Comput., zv. 9, str. 448–462, 2005.

[5] D. Zaharie, "Control of population diversity and adaptation in differential evolution algorithms," v Zbornik 9th International Conference on Soft Computing, R. Matoušek, P. Ošmera, edit. Brno (Češka Republika): Mendel 2003, str. 41–46.

[6] J. Brest in M. Sepesy Maučec, "Population size reduction for the differential evolution algorithm," Appl. Intell., zv. 29, str. 228–247, 2008.

[7] J. Teo, "Exploring dynamic self-adaptive populations in differential evolution," Soft Comput., zv. 10, str. 673–686, 2006.

[8] H. Y. Fan in J. Lampinen, "A trigonometric mutation operation to differential evolution," J. Glob. Optim., zv. 27, str. 105–129, 2003.

[9] S. Das, A. Abraham, U. K. Chakraborty in A. Konar, "Differential evolution using a neighborhood-based mutation operator," IEEE Trans. Evol. Comput., zv. 13, str. 526–553, 2009.

[10] D. Zaharie, "Influence of crossover on the behavior of differential evolution algorithms," Appl. Soft Comput., vol. 9, pp. 1126–1138, 2009.

[11] Z. Yang, J. He in X. Yao, "Making a difference to differential evolution," v Advances in metaheuristics for hard optimization, Z. Michalewicz in P. Siarry, edit.: Springer, 2007, str. 415–432.

[12] J. Olenšek, Á. Bűrmen, J. Puhan in T. Tuma, "DESA: a new hybrid global optimization method and its application to analog integrated circuit sizing," J. Glob. Optim., zv. 44, str. 53–77, 2009.

[13] J. Brest, S. Greiner, B. Bošković, M. Mernik in V. Žumer, "Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems," IEEE Trans. Evol. Comput., zv. 10, str. 646–657, 2006.

[14] S. Das, A. Konar in U. K. Chakraborty, "Two improved differential evolution schemes for faster global search," v Zbornik GECCO, Washington D.C.: 2005, str. 991–998.

[15] X. Yao, Y. Liu in G. Lin, "Evolutionary programming made faster," IEEE Trans. Evol. Comput., zv. 3, str. 82–102, 1999.

[16] R. Storn, "On the usage of differential evolution for function optimization," v Biennial Conference of the North American Fuzzy Information Processing Society (NAFIPS), Berkeley: 1996, str. 519–523.

Iztok Fajfar je diplomiral (1991), magistriral (1994) in doktoriral (1997) s področja elektrotehnike na Fakulteti za elektrotehniko Univerze v Ljubljani. V letu 1991 je bil raziskovalec na Institutu Jožef Stefan v Ljubljani, nakar je konec leta zasedel raziskovalno mesto na Fakulteti za elektrotehniko Univerze v Ljubljani. Trenutno je na fakulteti zaposlen kot izredni profesor. Poučuje več uvodnih in nadaljevalnih predmetov s področja računalniškega programiranja. Pri razvoju programske opreme je sodeloval pri več industrijskih projektih. Njegovo področje raziskovanja vključuje načrtovanje in optimizacijo elektronskih vezij.

Janez Puhan je doktoriral leta 2000 s področja elektrotehnike na Univerzi v Ljubljani. Je docent na fakulteti za elektrotehniko. Njegovo področje raziskovanja obsega modeliranje, simulacijo in optimizacijske postopke pri računalniškem načrtovanju vezij.

Sašo Tomažič je redni profesor na Fakulteti za elektrotehniko Univerze v Ljubljani. Je predstojnik Laboratorija za komunikacijske naprave in Katedre za telekomunikacije. Njegovo delo obsega raziskave na področju digitalne obdelave signalov, varnosti v telekomunikacijah in elektronskem poslovanju ter informacijskih sistemov.

Árpád Bűrmen se je rodil leta 1976 v Murski Soboti. leta 2003 je na Univerzi v Ljubljani doktoriral s področja elektrotehnike. Trenutno je izredni profesor na Fakulteti za elektrotehniko Univerze v Ljubljani. Njegovo raziskovalno področje vključuje zvezno in dogodkovno simulacijo vezij in sistemov, optimizacijske postopke, njihovo konvergenco in uporabo, ter algoritme za vzporedno in porazdeljeno računanje. Je eden vodilnih razvijalcev simulatorja vezij SPICE OPUS. Objavil je preko 20 člankov v recenziranih revijah.