SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 3093

Optimizacija parametara stohastičkih algoritama uporabom metode podijeli i usporedi

Kruno Tomola Fabro

Zagreb, lipanj 2013.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

Sadržaj

1. Uvod

Evolucijsko računanje je područje računarske znanosti koje obuhvaća algoritme i metode modelirane na temelju pojava iz prirode. Koriste se za rješavanje teških problema za koje klasične matematičke metode nisu uspješne ili se uopće ne mogu primijeniti. Stohastički algoritmi iz područja Evolucijskog računanja se najčešće koriste za problem optimiranja. Performanse i uspješnost takvih algoritama uveliko ovise o problemu koji rješavaju i ulaznim parametrima to jest početnim postavkama. Na problem se u većini slučajeva ne može utjecati kako bi se poboljšalo ponašanje algoritma, ali ono na što se može utjecati su odabir algoritma i njegovi ulazni parametri.

Ulazni parametri se mogu podijeliti na statičke i dinamičke. Dinamički parametri se mijenjaju tijekom izvođenja algoritma pomoću povratnih veza ili neke druge logike koja direktno ovisi o trenutačnom izvođenju algoritma. Dok su statički svi ostali. Optimiranje parametara stohastičkih algoritama za jedan problem ili više njih je veliki problem s kojim se bave računalni znanstvenici od samih početaka Evolucijskog računanja. Zbog stohastičke prirode algoritama su se počele primjenjivati statističke metode preuzete iz drugih grana znanosti. Taj trend uvođenja statističkih metoda se i dalje događa.

Također se pokazalo potrebno rangirati algoritme po raznim kategorijama performansi za razne kategorije problema kako bi se moglo brže i efikasnije odrediti najbolji algoritam za neki problem.

U poglavlju 2. se daje kratki uvod u dizajn izvođenje pokusa. Nakon toga je u poglavlju 3. navedeno i predstavljeno par klasičnih metoda za optimizaciju parametara stohastičkih algoritama. Poglavlje 4. se bavi opisom i definiranjem metode podijeli i usporedi. U poglavlju 5. se prikazuju rezultati dvaju ispitivanja algoritama podijeli i usporedi. Te se zadnje poglavlje 6. bavi izvedbom algoritama podijeli i usporedi za sustav ECF.

2. Dizajn eksperimenata

Osnovni problem u području evolucijskih algoritama je usporedba dvaju algoritma A i B. Prilikom izvođenja eksperimenata se pretpostavlja da je ponašanje algoritama različito[1]. Eksperimentalne hipoteze su najčešće oblika *"dodatak X, koji ima algoritam A, a nema B, uzrokuje posljedicu Y"*. Takve hipoteze se obično ne testiraju direktno nego se provodi niz testiranja oba algoritma nad setom problema. Na temelju rezultata izvođenja se zaključuje koji je algoritam bolji. Također je uobičajeno da za sve probleme, postavke algoritama ostaju iste. Takvi eksperimenti se zovu *opservacijski eksperimenti.* Oni su bili dominantni do 2000. od kada su se počeli primjenjivati *manipulativni eksperimenti*. Kod njih se variraju razni faktori algoritama kako bi se pokazala veza između X i Y. Međutim, odabir variranih faktora iziskuje veliku količinu iskustva rada s algoritmima A i B.

Zajedničko za obje vrste eksperimenata je potreba za usporedbom dvaju algoritama. Kako bi se dobili valjane rezultate potrebno je postaviti optimalne postavke algoritma za zadani problem. Metoda podijeli i usporedi rješava upravo taj problem pronalaženja optimalnih postavki za zadani problem.

Iz područja znanosti o dizajnu eksperimenata i njihovom provođenju te iz kasnije predstavljenog novog eksperimentalizma su izvučene ideje i metode koje se više ili manje koriste u eksperimentima u području evolucijskih algoritama i koje se koriste u izvedbi metode podijeli i usporedi. One se opisuju u nastavku ovog poglavlja.

Klasična metoda usporedbe dvaju algoritma se temelji na usporedbi medijana rezultata optimiranja u obliku hipoteze. Time se dolazi do standardne situacije iz statistike, usporedba uzoraka iz dvije populacije. Jedan od klasičnih testova s kojim se postiže usporedba je *Parni t-test*.

U nastavku je prikazan postupak *Parni t-test*. Nakon kojeg se daje kratki uvod u novi eksperimentalizam s naglaskom na uočenu razinu signifikantnosti.

[1] npr. drugi ima poboljšani operator križanja

2.1. Parni t-test

Pod pretpostavkom da uzorci dolaze iz normalne distribucije se može primijeniti Parni t-test. j ta uparena razlika

$$x_j = y_{1_j} - y_{2_j} \quad j = 1, 2, ..., n$$

se koristi za definiranje test statistike (2.1)

$$t_0 = \frac{\bar{x}}{S_x/\sqrt{n}} \tag{2.1}$$

gdje $\bar{x} = \frac{1}{n}\sum_{j=1}^{n} x_j$ označava očekivanje uzoraka, a (2.2)

$$S_x = \sqrt{\sum_{j=1}^{n} \frac{(x_j - \bar{x})^2}{n-1}} \tag{2.2}$$

je standardna devijacija uzoraka razlike. Neka (2.3)

$$\theta = \mu_1 - \mu_2 \tag{2.3}$$

označava razliku u očekivanjima. Null hipoteza $H : \mu_1 = \mu_2$ ili $H : \theta = 0$, ne bi bila prihvaćena ako bi $t_0 > t_{\alpha,n-1}$, gdje je $t_{\alpha,n-1}$ kvantil studentove razdiobe saα razinom značajnosti i sa n stupnjeva slobode. Općenito razina signifikantnosti predstavlja pogrešku prve vrste gdje se nulta hipoteza odbacila iako je ona istinita, a stupnjevi slobode je broj uzoraka iz populacije.

U području evolucijskih algoritama se na temelju prihvaćanja ili odbijanja hipoteze H donosi zaključak koji je algoritam bolji. Gdje su skupovi uzoraka $\vec{y}_1$ i $\vec{y}_2$ rezultati optimiranja algoritama koji se uspoređuju.

Mana ovog testa je pretpostavka da uzorci dolaze iz normalne distribucije što najčešće nije zadovoljeno. Također broj uzoraka iz obadvije populacije mora biti jednak.

Primjer Setovi uzoraka $\vec{y}_1, \vec{y}_2$ su izvučeni iz normalnih distribucija s jednakom devijacijom $\sigma = 2$, ali s različitim očekivanjima $\mu_1 = 1 \quad \mu_2 = -1$. Broj uzoraka $n = 10$.

$$\vec{y}_1 = \{-1.2, 1.04, 0.525, -0.76, -0.5, 3.5, 2.1, 3.26, 2.56, 2.74\}$$

$$\vec{y}_2 = \{-3.37, 1.4, -1.52, 0.41, -3.4, -0.25, -2.4, 2.1, -0.16, 0.18\}$$

Testira se nulta hipoteza $H : \theta = 0$ gdje je θ (2.3), a alternativna hipoteza je $H : \theta \neq 0$ s razinom signifikantnosti $\alpha = 0.05$. Oduzimanjem $\vec{y}_2$ seta od $\vec{y}_1$ dobiva se $\vec{x}$

$$\vec{x} = \{2.17, -0.36, 2.045, -1.17, 2.9, 3.75, 4.5, 1.16, 2.72, 2.56\}$$

Očekivanje seta $\vec{x}$ je

$$\bar{x} = \frac{1}{n}\sum_{j=1}^{n} x_j = 2.0275$$

dok je standardna devijacija

$$S_x = \sqrt{\sum_{j=1}^{n} \frac{(x_j - \bar{x})^2}{n-1}} = 1.7417$$

Na temelju izračunatog očekivanja i standardne devijacije može se izračunati vrijednost t_0 koja iznosi

$$t_0 = \frac{\bar{x}}{S_x/\sqrt{n}} = 3.49$$

Zbog toga što se radi o obostranom testu nulta hipoteza se odbacuje ako vrijedi

$$|t_0| > t_{\alpha/2,n-1} \tag{2.4}$$

Vrijednost kvantila studentove razdiobe iznosi $t_{0.025,9} = 2.262$. Temeljem toga što vrijedi (2.4) odbacuje se nulta hipoteza i prihvaća se alternativna hipoteza $H : \theta \neq 0$ odnosno da očekivanja normalnih distribucija iz kojih su izvučeni setovi uzoraka $\vec{y}_1, \vec{y}_2$ se razlikuju.

2.2. Novi eksperimentalizam

Do 1980. u području evolucijskih algoritama usporedba više algoritama se većinom bazirala na usporedbi medijana. Do 2000. su se u to područje uvodile klasične statističke metode, a od 2000. su se počelo uvoditi metode iz novog eksperimentalizma.

Važni predstavnici novog eksperimentalizma su (Hacking, 1983) i (Mayo, 1996). Osnovna razlika u konceptima između klasičnog i novog eksperimentalizma je u različitim pristupima:

- nema eksperimenata bez teorije
- teorije bi trebale biti opovrgljive

Klasični pristup se temelji na Poopperevoj paradigmi, dok novi eksperimentalizam prati:

- eksperiment može imati svoj život, može doći prije teorije
- opovrgljivost bi trebala bit popraćena sa potvrdljivošću

Novi eksperimentalizam je utjecajan trend u modernoj filozofiji znanosti koji nudi statističke metode da se postave eksperimenti, provedu testiranja i načini kako da se uči iz proizlazećih pogrešaka i uspjeha eksperimenta (Bartz-Beielstein, 2006).

U novom eksperimentalizmu se stavlja naglasak na znanstvenu umjesto na statističku relevantnost. Sa svrhom objektivnog izvođenja eksperimenata. Ono traži sigurnu bazu u eksperimentu, a ne u teoriji i opservaciji. Važni predstavnici novog eksperimentalizma su

Hacking, Galison, Gooding, Mayo i Franklin. Deborah Mayo, čiji se rad nalazi u središtu znanosti i filozofije zaključivanja iz statistike, predlaže detaljnu metodologiju kako znanstvene tvrdnje validirati s eksperimentima. Za znanstvenu tvrdnju se može reći da ju podupiru eksperimenti ako prođe značajan test. Tvrdnja najvjerojatnije ne bi prošla značajan test da nije točna. Mayova metoda omogućava eksperimentatoru da uči iz pogrešaka. Jedan od važnijih pojmova koje je uveo Mayo je uočena razina signifikantnosti.

2.3. Uočena razina signifikantnosti

Veza između odbijanja null hipoteze H i vrijednosti razlike očekivanja je bitna za interpretaciju odbijanja H. Da bi se odbijanje H moglo interpretirati Mayo je uveo uočenu razinu signifikantnosti (Bartz-Beielstein, 2006).

$$\alpha\left(\overline{x},\theta\right)=\hat{\alpha}\left(\theta\right)=P\left(\bar{X}\geq\bar{x}|\theta\right) \tag{2.5}$$

$\hat{\alpha}\left(\theta\right)$ iz (2.5) je površina desno od uočene vrijednosti očekivanja $\bar{x}$ seta uzoraka $\vec{x}$ ispod krivulje funkcije gustoće vjerojatnosti od[2] $\bar{X}$. To bi na slici 2.1 bio broj puta da je očekivanje bootstrap seta bilo veće od očekivanja $\bar{x}$. $\hat{\alpha}\left(\theta\right)$ se da interpretirati kao vjerojatnost da bi se uočila vrijednost jednaka ili veća od $\bar{x}$ ako bi vrijedilo $\theta=\theta'$. Odbijanje H je dobar pokazatelj da je $\theta>\theta'$ ako je vrijednost $\hat{\alpha}\left(\theta\right)$ mala, a ako je vrijednost $\hat{\alpha}\left(\theta\right)$ velika to onda ukazuje da je vrlo vjerojatno $\theta<\theta'$. Valja naglasiti da $\hat{\alpha}\left(0\right)$ predstavlja vjerojatnost pogreške prve vrste.

Uočena razina signifikantnosti prati princip novog eksperimentalizma da se treba učiti iz eksperimenata, uspješnih i neuspješnih.

2.4. Bootstrap

Mayo je uveo uočenu razinu signifikantnosti pod pretpostavkama da:

1. uzorci prate normalnu distribuciju
2. varijanca σ^2je poznata

U stvarnim situacijama ove pretpostavke ne vrijede. Uzorci testiranja nisu normalno distribuirani. To je očito već sa samom činjenicom da je puno vrijednosti lošije od optimuma ali nijedna nije bolja od optimuma što će za posljedicu imati deformaciju raspodjele na jednom njenom kraju. (Bartz-Beielstein, 2006)je predložio bootstrap pristup da se izbjegnu pretpostavke.

[2] $\bar{X}$ se ponaša kao slučajna varijabla

Bootstrap je metoda za procjenu distribucije statistike ili obilježja distribucije, poput očekivanja ili devijacije. Bootstrap metoda tretira uzorke, za slučaj usporedbe dvaju algoritama je to dobrota svakog pojedinačnog izvođenja algoritma, kao populaciju te se može primijeniti ako je distribucija uzoraka nepoznata (Efron i Tibshirani, 1993). Bootstrap zahtijeva reprezentativni uzorak populacije. U današnje vrijeme bootstrap se smatra standardnom metodom u statistici (Mammen i Nandi, 2012). Uspješno je primijenjen za rješavanje problema koji su previše složeni za klasične statističke tehnike i u situacijama gdje klasične statističke tehnike nisu valjane (Zoubir i Boashash, 1998). Ideja iza bootstrap je slična metodi koja se često primjenjuje u praksi. Eksperimenti se ponavljaju da bi se poboljšala procjena nepoznatog parametra. Bootstrap je računski zahtjevna tehnika. Neka je $\hat{\theta}$ procjena nepoznatog parametra θ do koje se došlo izračunavanjem statistike S od n uzoraka $y_1, y_2, ..., y_n$:

$$\hat{\theta} = S(y_1, y_2, ..., y_n) \tag{2.6}$$

Uzorkovanjem sa zamjenom, n_b bootstrap uzoraka se može dobiti. Bootstrap replike $\hat{\theta}^{*b}$

$$\hat{\theta}^{*b} = s(y^{*b}), b = 1, 2, ..., n_b \tag{2.7}$$

nude procjenu distribucije $\hat{\theta}$. Generična bootstrap procedura je

1. Izračunati $\hat{\theta}$ iz reprezentativnog uzorka $y = (y_1, y_2, ..., y_n)$.
2. Za generiranje bootstrap seta uzoraka $y^{*b} = (y_1^{*b}, y_2^{*b}, ..., y_n^{*b})$ uzorkovati sa zamjenom iz originalnog uzorka.
3. Upotrijebom bootstrap seta y^{*b} treba izračunti $\hat{\theta}^{*b}$.
4. Ponavljati korake 2 i 3 n_b puta.
5. Upotrijebiti procjenu distribucije $\hat{\theta}$ da se dobije željeni parametar.

Bootstrap metoda je pogodna za izvedbu na računalima.

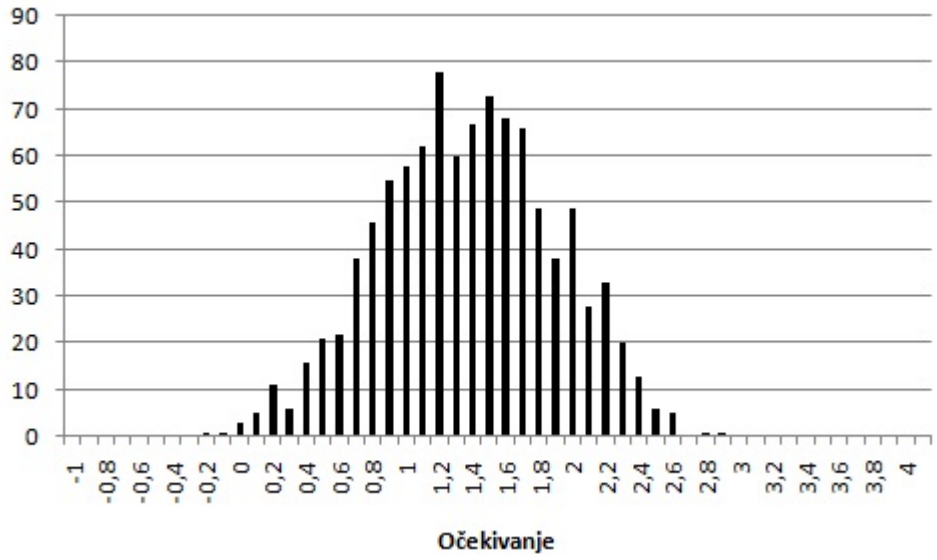
Ima više različitih metoda za smanjivanje varijance kod bootstrap od kojih su neke:

- Antihetic variate
- Zajednički slučajni brojevi[3]
- Kontrolne varijacije[4]

Od njih je predstavljena prva metoda koja se koristi kasnije u algoritmima podijeli i usporedi.

[3] *eng. Common random numbers*

[4] *eng. Control variates*



Slika 2.1: Bootstrap histogram iz primjera

Primjer Iz seta uzoraka $\vec{y}$

$$\vec{y} = \{-1.2, 1.04, 0.525, -0.76, -0.5, 3.5, 2.1, 3.26, 2.56, 2.74\}$$

izvučenih iz normalne distribucije sa standardnom devijacijom $\sigma = 2$ i očekivanjem $\mu = 1$ se želi dobiti distribucija očekivanja. $n_b = 1000$ bootstrap uzoraka se izrađuje.

1. Izračunato očekivanje $\bar{y}$ iz seta $\vec{y}$ iznosi

$$\bar{y} = \frac{1}{n}\sum_{j=1}^{n} y_j = 1.3265$$

2. Iz seta $\vec{y}$ se generira novi set uzoraka tako da je novi set iste kardinalnosti kao $\vec{y}$ dok se njegovi članovi nasumično odabrani iz seta $\vec{y}$.

$$npr. \qquad \vec{y}^{*b} = \{1.04, -0.76, 0.525, -0.76, -0.5, 3.5, 2.1, 2.74, 3.26, 3.26\}$$

3. Za bootstrap set $\vec{y}^{*b}$ se izračunava očekivanje $\bar{y}^{*b}$.

4. Koraci 3. i 2. se ponavljaju n_b puta te se pamte vrijednosti očekivanja $\bar{y}^{*b}$.

5. Od dobivenih n_b očekivanja i $\bar{y}$ se izrađuje histogram distribucije očekivanja vidljiv na slici 2.1.

Na slici 2.1 je prikazana distribucija očekivanja seta $\vec{y}$ nastala od očekivanja bootstrap setova podijeljenih u intervale širine 0.1 .

2.5. Metoda Antihetic variates

Antihetic variates je metoda smanjivanja varijance prilikom provođenja bootstrap metode. Klasična izvedba ove metode je da se sortira uzorak, dobiven testiranjem algoritma te da se prilikom generiranja bootstrap seta kod bootstrap metode u koraku 2. kada se nasumično odabere i ti član se također uzima i $n-i$ ti član, gdje je n veličina populacije, uzorka. Međutim odabrana je malo drugačija naprednija metoda predloženu u (Craiu, 2001).

Predložena metoda se sastoji od uzorkovanja dva dodatna člana uzorka na temelju odabira jednog. Ideja je da se konstruiraju matrice koje opisuju takve kombinacije uzorkovanja koje smanjuju varijancu procjene parametra. Način na koji to postižu, iako se napominje da nije optimalan ali je svejedno bolji od klasične metode, je pomoću ciklusa s kojima se grupiraju ekstremi uzorka $y_1 \leq y_2 \leq ... \leq y_n$. Gdje je prvi ciklus $1, n, n-1$, drugi $2, 3, n-2$ i tako dalje dok ne ostane manje od tri člana koji nisu uključeni u ciklus. Ti preostali članovi ostaju nepromijenjeni. Od tih ciklusa se rade tri matrice. Prva matrica je simetrična u smislu da u oba retka pod stupcem i piše i. Za $n = 7$ matrica je (2.8).

$$\pi_1 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{pmatrix} \tag{2.8}$$

Druga matrica se konstruira koristeći cikluse i redoslijed $1, 2, 3, 1, 2, 3, 1...$ Odnosno ako imamo ciklus $1, n, n-1$ onda na mjesto prvog člana (1) se stavlja drugi član (n), zatim na mjesto trećeg člana $(n-1)$ se stavlja prvi član (1) te na mjesto drugog člana (n) se stavlja treći član $(n-1)$. Postupak se ponavlja za sve cikluse. Za $n = 7$ matrica je (2.9).

$$\pi_2 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 7 & 3 & 5 & 4 & 2 & 1 & 6 \end{pmatrix} \tag{2.9}$$

Treća matrica se konstruira koristeći cikluse i redoslijed $1, 3, 2, 1, 3, 2, 1...$ Odnosno ako imamo ciklus $1, n, n-1$ onda na mjesto prvog člana (1) se stavlja treći član $(n-1)$, zatim na mjesto drugog člana (n) se stavlja prvi član (1) te na mjesto trećeg člana $(n-1)$ se stavlja drugi član (n). Postupak se ponavlja za sve cikluse. Za $n = 7$ matrica je (2.10).

$$\pi_3 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 6 & 5 & 2 & 4 & 3 & 7 & 1 \end{pmatrix} \tag{2.10}$$

Tako konstruirane matrice se koriste prilikom uzorkovanja uzorka tako da se slučajnim odabirom odabere broj $i \in [1, n]$ te se pomoću njega pristupi u druge retke sve tri matrice. Dobiveni brojevi su indeksi članova iz uzorka koji su odabrani.

3. Determinističke optimizacijske metode

Optimizacijske metode se u grubo mogu podijeliti u dvije kategorije:

- stohastički
- deterministički

U prvu skupinu spadaju Evolucijski algoritmi. Za optimizaciju parametara stohastičkih algoritama, nije povoljno koristiti drugi stohastički algoritam jer se onda opet dolazi do problema optimiranja parametara. Zato su determinističke metode primjenjivije. Rad se od sad pa nadalje bavi samo determinističkim metodama. Neke od njih su:

- Taguchi metoda (Trosset, 1997)
- TOPSIS (Yoon i Hwang, 1995)
- Nelder-Mead simplex (Lagarias et al., 1998)
- Varijabilna metrika[1] (Davidon, 1991)
- Metoda podijeli i usporedi

od njih su prve dvije pobliže opisane u ovom poglavlju dok zadnja metoda podijeli i usporedi ima svoje poglavlje te je ujedno i glavna tema rada.

3.1. Taguchi metoda

Doprinos Genchi Taguchi-a u području inženjerstva kvalitete je širokog utjecaja i premašuje područje ovog završnog rada. Zato je naglasak na njegovoj metodi optimiziranja dizajna proizvoda koja se uvela u područje Evolucijskog računarstva.

Taguchi je zamislio tri koraka za optimizaciju dizajna:

1. Odrediti moguću regiju parametara za optimizacijski problem.
2. Optimiziranje ciljne funkcije koja operira nad pojmom kvalitete.

[1] *eng. Variable metric*

3. Fino tuniranje optimalnog dizajna dobivenog u drugom koraku.

Taguchi je imao veoma preciznu ideju o tome kakva je ciljna funkcija za optimizaciju (Trosset, 1997). Možda njegov najbitniji doprinos inženjerstvu kvalitete jest njegova opservacija da bi dizajn proizvoda trebao biti takav da su njegove performanse neosjetljive na varijacije varijabla koje su izvan utjecaja dizajnera. Zbog toga se Taguchi naširoko smatra ocem robusnog dizajna.

Taguchi metoda razlikuje dva tipa ulaza u sustav:

1. kontrolni faktori

2. šum

Kontrolni faktori su ulazi koji se mogu lako kontrolirati i mijenjati prema volji dizajnera, to su optimizacijske varijable x. Dok je šum skupina varijabli ξ koje je teško ili skupo kontrolirati. ξ su varijable na čije varijacije bi proizvod trebao biti neosjetljiv.

Da bi dobili ciljnu funkciju neka y označava kvalitetu karakteristike koju promatramo. Ta karakteristika može biti jedna od tri tipa za koje je Taguchi definirao mjeru očekivanja korijena devijacije *eng. Mean Squared Deviation* (MSD). Neka $y_1, y_2, ..., y_n$ označavaju uzorke dobivene variranjem ξ za fiksirani x. Ako se kvaliteta mjeri koliko je y blizu ciljane vrijednosti m, onda se primjenjuje (3.1)

$$MSD = \frac{1}{n}\sum_{i=1}^{n}(y_i - m)^2 \tag{3.1}$$

ako je kvaliteta mali y, onda se primjenjuje (3.2)

$$MSD = \frac{1}{n}\sum_{i=1}^{n} y_i^2, \tag{3.2}$$

a ako je kvaliteta veliki y, onda se primjenjuje (3.3)

$$MSD = \frac{1}{n}\sum_{i=1}^{n} y_i^{-2} \tag{3.3}$$

Ciljna funkcija koju se treba minimizirati variranjem x je onda omjer signala po šumu *eng. Signal to Noise Ratio*(SNR), $-10\log_{10}(MSD)$. Naglasak SNR kao ciljne funkcije je glavno obilježje Taguchi metode.

Pristup optimiziranju SNR je inspiriran principima klasičnog dizajna eksperimenata. Kontrolne varijable x se sistematički variraju prema ortogonalnom polju, kontrolno ili unutrašnje polje. Za svaku vrijednost x se ξ sistematički variraju prema drugom ortogonalnom polju, šum ili vanjsko polje. Za svaku vrijednost x, podaci dobiveni repliciranjem pokusa za različite ξ se koriste za procjenu SNR. Tako se dobiva polje procijenjenih SNR koje

se onda analizira sa standardnim analizama varijance da se odrede vrijednosti x za koje je ponašanje dizajna robusno.

Ortogonalna polja koja se koriste u Taguchi metodi su dizajnirana tako da dovedu što više do izražaja među interakcije između parametara sa što manje testiranja. Jedna od web stranica gdje se mogu naći definicije i upute kako se one koriste je wiki stranica od University of Michigan[2].

Valja naglasiti da Taguchi metoda specificira vrijednosti x za koje će se ciljna funkcija testirati prije nego što se provedu bilo koja testiranja. Ovakav pristup narušava temeljno načelo numeričkog optimiranja da bi se trebalo izbjegavati raditi previše posla dok se ne bude blizu rješenja. No u njegovu obranu bi trebalo biti rečeno da su ga je najviše brinule situacije kada sekvencijalno testiranje nije moguće jer je za potrebe testiranja potrebno dizajnirati, i posve posvetiti, kompletnu tvornicu.

Primjer Neka se optimiraju tri kontrolna faktora s dvije razine ispitivanja te dva šuma s dvije razine ispitivanja. U oba slučajeva se koriste ortogonalna polja tipa L4. Kontrolno polje je u tablici 3.1 dok je šum polje u tablici 3.2.

Tablica 3.1: Kontrolno ortogonalno polje

Eksperiment	Faktor 1	Faktor 2	Faktor 3
1.	10	0	-20
2.	10	0.5	-10
3.	40	0	-10
4.	40	0.5	-20

Tablica 3.2: Šum ortogonalno polje

Eksperiment	Šum 1	Šum 2
a	0.1	0
b	0.1	25
c	0.7	0
d	0.7	25

Ciljna funkcija ima samo jednu kvalitetu y za koju se želi da bude što manje vrijednosti. Zbog toga se koristi formula 3.2.

Za svaki zapis kombinacija vrijednosti kontrolnih faktora u kontrolnom polju se provodi pokus sa svakim zapisom kombinacija vrijednosti šuma u šum polju. Kvaliteta y dobivena

[2]controls.engin.umich.edu/wiki/index.php/Design_of_experiments_via_taguchi_methods:_orthogonal_arrays

tim pokusima je u tablici 3.3.

Tablica 3.3: Kvaliteta po pokusima

Kombinacije	Šum	a	b	c	d
Kontrolne					
1.		0.1	0.3	0.15	0.4
2.		0.15	0.1	0.5	0.05
3.		0.4	0.2	0.18	0.2
4.		0.1	0.2	0.3	0.05

Izračunati MSD i SNR se nalaze u tablici 3.4. Sada bi se korištenjem podataka iz tablice 3.4 radila jedna od standardnih analiza varijance, ali to prelazi područje ovog završnog rada.

Tablica 3.4: MSD, SNR tablica

Kontrolni eksperiment	MSD	SNR
1.	0.07	11.55
2.	0.071	11.49
3.	0.068	11.67
4.	0.036	14.44

3.2. TOPSIS

TOPSIS označava prevedeno tehnika preferiranja sličnosti do idealnog riješenja[3]. TOPSIS su predstavili (Yoon i Hwang, 1995). TOPSIS je metoda za optimizaciju po više kriterija koja identificira najbolju konfiguraciju od konačnog broja alternativa (konfiguracija parametara). Osnovna ideja je da izabrana alternativa treba imati najmanju udaljenost od positivnog *ideal* riješenja i najveću udaljenost od negativnog *ideal* riješenja (*nadir*). Prednost TOPSIS metode je što zahtjeva limitirani subjektivni ulaz, samo težine. TOPSIS metoda je uspiješno primjenjena na široki raspon područja od financijskog sektora, robotike, evolucijskog računarstva, ...

TOPSIS metoda se može opisati sa nizom koraka.

1. Dobiti podatke performanse za n alternativa po k kriterija. Sirova mjerenja se obično standardiziraju, pretvarajući sirova mjerenja x_{ij} u standardizirane mjere s_{ij}.

[3] *eng. technique for preference by similarity to the ideal solution*

2. Razviti set težinskih faktora ω_k, za svaki kriterij. Osnova težina može biti bilo što, ali je uobičajeno *ad hoc* određivanje. Skala nije problem ako je provedena standardizacija u 1. koraku.

3. Identificirati *ideal* alternativu s^+(extremne perfomanse za svaki kriterij).

4. Identificirati *nadir* alternativu s^-(suprotno extremne performanse za svaki kriterij).

5. Razviti mjeru udaljenosti za performanse alternativa od *ideal*(D^+) i *nadir* (D^-).

6. Za svaku alternativu, odrediti omjer R koji je jednak udaljenosti od *nadir* podijeljeno sa sumom udaljenosti od *nadir* i *ideal*.

$$R = \frac{D^-}{D^- + D^+}$$

7. Rangiraj alternative maksimizirajući omjer u 6. koraku.

Razne mjere udaljenosti se mogu primjeniti. Tradicionalni TOPSIS koristi euklidsku udaljenost L_1 dok se još koriste i L_2 i L_∞ norme.

Primjer Određuje se najbolja alternativa od njih 4 po 2 kriterija. Zbog apstraktnosti metode nije niti potrebno znati koje su alternative za ovaj primjer. Zna se da je za kriterij 1 bitno da je što manji, a za kriterij 2 da je što veći te također da je kriterij 1 važniji od kriterija 2.

1. Standardizirane performanse alternativa se nalaze u tablici 3.5.

Tablica 3.5: Standardizirane performanse alternativa

Alternativa	Kriterij 1	Kriterij 2
1.	0.3	0.1
2.	0.5	0.7
3.	0.1	0.05
4.	0.2	0.8

2. Težine za svaki kriterij su određene prema važnosti svakog kriterija i mogu se naći u tablici 3.6.Težine se koriste tako da se pomnože s performansom svake alternative.

Tablica 3.6: Težine kriterija

	Kriterij 1	Kriterij 2
Težine	1	0.7

3. Performansa ideal alternative je napravljena na temelju zahtijeva za svaki kriterij. Njene vrijednosti su u tablici 3.7 .

Tablica 3.7: Performanse ideal i nadir alternativa

	Kriterij 1	Kriterij 2
Ideal	0	1
Nadir	1	0

4. Nadir alternativa ima suprotne performanse od ideal alternative. Njene vrijednosti su u tablici 3.7.

5. Koristi se $L2$ norma kao mjera udaljenosti performansa alternativa od ideal i nadir. Udaljenosti su u tablici 3.8. npr kompletan izračuna za udaljenost 1. alternative od ideal je

$$D_1^+ = \sqrt{(w_1 * a_{1,1} - i_1)^2 + (w_2 * a_{1,2} - i_2)^2} = 0.977$$

gdje su w_1, w_2 težine, i_1, i_2 performansa ideala za svaki kriterij i $a_{1,1}, a_{1,2}$ performanse alternative 1. za svaki kriterij.

Tablica 3.8: Udaljenosti performansa alternativa od ideal i nadir

Alternative	Ideal D^+	Nadir D^-
1.	0.977	0.7
2.	0.714	0.7
3.	0.97	0.9
4.	0.48	0.976

6. Na temelju udaljenosti alternativa od nadir i ideal se izračunavaju omjeri R. Nalaze se u tablici 3.9 .

Tablica 3.9: Omjeri R

Alternative	R
1.	0.42
2.	0.49
3.	0.48
4.	0.67

7. Rangiranjem alternativa maksimizirajući R se zaključuje da je 4. alternativa bolja od ostalih razmatranih alternativa.

4. Metoda podijeli i usporedi

Metoda podijeli i usporedi je deterministička metoda namijenjena za optimizaciju parametara stohastičkih algoritama. Ona je razvijena i ostvarena na poticaj moga mentora Marina Goluba za predmet Projekt s ciljem daljnjeg razvoja tijekom nadolazećih godina. Svojstva koja imaju algoritmi razvijeni na temelju metode podijeli i usporedi su:

- *stabilnost* - sposobnost da se ponašanje algoritma, za određene vrijednosti njegovih parametra, ne mijenja s promjenom problema.
- *objektivnost* - ne zahtijeva nikakve pretpostavke ili bilo koje druge dodatne informacije o problemu koji se optimizira.
- *automatiziranost* - moguće je efikasno izvesti cijeli postupak optimiranja na računalu.

Ono što nemaju algoritmi razvijeni na temelju metode podijeli i usporedi je ponovljivost optimiranja. Odnosno više izvođenja algoritma s istim postavkama nad istim problemom neće nužno vratiti iste rezultate, zbog stohastičke prirode problema. Ali će vraćati slične ili jednako dobra rješenja.

Za razliku od TOPSIS i Taguchi metoda koje nespecificiraju koje alternative, vrijednosti kontrolnih varijabla treba ispitati, metoda podijeli i usporedi daje okvir unutar kojeg se trebaju odabrati alternative.

U nastavku je prvo opisana metoda podijeli i usporedi nakon čega se utvrđuje jedan od mogućih načina njene izvedbe.

4.1. Metoda podijeli i usporedi

U metodi se razmatra područje, dio ili cijela domena ulaznih parametara stohastičkog algoritma, kao osnovni entitet obrade umjesto točke kao u dosadašnjim optimizacijskim metodama. Jedan od benefita koji se dobiva takvim pogledom je smanjenje broja potrebnih testiranja, kako i zašto je objašnjeno u poglavlju 4.2. Problem kojeg optimiziraju stohastički algoritmi je zajednički svima te se nadalje on podrazumijeva. Algoritam se izvodi u iteracijama, a broj tih iteracija se naziva *dubina* optimiranja. Iteracija počinje od početnog skupa gdje je član skupa definiran algoritam. Definiran algoritam čine par stohastički algoritam i područje nad kojim se optimizira. Koraci metode su:

1. Iz svakog definiranog algoritma, roditelja, početnog skupa iteracije stvoriti nove definirane algoritme, djecu, koja nasljeđuju algoritam koji se optimizira od roditelja te svako dijete nasljeđuje dio roditeljskog područja. Pri tome se može ali ne mora pratiti pravilo da je svaki dio područja dodijeljen točno samo jednom djetetu. Novonastali definirani algoritmi čine među skup.

2. Testiranje svakog definiranog algoritma iz među skupa.

3. Uspoređivanje svakog člana iz među skupa s svim ostalim članovima iz među skupa, gdje se za svaki član određuje broj puta kada je bio strogo bolji od drugog člana. Tim se postupkom dolazi do dobrote definiranog algoritma.

4. Stvaranje završnog skupa kojeg čine definirani algoritmi iz među skupa s najvećom odnosno najmanjom dobrotom ovisno je li se radi o problemu maksimizacije ili minimizacije.

Završni skup se može dalje dati kao ulaz sljedećoj iteraciji ili vratiti kao rezultat optimiranja.

U ovisnosti o tome što se daje kao početni skup prve iteracije mogu se dobiti sljedeći tipovi optimiranja:

- Početni definirani algoritmi imaju različite stohastičke algoritme. Time se uz optimizaciju može dobiti dobar pokazatelj koji stohastički algoritam je najbolji za promatrani problem.
- Početni definirani algoritmi imaju isti bazni stohastički algoritam, ali se razlikuju u nekim dijelovima[1] i mogu se razlikovati u područjima optimiranja parametara. Uz optimizaciju se može dobiti dobar pokazatelj na učinke različitih dijelova stohastičkog algoritma na njegovo ponašanje.
- Početni definirani algoritmi imaju isti stohastički algoritam, ali se razlikuju u području optimiranja parametara. Time se dobiva optimiranje nad selektivnije odabranim područjima parametara. Ova vrsta će biti potrebna u slučaju diskretnih ulaznih parametara stohastičkog algoritma kojeg se optimizira.
- Početni skup se sastoji od samo jednog definiranog algoritma. To je najosnovnija vrsta optimiranja parametara jednog algoritma nad odabranim područjem.

Zbog svoje važnosti će naglasak u radu biti na zadnjoj vrsti optimiranja jer je ona baza za ostale. U nastavku se izvodi postupak 3. koraka metode podijeli i usporedi iz čije izvedbe proizlaze razni zahtjevi za druge korake metode.

[1] npr. drugačiji operatori križanja kod genetskih algoritama

4.2. Izvod usporedbe

Stohastički algoritam koji prima parametre $\vec{X} = x_1, x_2, ..., x_N; x_i \in R, i = 1, 2, ..., N$ se može prikazati kao slučajna varijabla $S_{\vec{X}}$ s nepoznatom razdiobom. Neka funkcija $m(\vec{X})$ vraća očekivanje slučajne varijable $S_{\vec{X}}$. Dobrota područja parametara za koje vrijedi $\vec{X} \in [\vec{X}_{MIN}, \vec{X}_{MAX}]$ je (4.1).

$$d = \frac{1}{\prod_{i=1}^{N}(x_{MAX_i} - x_{MIN_i})} \int_{\vec{X}_{MIN}}^{\vec{X}_{MAX}} m(\vec{X}) d\vec{X} \tag{4.1}$$

Gdje se $\int_{\vec{X}_{MIN}}^{\vec{X}_{MAX}} m(\vec{X}) d\vec{X}$ može prikazati kao Riemann integral gdje broj pravokutnika teži u beskonačnost (4.2).

$$\begin{aligned}
&\int_{\vec{X}_{MIN}}^{\vec{X}_{MAX}} m(\vec{X}) d\vec{X} = \lim_{n\to\infty} \frac{\prod_{i=1}^{N}(x_{MAX_i} - x_{MIN_i})}{n^N} \sum_{\vec{X}\in B} m(\vec{X}) \\
&B = x_1, x_2, ..., x_N; x_j \in (x_{MIN_j} + 1 * \frac{x_{MAX_j} - x_{MIN_j}}{n}, \\
&\qquad\qquad x_{MIN_j} + 2 * \frac{x_{MAX_j} - x_{MIN_j}}{n}, \\
&\qquad\qquad ..., \\
&\qquad\qquad x_{MIN_j} + n * \frac{x_{MAX_j} - x_{MIN_j}}{n}) \\
&\qquad\qquad j = 1, ..N
\end{aligned} \tag{4.2}$$

Neka je $s_{\vec{x}}$ vrijednost koju je poprimila slučajna varijabla $S_{\vec{X}}$ tada se $m(\vec{X})$ može izraziti kao (4.3).

$$m(\vec{X}) = \lim_{m\to\infty} \frac{1}{m} \sum_{j=1}^{m} s_{\vec{X}j} \tag{4.3}$$

Sada se uvođenjem (4.2) i (4.3) u (4.1) dobiva (4.5).

$$\mu = \frac{1}{\prod_{i=1}^{N}(x_{MAX_i} - x_{MIN_i})} \lim_{n\to\infty} \frac{\prod_{i=1}^{N}(x_{MAX_i} - x_{MIN_i})}{n^N} \sum_{\vec{X}\in B} \lim_{m\to\infty} \frac{1}{m} \sum_{j=1}^{m} s_{\vec{X}j} \tag{4.4}$$

$$\mu = \lim_{\substack{n\to\infty \\ m\to\infty}} \frac{1}{m * n^N} \sum_{\vec{X}\in B} \sum_{j=1}^{m} s_{\vec{X}j} \tag{4.5}$$

Neka je razlika dobrota između dva definirana algoritma (4.6) .

$$\theta = \mu_1 - \mu_2 = \lim_{\substack{n\to\infty \\ m\to\infty}} \frac{1}{m * n^N} \sum_{\substack{\vec{X}_1\in B_1 \\ \vec{X}_2\in B_2}} \sum_{j=1}^{m} (s_{\vec{X}_1 j} - s_{\vec{X}_2 j}) \tag{4.6}$$

Supstitucijom $v_{\vec{X}_1\vec{X}_{2j}} = s_{\vec{X}_{1j}} - s_{\vec{X}_{2j}}$, u (4.6), gdje se $v_{\vec{X}_1\vec{X}_{2j}}$ može promatrati kao vrijednost koju je poprimila slučajna varijabla $V_{\vec{X}_1\vec{X}_2}$ dolazi se do oblika (4.7).

$$\theta = \mu_1 - \mu_2 = \lim_{\substack{n\to\infty \\ m\to\infty}} \frac{1}{m * n^N} \sum_{\substack{\vec{X}_1 \in B_1 \\ \vec{X}_2 \in B_2}} \sum_{j=1}^{m} v_{\vec{X}_1\vec{X}_{2j}} \tag{4.7}$$

Na temelju vrijednosti θ se može zaključiti sljedeće :

- $\theta < 0$ prvi definiran algoritam ima manju dobrotu od drugog
- $\theta = 0$ definirani algoritmi imaju jednaku dobrotu
- $\theta > 0$ drugi definirani algoritam ima manju dobrotu od prvog

Iz jednadžbe (4.7) se može zanemariti učinak člana $\frac{1}{m*n^N}$ jer on nema utjecaja na ishod gornjih zaključaka. Tako se dobiva (4.8).

$$\theta = \mu_1 - \mu_2 = \lim_{\substack{n\to\infty \\ m\to\infty}} \sum_{\substack{\vec{X}_1 \in B_1 \\ \vec{X}_2 \in B_2}} \sum_{j=1}^{m} v_{\vec{X}_1\vec{X}_{2j}} \tag{4.8}$$

Sada se iskorištava tumačenje uočene razine signifikantnosti iz poglavlja 2.3 za hipotezu $H : \theta = 0$. Tada je $\hat{\alpha}(\theta)$ vjerojatnost pogreške prve vrste. Vrijednost $\hat{\alpha}(0)$ se koristi kao test hipoteze H, gdje α predstavlja minimalnu razinu signifikantnosti:

- $\hat{\alpha}(0) < \alpha$ odbacuje se hipoteza H te se na temelju male vrijednosti $\hat{\alpha}(\theta)$ zaključuje da je $\theta > 0$.
- $\hat{\alpha}(0) > 1 - \alpha$ odbacuje se hipoteza H te se na temelju velike vrijednosti $\hat{\alpha}(\theta)$ zaključuje da je $\theta < 0$.
- $\alpha \leq \hat{\alpha}(0) \leq 1 - \alpha$ prihvaća se hipoteza H da vrijedi $\theta = 0$.

Vrijednost $\hat{\alpha}(0)$ za konačan broj uzoraka slučajne varijable $V_{\vec{X}_1\vec{X}_2}$, se može izračunati pomoću bootstrap metode iz poglavlja 2.4. I to sljedećom procedurom za skup uzoraka $\vec{v} = v_1, v_2, ..., v_n$:

1. Odrediti $\bar{v} = \frac{1}{n}\sum_{j=1}^{n} y_j$
2. Odrediti $\vec{v}_0 = \vec{v} - \bar{v}$
3. Generiraj n_b bootstrap setova uzoraka $\vec{v}_i, i = 1, ..., n_b$ iz $\vec{v}_0$ sa Antihetic variates metodom.
4. Odrediti n_b očekivanja $\bar{v}_i$ iz setova uzoraka $\vec{v}_i$
5. Odrediti m, to jest broj puta da je $\bar{v}_i > \bar{v}$
6. Odrediti $\hat{\alpha}(0) = \frac{m}{n_b}$

Opisana bootstrap procedura iz (Bartz-Beielstein, 2006) zadovoljava teorem konzistentnosti Beran and Ducharme 1991 (Horowitz, 2001). Dokaz se može naći u (Horowitz, 2001) *"Example 2.1"*. Preporučena metoda za smanjivanje varijance je Antihetic variates 2.5 zbog performansi i generičnosti.

Ostale su sljedeće nedefiniranosti:

1. Koji omjer $\frac{n}{m}$,iz jednadžbe (4.8) za konačan broj $T = n*m$ vrednovanja stohastičkog algoritma, je najbolji?

2. Kojim postupkom odabrati parove $(\vec{X}_1 \in B_1, \vec{X}_2 \in B_2)$ iz jednadžbe (4.8) odnosno (4.6)?

3. Zadržati (4.2) kao definiciju područja B ili odabrati neku drugu[2] ?

Za 1. nedefiniranost se supstitucijom $s_{\vec{X}j} = m(\vec{X}) + \delta_{\vec{X}j}$, gdje je $\delta_{\vec{X}j}$ odstupanje (šum) koje ovisi o varijanci $S_{\vec{X}}$, u jednadžbu (4.5) dobiva (4.10).

$$\theta = \lim_{\substack{n\to\infty \\ m\to\infty}} \frac{1}{m * n^N} \sum_{\vec{X}\in B} \sum_{j=1}^{m} (m(\vec{X}) + \delta_{\vec{X}j}) \tag{4.9}$$

$$\theta = \lim_{n\to\infty} \frac{1}{n^N} \sum_{\vec{X}\in B} m(\vec{X}) + \lim_{\substack{n\to\infty \\ m\to\infty}} \frac{1}{m * n^N} \sum_{\vec{X}\in B} \sum_{j=1}^{m} \delta_{\vec{X}j} \tag{4.10}$$

Uvođenjem jednadžbe (4.2) u (4.1) se dobiva (4.11).

$$\theta = \lim_{n\to\infty} \frac{1}{n^N} \sum_{\vec{X}\in B} m(\vec{X}) \tag{4.11}$$

Zatim se uvođenjem (4.11) u (4.10) dobiva (4.12).

$$\lim_{\substack{n\to\infty \\ m\to\infty}} \frac{1}{m * n^N} \sum_{\vec{X}\in B} \sum_{j=1}^{m} \delta_{\vec{X}j} = 0 \tag{4.12}$$

Jednadžba (4.12) pokazuje da šum teži u 0 za $n \to \infty, m \to \infty$. Ako su varijance od $S_{\vec{X}}, \vec{X} \in B$ dovoljno slične može se pokazati da vrijedi (4.13).

$$\lim_{n\to\infty} \frac{1}{n^N} \sum_{\vec{X}\in B} \delta_{\vec{X}} = 0 \tag{4.13}$$

Iz jednadžbe (4.11) se uviđa da se dobiva bolja aproksimacija θ što je n veći te iz (4.13) da se ukupan šum među uzorcima smanjuje što je n veći. Prema ovim zaključcima je najbolja $n = T, m = 1$ kombinacija. U situacijama kada ne vrijedi da su varijance od $S_{\vec{X}}, \vec{X} \in B$ dovoljno slične se pokazalo da u slučaju $n = T, m = 1$ kombinacije, algoritam podijeli i

[2] npr. skup slučajno odabranih točaka

usporedi preferira područja s manjom varijancom. Odnosno preferira definirane algoritme sa stabilnijim ponašanjem.

Za 2. nedefiniranost se pokazalo u praksi, što se tiče bootstrap algoritma, da ima vrlo mali utjecaj koji može biti pozitivan ili negativan. Što znači da je najbolje uparivati uzorke onako kako su poredani, a i najjednostavnije.

3. nedefiniranost se poziva na 1. nedefiniranost odnosno na jednadžbu (4.11) kada je n konačan i posljedicu koja proizlazi iz toga, koja se tu sad ne vidi ,ali je primjetna u izvornoj jednadžbi (4.2). A to je da kada je n konačan, tada $m(\vec{X})$ predstavlja dio domene $\vec{X}$ (*Riemman Integral*). Ako bi ti dijelovi domene bili različite veličine tada bi veći dijelovi domene imali manji utjecaj, odnosno uzorci koji ih predstavljaju, na vrijednost θ. To bi dovelo do preferiranja nekih dijelova domene što je potrebno izbjeći kako bi se dobila što kvalitetnija aproksimacija θ. Učinak ove pojave slabi sa povećanjem n i ovisi o ujednačenosti promatranog područja. Način podjele predložen u (4.2) je jedan od optimalnijih načina jer se već za male vrijednosti n može značajno smanjiti utjecaj te pojave. Zbog toga se i koristi. Što se tiče slučajnog odabira $\vec{X}, \vec{X} \in [\vec{X}_{MIN}, \vec{X}_{MAX}]$ očekuje se da će algoritam biti nestabilniji. Ispitivanje te pretpostavke se može vidjeti u 5.3.

4.3. Izvedbe

Na temelju metode podijeli i usporedi su razvijena tri algoritma:

- *All For All* (AFA)
- *One For One* (OFO)
- *All For One* (AFO)

Algoritam AFA je dosljedan metodi podijeli i usporedi. Algoritam OFO je nastao kao odgovor na veliku vremensku složenost algoritma AFA. Dok je algoritam AFO nastao kao odgovor na povećanu subjektivnost algoritma OFO. Svi ovi algoritmi su sposobni za sve tipove optimiranja navedenih u poglavlju 4.1.

Bez gubitka općenitosti se pretpostavlja problem minimizacije dobrote stohastičkih algoritama.

4.3.1. Algoritam AFA

Algoritam AFA je direktna realizacija metode podijeli i usporedi gdje je način testiranja definiranih algoritama i njihova usporedba izvedena prema teoretskoj podlozi iz poglavlja 4.2.

Definirani algoritam se sastoji od

- fun- stohastička funkcija čiji se parametri optimiraju.

– $područje$- definirano je za svaki parametar (kontinuiran ili diskretan) donja i gornja granica vrijednosti.

– $maxDubina$- za svaki parametar je definirano koliko puta se bude optimirao.

– $dobrota$- polje za brojanje dobrote, početna vrijednost je 0 .

Jedna napomena za diskretne parametre. Kod njih se mora biti na oprezu jer za susjedne diskretne vrijednosti često ne vrijedi da će, u slučaju kada se vrijednosti parametara stohastičkog algoritma razlikuju samo za taj jedan diskretan parametar, rezultirati sličnim ponašanjem algoritma[3]. Te ih zbog toga treba pažljivo podijeliti u zasebne grupe (područja), za koja su ponašanja algoritma unutar grupe dovoljno slična, s kojima se onda stvaraju dodatni definirani algoritmi nad istim stohastičkim algoritmom.

Pseudokod algoritma AFA je prikazan u opisu 4.1 na sljedećoj stranici. Jedina veća promjena u odnosu na metodu podijeli i usporedi su zasebni faktori dubine za svaku varijablu koja se optimizira. Do promjene je došlo zbog potrebe za takvom mogućnošću koja proistječe iz algoritama OFO i AFO.

Prva for petlja, koja se pojavljuje nakon while petlje, se podudara sa 1. i 2. korakom metode podijeli i usporedi. Gdje funkcija $brojAktivnihParametara(d)$ vraća broj parametara koje se još mogu optimirati kod definiranog algoritma d, ovisi o $maxDubina$ koja je definirana za svaki parametar kod svakog definiranog algoritma. Funkcija $split$ obavlja funkcionalnost 1. koraka za jedan definiran algoritam, gdje je način dijeljenja vrlo direktan. Područje svakog aktivnog parametra, aktivan je onaj koji se nije $maxDubina$ puta optimirao, se dijeli na $nChild$ jednakih intervala. Tako dobiveni intervali parametra x_i čine skup S_i. Ako parametar x_j nije aktivan onda je njegov interval jedini član skupa S_j. Sada se od svake uređene N-torke kartezijevog produkta $S_1 \chi S_2 \chi ... \chi S_N$ radi jedno dijete. Skup novo nastale djece se vraća kao izlaz funkcije $split$. 2. korak generičkog algoritma obavlja funkcija $testiraj$ koja testira, uzorkuje, jedan definirani algoritam na način objašnjen u poglavlju 4.2. U slučaju da $brojAktivnihDimenzija(d)$ vrati 0, što označava da se d ne mora više optimirati, d se dodaje skupu novih.

Kada skup novih čine samo stari definirani algoritmi, optimiranje se prekida.

Druga for petlja unutar while petlje obavlja 3. korak. Gdje je $bootstrap$ funkcija realizirana na način objašnjen u 4.2. Vrijednost varijable r, koju vraća $bootstrap$ funkcija, je uočena razina signifikantnosti koja se koristi na način objašnjen u 4.2.

Treća for petlja unutar while petlje obavlja 4. korak.

Vremenska složenost je izražena za broj testiranja stohastičke funkcije čiji se parametri optimiraju jer će to uglavnom daleko najviše vremena trošiti u odnosu na ostatak algoritma.

[3]npr. kod generacijskog genetskog algoritma, parametar elitizam.

Algorithm 4.1 AFA pseudokod

```
1: Ulaz:  startList - lista početnih definiranih algoritama koji se optimiziraju
2: Ulaz:  alfa - traženi nivo signifikantnosti kod usporedbe dvaju definiranih algoritama
3: Ulaz:  nBoot - broj generiranih setova uzoraka kod Bootstrap metode
4: Ulaz:  dimTest - broj testiranja po aktivnoj dimenziji definiranog algoritma
5: Ulaz:  nChild - broj djece po dimenziji područja definiranog algoritma
6: Izlaz: definirani algoritmi s najmanjom dobrotom
7: while true do
8:    novi := {∅}
9:    aktivni := 0
10:   for all d : startList do
11:      n := brojAktivnihParametara(d)
12:      if n > 0 then
13:         aktivni++
14:         djeca := split(d, nChild)
15:         novi ∪ := djeca
16:         for all djete : djeca do
17:            testiraj(dijete, dimTest)
18:      else
19:         d.dobrota := 0
20:         novi ∪ := d
21:   if aktivni == 0 then
22:      return startList
23:   startList := {∅}
24:   for all {(s1,s2)| s1,s2 ϵ novi, s1≠s2} do
25:      r := bootstrap(s1, s2, nBoot)
26:      if r < alfa then
27:         s1.dobrota++
28:      else if r>1-alfa then
29:         s2.dobrota++
30:   minDobrota := novi.size
31:   for all d : novi do
32:      if d.dobrota < minDobrota then
33:         minDobrota := d.dobrota
34:   for all d : novi do
35:      if d.dobrota == minDobrota then
36:         startList ∪ := d
```

Algorithm 4.2 OFO pseudokod

```
1: Ulaz:  startList - lista početnih definiranih algoritama koji se optimiziraju
2: Ulaz:  alfa - traženi nivo signifikantnosti kod usporedbe dvaju definiranih algoritama
3: Ulaz:  nBoot - broj generiranih setova uzoraka kod Bootstrap metode
4: Ulaz:  dimTest - broj testiranja po aktivnoj dimenziji definiranog algoritma
5: Ulaz:  nChild - broj djece po dimenziji područja definiranog algoritma
6: Izlaz: definirani algoritmi s najmanjom dobrotom
7: for all t = 1, 2, .., N do
8:    for all d : startList do
9:       d.andDubina(t)
10:   startList = AFA(startList, alfa, nBoot, dimTest, nChild)
```

Za jednu iteraciju algoritma t najgora složenost je

$$O\left(|startList_t| * (nChild * dimTest)^N\right)$$

odnosno za cijelo izvođenje algoritma najgora složenost je

$$O\left(|startList_0| * nChild^{N*(\max(maxDubina_i|i=1,2..,N)-1)} * (nChild * dimTest)^N\right)$$

[4][5]. U praksi se ne događa da sva djeca nastala u iteraciji t dođu do iteracije $t+1$, obično je $|startList_t|\,\epsilon\,[1,6]$.No to ovisi o problemu.

Prostorna složenost se odnosi na najveći mogući broj definiranih algoritama tijekom izvođenja te iznosi

$$O\left(|startList_0| * nChild^{N*\max(maxDubina_i|i=1,2..,N)}\right)$$

.

4.3.2. Algoritam OFO

Motiv za nastanak algoritma OFO je velika vremenska složenost algoritma AFA. Ideja je jednostavna, optimira se parametar po parametar. Izvedba je također jednostavna. Oslanja se na mogućnost specifikacije $maxDubine$ svakog parametra. Pseudokod algoritma OFO se nalazi u opisu 4.2.

Za svaki parametar provodi jednaku radnju, gdje prvo kod svakog definiranog algoritma s metodom $andDubina$ postavi sve $maxDubina_i = 0, \quad i = 1, 2, ..., N, \quad i \neq t$, a

[4] $|startList_0|$ je kardinalnost skupa startList u iteraciji 0, prije bilo kakvog optimiranja.

[5] Pod $maxDubina_i$ se prešutno podrazumijeva da je to maksimalna dubina za parametar pod indeksom i za sve definirane algoritme iz skup $startList_0$.

$maxDubina_t$ postavlja na originalnu vrijednost. Za optimiranje takvih izmijenjenih definiranih algoritama se koristi algoritam AFA.

Vremenska složenost je izražena za broj testiranja stohastičke funkcije čiji se parametri optimiraju jer će to uglavnom daleko najviše vremena trošiti u odnosu na ostatak algoritma. Za iteraciju t algoritma OFO najgora vremenska složenost je

$$O\left(|startList_t| * nChild^{maxDubina_t} * dimTest\right)$$

[6] odnosno za cijelo izvođenje algoritma OFO najgora složenost je

$$O\left(|startList_0| * nChild^{\sum_{t=1}^{N} maxDubina_t} * dimTest\right)$$

U praksi se ne događa da sva djeca nastala u iteraciji t dođu do iteracije $t+1$, obično je $|startList_t|\,\epsilon\,[1,6]$. No to ovisi o problemu.

Prostorna složenost se odnosi na najveći mogući broj definiranih algoritama tijekom izvođenja te iznosi

$$O\left(|startList_0| * nChild^{\sum_{t=1}^{N} maxDubina_t}\right)$$

Algoritam OFO izbjegava eksponencijalnu složenost u ovisnosti o broju parametara N. Nedostaci su lošije optimiranje te pojava subjektivnosti u obliku poretka parametara stohastičkog algoritma. Očito je da će prvi parametar imati veći utjecaj od zadnjeg.

Mogući dodatak je da se može specificirati početna konfiguracija u području optimiranja parametara čije će se vrijednosti koristiti kao zamjena za parametre koji se još nisu poćeli optimirati tijekom izvođenja.

4.3.3. Algoritam AFO

Algoritam AFO je nastao s motivom zadržavanja benefita algoritma OFO, ali tako da se izbjegne dodana subjektivnost u poretku parametara. Ideja je opet vrlo jednostavna. U istoj iteraciji se optimiraju svi parametri zasebno. Pseudokod algoritma AFO se nalazi u opisu 4.3 na sljedećoj stranici.

Najvažniji dio algoritma AFO je metoda $splitPerActive$ koja operira nad definiranim algoritmom. Ona za svaki aktivni parametar t definiranog algoritma stvara novi definirani algoritam koji ima $maxDubina_i = 0, \quad i = 1,2,...,N, \quad i \neq t,$ a $maxDubina_t = 1$. Metoda vraća novo stvorene definirane algoritma, a ako nema aktivnih parametara vraća definirani algoritam nad kojim operira. Sad bi trebao biti i jasan uvjet prekida rada algoritma AFO. Za takve nove definirane algoritme se izvodi optimiranje algoritmom AFA.

[6] $maxDubina_t$ je maksimalna dubina za parametar s indeksom t među definiranim algoritmima skupa $startList_t$.

Algorithm 4.3 AFO pseudokod

```
1: Ulaz:  startList - lista početnih definiranih algoritama koji se optimiziraju
2: Ulaz:  alfa - traženi nivo signifikantnosti kod usporedbe dvaju definiranih algoritama
3: Ulaz:  nBoot - broj generiranih setova uzoraka kod Bootstrap metode
4: Ulaz:  dimTest - broj testiranja po aktivnoj dimenziji definiranog algoritma
5: Ulaz:  nChild - broj djece po dimenziji područja definiranog algoritma
6: Izlaz: definirani algoritmi s najmanjom dobrotom
7: while true do
8:    novi = {∅}
9:    for all d : startList do
10:      novi ∪= d.splitPerActive()
11:   if |novi|==|startList| then
12:      return startList
13:   startList = AFA (novi, alfa, nBoot, dimTest, nChild)
```

Vremenska složenost je izražena za broj testiranja stohastičke funkcije čiji se parametri optimiraju jer će to uglavnom daleko najviše vremena trošiti u odnosu na ostatak algoritma. Za iteraciju t algoritma OFO najgora vremenska složenost je

$$O\left(|novi_t| * nChild * dimTest\right)$$

odnosno za cijelo izvođenje algoritma AFO najgora složenost je

$$O\left(|startList_0| * (N * nChild)^{\sum_{t=1}^{N} maxDubina_t} * dimTest\right)$$

U praksi se ne događa da sva dijeca nastala u iteraciji t dođu u iteraciju $t+1$, obično je $|startList_t|\,\epsilon\,[1,3]$.

Prostorna složenost se odnosi na najveći mogući broj definiranih algoritama tijekom izvođenja te iznosi

$$O\left(|startList_0| * nChild^{\sum_{t=1}^{N} maxDubina_t}\right)$$

U 5.2 je navedeno za svako ispitivanje točan broj testiranja stohastičkog algoritma kojemu se parametri optimiraju. Te se primjećuje da broj testiranja nije niti približno velik kao u najgorim slučajevima.

Mogući dodatak je da se umjesto $maxDubina$ definira maksimalni broj iteracija algoritma AFO. To bi za posljedicu imalo dinamičko određivanje bitnijih parametara. Ali tada bi točnost određivanja tih bitnih parametara bila podložna parametru $nChild$ na način objašnjen u 4.4.

4.4. Parametri

Za parametre vrijede sljedeće smjernice za njihov odabir, ovisno o algoritmu i željenoj preciznosti optimiranja. Parametri:

- $alfa[0, 0.5\rangle$- *minimalna signifikantnost* prilikom donošenja odluke koje je područje bolje. U globalu je bolje da je alfa što manji, ali s manjim alfa će se složenost izvođenja algoritma približavati najgoroj složenosti. Variranjem faktora $alfa$ se može postići najgora i najbolja složenost algoritama što će imati odgovarajući učinak na optimizaciju. Uobičajena vrijednost je kao kod eksperimenata $[0.01, 0.05]$.
- $nBoot[1, \infty\rangle$ - *broj reuzorkovanja kod bootstrap metode.* Bolje je što veća vrijednost, cc. $[2000, 4000]$, veće vrijednosti nemaju učinka na bootstrap zbog manjka numeričke preciznosti u računalima. Ima za posljedicu glađenje funkcije uočene razine signifikantnosti.
- $dimTest[1, \infty\rangle$- *broj testiranja po dimenziji* . Najbitniji parametar za vjerodostojnost optimiranja i smanjenje šuma. Također jedini parametar kod kojeg se razlikuju smjernice za različite algoritme. Kod AFA algoritma vrijednost $dimTest$ ne mora biti velika, cc. $[3, 5]$, jer prema (4.12) je za smanjenje šuma među podacima bitan samo ukupan broj testiranja koji će iznositi $dimTest^N$. Dok kod OFO i AFO algoritama $dimTest$ treba biti veći, cc. $[8, 20]$, kako bi se dobilo željeno ponašanje. Najbolji pokazatelj da je vrijednost $dimTest$ premala je kada performansa za najbolje područje pronađeno optimiranjem[7], za magnitudu bolja od performanse utvrđene detaljnim testiranjem tog istog područja.
- $nChild[2, \infty\rangle$- *podjela po parametru* je način definiranja željene globalne preciznosti optimiranja. Veće vrijednosti imaju dobar učinak na ponašanje algoritama zbog ravnopravnijeg ispitivanja domene problema, ali pod cijenu veće kompleksnosti.
- $maxDubina[0, \infty\rangle$ - *dubina optimiranja* definirana za svaki parametar kod svakog definiranog algoritma zasebno. To je parametar preko kojeg se definira željena preciznost optimiranja na razini parametra stohastičkih algoritama čiji se parametri optimiraju.

[7] Gdje je performansa dobivena iz rezultata testiranja koji su korišteni tijekom optimiranja

5. Ispitivanje

Ispitivanje bi trebala dati odgovore na pitanja:

- koji algoritam od AFA, AFO i AFO najbolje optimira parametre nekog stohastičkog algoritma?
- koji način odabira točaka, za ispitivanje, iz domene parametara je stabilniji?

Koristi se generacijski genetski algoritam, stohastički algoritam kojemu se optimiraju parametri. Operator križanja je blend crossover s fiksnim faktorom križanja od 0.2. Operator mutiranja koristi slučajnu varijablu s normalnom razdiobom koja se množi s faktorom $mutCh$. Roditelji se nasumično odabiru iz stare generacije. Parametri koji se optimiraju su:

- pop- veličina populacije. Nad domenom $[20, 500]$
- $birth$- broj nove djece. Nad domenom $[20, 600]$
- $mutCh$- faktor mutacije. Nad domenom $[0, 0.4]$
- $elite$- elitizam. Nad domenom $[0, 20]$

Korištena su ostvarenja algoritama AFA, AFO i OFO, gdje neki imaju par modifikacija. Ostvareni algoritam AFA je identičan onom iz poglavlja 4.3.1. Ostvareni algoritam OFO je isti kao i u poglavlju 4.3.2 osim što ima dodatak naveden na kraju tog poglavlja. Ostvareni algoritam AFO je isti kao u poglavlju 4.3.3, ali s dodatkom navedenim na kraju tog poglavlja gdje je korištena maksimalna iteracija jednaka20.

Korištene vrijednosti parametara su:

- $alfa = 0.05$
- $nBoot = 4000$
- $dimTest = 3$, algoritam AFA
- $dimTest = 10$, algoritam AFO i OFO
- $nChild = 2$
- $maxDubina_{pop} = 5, \quad maxDubina_{birth} = 5, \quad maxDubina_{mutCh} = 6, \quad maxDubina_{elite} = 4$

Parametri su odabrani prema smjernicama iz poglavlja 4.4 s ciljem što manje kompleksnosti.

Provedena su dva ispitivanja. Prvi je usporedba rezultata optimiranja, sva tri algoritma AFA, OFO i AFO, parametara generacijskog genetskog algoritma za sve testne funkcije. Drugo ispitivanje je ispitivanje utjecaja slučajnog odabira konfiguracije parametara iz domene parametara za testiranje protiv uređenog načina opisnog u poglavlju 4.2. Uređeni način se koristi za sve izvedbe algoritama AFA, AFO i OFO osim ako nije drugačije navedeno.

U sljedećem odjeljku su navedene sve ispitne funkcije koje optimira korišteni generacijski genetski algoritam.

5.1. Ispitne funkcije

Funkcije upotrijebljene u svrhu ispitivanja dobrote optimiranja parametara korištenog generacijskog genetskog algoritma se nalaze tablici 5.1. Njihov detaljan opis se nalazi u tablici 5.2.

Tablica 5.1: Ispitne funkcije

no.	naziv ispitne funkcije	max evaluacija
1.	Sphere Model	150000
2.	Schwefels problem 2.22	200000
3.	Schwefels problem 1.2	500000
4.	Schwefels problem 2.21	500000
5.	Generalized Rosenbrocks Function	2000000
6.	Step Function	300000
7.	Quartic Function with Noise	300000
8.	Generalized Schwefels Problem 2.26	500000
9.	Generalized Rastrigins Function	500000
10.	Ackleys Function	150000
11.	Generalized Griewank Function	200000
12.	Penalized Function P8	150000
13.	Penalized Function P16	150000
14.	Shekel's foxholes function	10000
15.	Kowalik's function	400000
16.	Six-hump camel-back function	10000

Gdje stupac test funkcija sadrži matematičke formule funkcija. Stupac n sadrži korištenu dimenzionalnost problema. Stupac domena sadrži domenu za svaku dimenziju problema.

Tablica 5.2: Detalji ispitnih funkcija

no	ispitna funkcija	n	domena	f_{min}
1.	$f_1(x) = \sum_{i=1}^{n} x_i^2$	30	$[-100, 100]^n$	0
2.	$f_2(x) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_{i\rvert} \rvert$	30	$[-10, 10]^n$	0
3.	$f_3(x) = \sum_{i=1}^{n} \left(\sum_{i=1}^{n} x_i\right)^2$	30	$[-100, 100]^n$	0
4.	$f_4(x) = \max\{\lvert x_i \rvert, 1 \leq i \leq n\}$	30	$[-100, 100]^n$	0
5.	$f_5(x) = \sum_{i=1}^{n-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + \left(x_i - 1\right)^2\right]$	30	$[-30, 30]^n$	0
6.	$f_6(x) = \sum_{i=1}^{n} \lfloor x_i + 0.5 \rfloor$	30	$[-100, 100]^n$	0
7.	$f_7(x) = \sum_{i=1}^{n} i x_i^4 + random[0, 1)$	30	$[-1.28, 1.28]^n$	0
8.	$f_8(x) = \sum_{i=1}^{n} -x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)$	30	$[-500, 500]^n$	-12569.5
9.	$f_9(x) = \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right]$	30	$[-5.12, 5.12]^n$	0
10.	$f_{10}(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right)$ $-\exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e$	30	$[-32, 32]^n$	0
11.	$f_{11}(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	30	$[-600, 600]^n$	0
12.	$f_{12}(x) = \frac{\pi}{n}\left\{10\sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2\right.$ $\left[1 + 10\sin^2(\pi y_{i+1})\right] + (y_n - 1)^2\Big\}$ $+ \sum_{i=1}^{n} u(x_i, , 10, 100, 4)$ $y_i = 1 + \frac{1}{4}(x_i + 1)$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m & x_i > a \\ 0 & -a \leq x_i \leq a \\ k(-x_i - a)^m & x_i < -a \end{cases}$	30	$[-50, 50]^n$	0
13.	$f_{13}(x) = 0.1\{\sin^2(3\pi x_1)$ $+ \sum_{i=1}^{n-1}(x_i - 1)^2\left[1 + \sin^2(3\pi x_{i+1})\right]$ $+ (x_n - 1)\left[1 + \sin^2(2\pi x_n)\right]\}$ $+ \sum_{i=1}^{n} u(x_i, , 5, 100, 4)$	30	$[-50, 50]^n$	0
14.	$f_{14}(x) = \left[\frac{1}{500} + \sum_{j=1}^{25}\frac{1}{j + \sum_{i=1}^{2}(x_i - a_{ij})^6}\right]$	2	$[-65.536, 65.536]^n$	1
15.	$f_{15}(x) = \sum_{i=1}^{11}\left[a_i - \frac{x_1\left(b_i^2 + b_i x_2\right)}{b_i^2 + b_i x_3 + x_4}\right]^2$ $\vec{a} = \{0.1957, 0.1947, 0.1735, 0.1600, 0.0844,$ $0.0627, 0.0456, 0.0342, 0.0323, 0.0235, 0.0246\}$ $\vec{b} = \{4, 2, 1, 0.5, 0.25, 1./6$ $, 0.125, 0.1, 1./12, 1./14, 0.0625\}$	4	$[-5, 5]^n$	0.0003075
16.	$f_{16}(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1 x_2 - 4x_2^2 + 4x_2^4$	2	$[-5, 5]^n$	-1.0316285

Tablica 5.3: Rezultati podešavanja parametara evolucijskog algoritma AFA algoritmom

	AFA					
no.	pop	birth	mutCh	elite	broj testiranja	broj područja
1.	$[95, 110]$	$[491, 509]$	$[0.05, 0.05625]$	$[4, 4]$	35556	102
2.	$[65, 80]$	$[436, 455]$	$[0, 0.00625]$	$[19, 19]$	41766	55
3.	$[20, 35]$	$[38, 56]$	$[0.10625, 0.1125]$	$[19, 19]$	20748	3
4.	$[20, 35]$	$[346, 364]$	$[0.2, 0.20625]$	$[18, 18]$	16752	5
5.	$[200, 215]$	$[328, 346]$	$[0.05, 0.05625]$	$[10, 10]$	36534	66
6.	$[20, 35]$	$[20, 38]$	$[0.24375, 0.25]$	$[18, 18]$	13674	5
7.	$[50, 65]$	$[400, 418]$	$[0.0125, 0.01875]$	$[18, 18]$	18552	17
8.	$[80, 95]$	$[527, 545]$	$[0.3937, 0.4]$	$[18, 18]$	27474	8
9.	$[470, 485]$	$[563, 581]$	$[0, 0.00625]$	$[10, 10]$	13488	8
10.	$[95, 110]$	$[328, 346]$	$[0.1, 0.10625]$	$[19, 19]$	25806	14
11.	$[80, 95]$	$[382, 400]$	$[0.2187, 0.225]$	$[15, 15]$	31518	31
12.	$[290, 305]$	$[581, 600]$	$[0.2, 0.20625]$	$[6, 6]$	30576	39
13.	$[20, 35]$	$[491, 509]$	$[0.206, 0.2125]$	$[19, 19]$	17868	11
14.	$[185, 200]$	$[581, 600]$	$[0.3875, 0.39375]$	$[14, 14]$	36678	239
15.	$[215, 230]$	$[146, 165]$	$[0.1875, 0.025]$	$[6, 6]$	35280	27
16.	$[20, 35]$	$[581, 600]$	$[0, 0.00625]$	$[11, 11]$	31716	8

Te stupac f_{min} sadrži vrijednosti globalnog minimuma problema. Dok su u tablici 5.1 prikazana imena testnih funkcija i maksimalni broj evaluacija testne funkcije što služi kao uvjet prekida izvođenja generacijskog genetskog algoritma.

5.2. Usporedba rezultata optimiranja

U ovom ispitivanju je provedeno optimiranje s algoritmima AFA, OFO i AFO nad generacijskim genetskim algoritmom za sve testne funkcije iz 5.1. Rezultati optimiranja se nalaze u tablicama 5.3, 5.4 i 5.5. Svaka tablica prikazuje za određeni algoritam za pojedine probleme rezultate optimiranja. U tablicama je prikazano samo jedno područje po problemu, gdje je to jedno područje odabrano na temelju najmanjeg očekivanja među područjima vraćenima kao rezultat optimiranja. Broj testiranja označava koliko je puta pokrenut generacijski genetski algoritam. Dok je zadnji stupac broj područja, broj područja koja je vratio algoritam kao rezultat optimiranja.

Veliki broj područja vraćenih kao rezultat optimiranja može ukazivati na preveliku osjetljivost algoritma za taj problem.

Tablica 5.4: Rezultati podešavanja parametara evolucijskog algoritmaOFO algoritmom

	OFO					
no.	pop	birth	mutCh	elite	broj testiranja	broj područja
1.	$[20, 35]$	$[128, 146]$	$[0.05, 0.0625]$	$[3, 3]$	542	2
2.	$[20, 35]$	$[20, 38]$	$[0.0125, 0.025]$	$[10, 10]$	464	4
3.	$[20, 35]$	$[20, 38]$	$[0.05, 0.0625]$	$[5, 5]$	464	4
4.	$[20, 35]$	$[165, 183]$	$[0.05, 0.0625]$	$[3, 3]$	543	3
5.	$[20, 35]$	$[364, 382]$	$[0.0125, 0.025]$	$[4, 4]$	905	5
6.	$[20, 35]$	$[56, 74]$	$[0.1, 0.1125]$	$[3, 3]$	502	2
7.	$[20, 35]$	$[20, 38]$	$[0, 0.0125]$	$[4, 4]$	462	2
8.	$[65, 80]$	$[38, 56]$	$[0.1625, 0.175]$	$[17, 17]$	723	3
9.	$[35, 50]$	$[545, 563]$	$[0.025, 0.0375]$	$[10, 10]$	886	6
10.	$[20, 35]$	$[436, 455]$	$[0.2, 0.2125]$	$[0, 0]$	841	1
11.	$[20, 35]$	$[38, 56]$	$[0.2, 0.2125]$	$[15, 15]$	581	1
12.	$[20, 35]$	$[527, 545]$	$[0.0375, 0.05]$	$[10, 10]$	682	2
13.	$[20, 35]$	$[328, 346]$	$[0.1, 0.1125]$	$[5, 5]$	822	2
14.	$[320, 335]$	$[310, 328]$	$[0.2875, 0.3]$	$[19, 19]$	885	5
15.	$[50, 65]$	$[38, 56]$	$[0.025, 0.0375]$	$[18, 18]$	1206	6
16.	$[35, 50]$	$[92, 110]$	$[0, 0.0125]$	$[4, 4]$	643	3

Tablica 5.5: Rezultati podešavanja parametara evolucijskog algoritma AFO

	AFO					
no.	pop	birth	mutCh	elite	Broj testiranja	Broj područja
1.	$[22, 22]$	$[20, 56]$	$[0, 0.1]$	$[19, 19]$	7600	5
2.	$[65, 80]$	$[264, 273]$	$[0, 0.0125]$	$[19, 19]$	9280	9
3.	$[20, 27]$	$[24, 29]$	$[0, 0.05,]$	$[18, 18]$	6000	6
4.	$[20, 20]$	$[20, 29]$	$[0, 0.1]$	$[15, 20]$	6640	10
5.	$[20, 23]$	$[20, 165]$	$[0, 0.00625]$	$[0, 0]$	6320	11
6.	$[21, 22]$	$[20, 56]$	$[0.0, 0.4]$	$[6, 6]$	5360	7
7.	$[50, 65]$	$[103, 106]$	$[0, 0.025]$	$[10, 12]$	6480	5
8.	$[35, 50]$	$[563, 600]$	$[0, 0.4]$	$[2, 5]$	4800	7
9.	$[485, 500]$	$[581, 590]$	$[0, 0.00625]$	$[0, 2]$	8560	10
10.	$[20, 21]$	$[246, 255]$	$[0, 0.4]$	$[19, 19]$	7920	5
11.	$[65, 65]$	$[219, 237]$	$[0, 0.4]$	$[16, 17]$	8160	7
12.	$[288, 290]$	$[563, 572]$	$[0.15, 0.1625]$	$[10, 20]$	14560	22
13.	$[20, 20]$	$[92, 165]$	$[0, 0.4]$	$[17, 18]$	8160	18
14.	$[46, 46]$	$[527, 600]$	$[0.2, 0.3]$	$[11, 12]$	24000	30
15.	$[52, 53]$	$[165, 310]$	$[0.020312, 0.02187]$	$[0, 10]$	12480	11
16.	$[20, 260]$	$[310, 600]$	$[0.0017578, 0.00195]$	$[15, 15]$	2480	1

Tablica 5.6: Zadana konfiguracija

pop	birth	mutCh	elite
50	50	0.13	1

OFO daje najbolji omjer optimiranja po broju ispitivanja što se vidi iz tablica 5.4 i 5.7.

Ono što se ne vidi u tablici 5.5 za AFO algoritam je da je bilo dodijeljeno previše iteracija za optimizaciju što se očitavalo u optimizaciji diskretnih parametara na decimalu[1]. Također je način izvedbe algoritma utjecao na odabir ovih područja. Zbog načina kako se računa pozicija neaktivnog parametra za potrebe testiranja, a to je da se uzme sredina domene.

Ispitivanja za usporedbu optimiranja su provedena za četiri konfiguracije parametara po testnoj funkciji. Prve tri konfiguracije su sredine područja navedenih u tablicama 5.3, 5.4, 5.5 dok je četvrta *zadana* konfiguracija. *Zadana* je ne optimizirana preporučena konfiguracija koja je također služila kao i početna točka za optimizaciju kod algoritma OFO. Njene vrijednosti se nalaze u tablici 5.6.

[1]Program napisan u Javi koji je bio namijenjen za ispitivanja, za diskretne parametre koristi tip podatka float što je dovelo do takvog ponašanja. Izvedba namijenjena za ECF koristi ispravan tip podatka.

Tablica 5.7: Rezultati ispitivanja rezultata optimiranja parametara genetskog algoritma

no.	AFA	AFO	OFO	zadana	AFA	AFO	OFO	zadana
	očekivanje				standardna devijacija			
1.	1,896	0,0278	10,105	23,763	3,907	0,00289	0,00748	77,437
2.	0,142	0,225	0,28	3,504	0,212	0,399	0,0161	0,414
3.	0,0368	0,0112	0,0816	2,446	0,00504	0,00134	0,0113	1,180
4.	0,24	0,06	0,0865	2,863	0,0135	0,00379	0,246	1,743
5.	41,605	5,623	32,415	94,030	27,541	17,836	30,847	86,941
6.	0.0	0.0	0.0	1224.1	0.0	0.0	0.0	789.481
7.	0,00108	8,579E-4	0,00101	0,445	2,863E-4	2,657E-4	4,014E-4	0,141
8.	-7401,58	-7398,01	-6687,3	-6323,1	631,972	638,820	694,836	808,944
9.	5,668	2,843	59,207	98,110	2,846	1,723	14,681	12,991
10.	0,524	1,068	1,873	12,065	0,0447	0,0715	0,109	2,263
11.	0,0484	0,0394	0,0434	15,922	0,00989	0,0173	0,0265	8,692
12.	0.064	0.137	17.343	17.0969	0.14	0.303	11.557	7.02
13.	0,0909	0,0746	0,535	9,371	0,00936	0,00841	1,0369	9,971
14.	1,00005	1,0	1,310	1,74	2,84E-4	1,84E-8	0,425	0,438
15.	3,831E-4	3,67E-4	0,00124	0,00131	1,623E-4	2,348E-4	0,00391	0,00391
16.*	0	0	1E-7	1,57E-4	1,632E-8	8,362E-9	4,474E-8	1,68E-4

Rezultati ispitivanja svih konfiguracija se nalaze u tablici 5.7 [2]. Podaci su skupljeni izvođenjem generacijskog algoritma 50 puta za svaku konfiguraciju kod svakog testnog problema gdje je uvjet zaustavljanja generacijskog genetskog algoritma maksimalni broj evaluacija ispitne funkcije prema tablici 5.1.

Iz rezultata ispitivanja[3] se vidi da su konfiguracije AFA algoritma 4 puta najbolje, usporedbom očekivanja, a konfiguracije AFO algoritma kod ostalih 12 slučajeva. U svim slučajevima su konfiguracije od AFA i AFO algoritama bolje od defaultne konfiguracije. To vrijedi i za OFO algoritam osim za 12. testnu funkciju.

Najveća anomalija su loši rezultati za AFA algoritam, loši u usporedbi s AFO algoritmom kada se uzme u obzir broj izvođenja genetskog generacijskog algoritma. Mogući i najvjerojatniji uzrok je poduzorkovanost područja oko donjih granica svih parametara pogotovo za parametar pop. Usporedbom područja iz tablica 5.3 i 5.5 se može primijetiti da je AFO češće davao područja, koja su blizu donjim granicama parametara, od AFA algoritma. Ova

[2]Podaci u tablici su zaokružene na maksimalno prve tri ne nula znamenke iza decimalne točke radi preglednosti.

[3]Očekivanja kod 16. problema je oduzeto od globalnog minimuma

pretpostavka zahtjeva dodatne testove koji nisu predviđeni u ovom radu te će biti obavljeni u mojim budućim radovima. Jedan od načina s kojim bi se moglo izbjeći poduzorkovanost je podjela početnog definiranog algoritma na više dijelova. S naglaskom na područja blizu rubnih granica. Mana je što bi se s time unosilo više subjektivnost u optimizaciju. U slučaju da ova anomalija proizlazi iz AFA algoritma i nema drugog uzorka, tada rezultati ispitivanja pokazuju da AFO bolje optimizira za manji broj ispitivanja od AFA algoritma.

Zanimljivi su i slučajevi kada optimirane postavke od OFO imaju bolja svojstva od optimiranih postavka AFA algoritma. Ima ukupno tri takva slučaja za 4., 5. i 11. ispitnu funkciju. Za to je najvjerojatnije razlog isti kao u prethodnoj anomaliji.

Jedino za 8. testnu funkciju nije postignuto značajnije poboljšanje. Moguće je da to ukazuje na ograničenja generacijskog genetskog algoritma ili na odabranu premalu domenu parametara.

Kod 6. i 16. ispitne funkcije za sve tri kombinacije optimiranih parametara od algoritama AFA,AFO i OFO je generacijski genetski algoritam našao globalni optimum[4].

Slike od 5.1 do 5.16 prikazuju superponirane dobrote najboljih rješenja tijekom rada generacijskog genetskog algoritma, za sve postavke, grupirane po testnim funkcijama.

Vertikalna os je u logaritamskoj skali i predstavlja razliku superponiranih dobrota najboljih rješenja i globalnog minimuma testne funkcije.

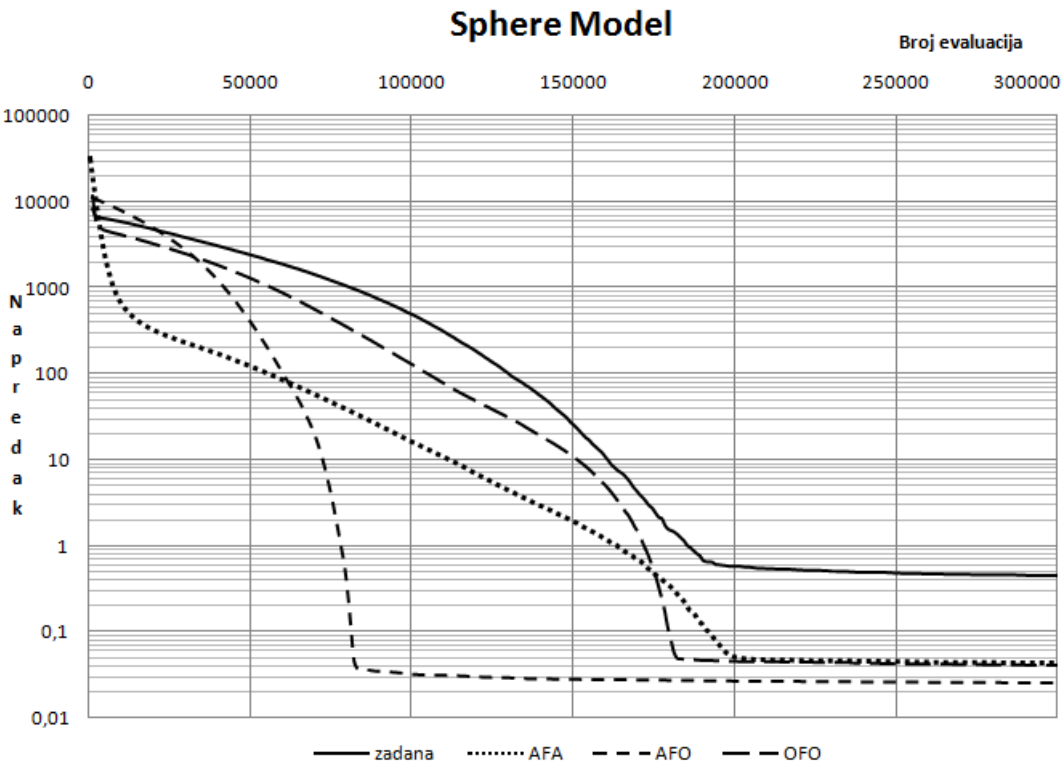
Horizontalna os predstavlja broj evaluacija testne funkcije od strane generacijskog genetskog algoritma.

Nestanak linije na grafovima 5.6 i 5.14 znači da je pronađeno najbolje rješenje.

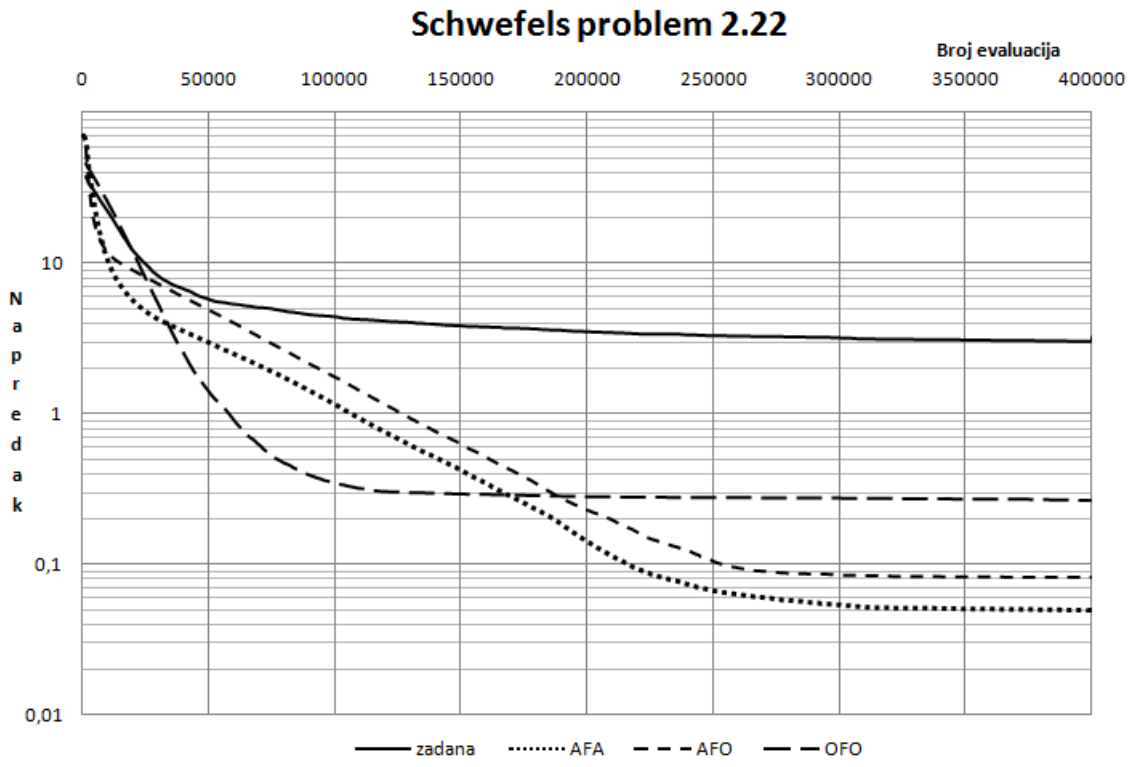
Za izradu grafova na slikama 5.1, 5.2 i 5.4 je kod uvjeta prekida rada genetskog algoritma korišten dvostruki iznos maksimalnog broja evaluacija ispitne funkcije od navedenih u tablici 5.1. Jer su u suprotnom ti grafovi završavali na zanimljivim dijelovima.

Zaključak ovog ispitivanja je da AFO algoritam, za zadane testne funkcije, najbolje optimizira. Ali OFO ima najbolji omjer optimiranja po broju testiranja stohastičkog algoritma komu se parametri optimiraju.

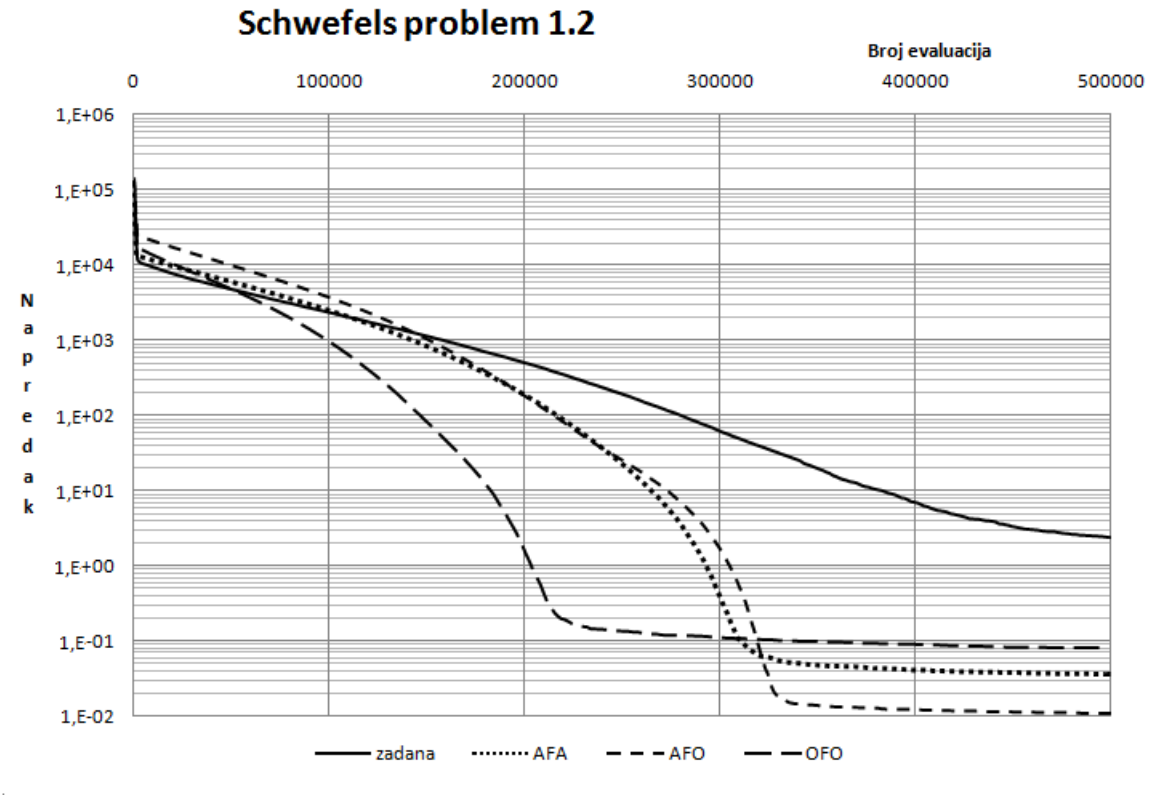
[4]Sa numeričkom preciznošću float tipa podatka



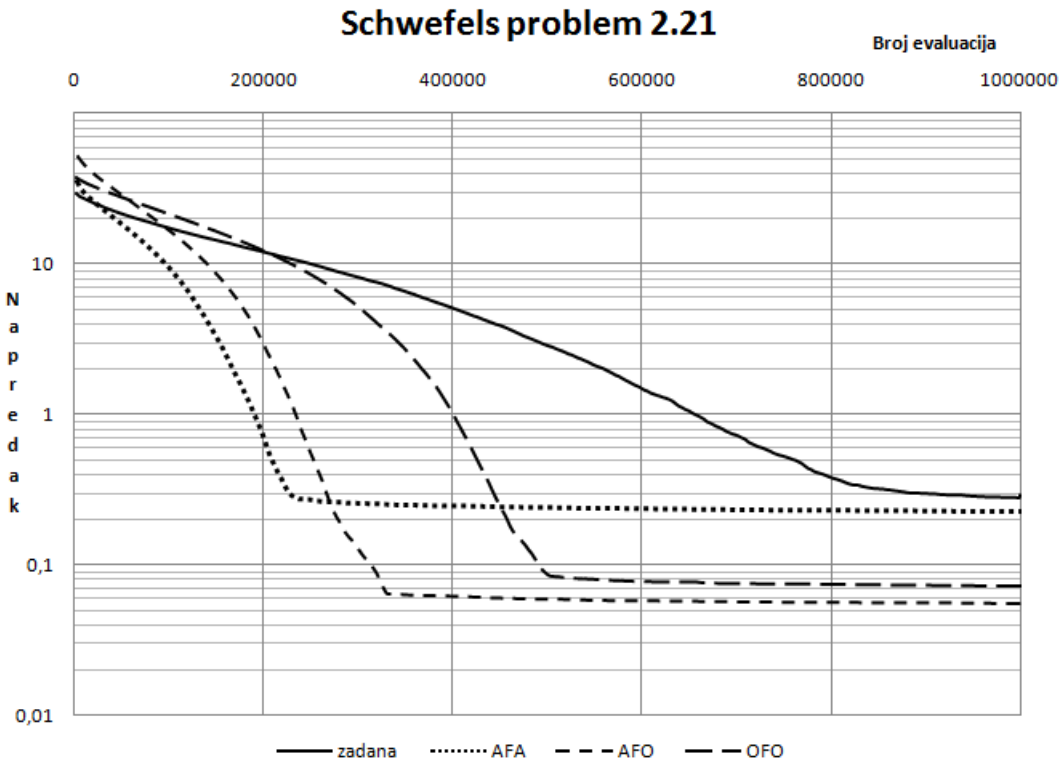
Slika 5.1: Najbolje rješenje prilikom optimiranja ispitne funkcije Sphere Model



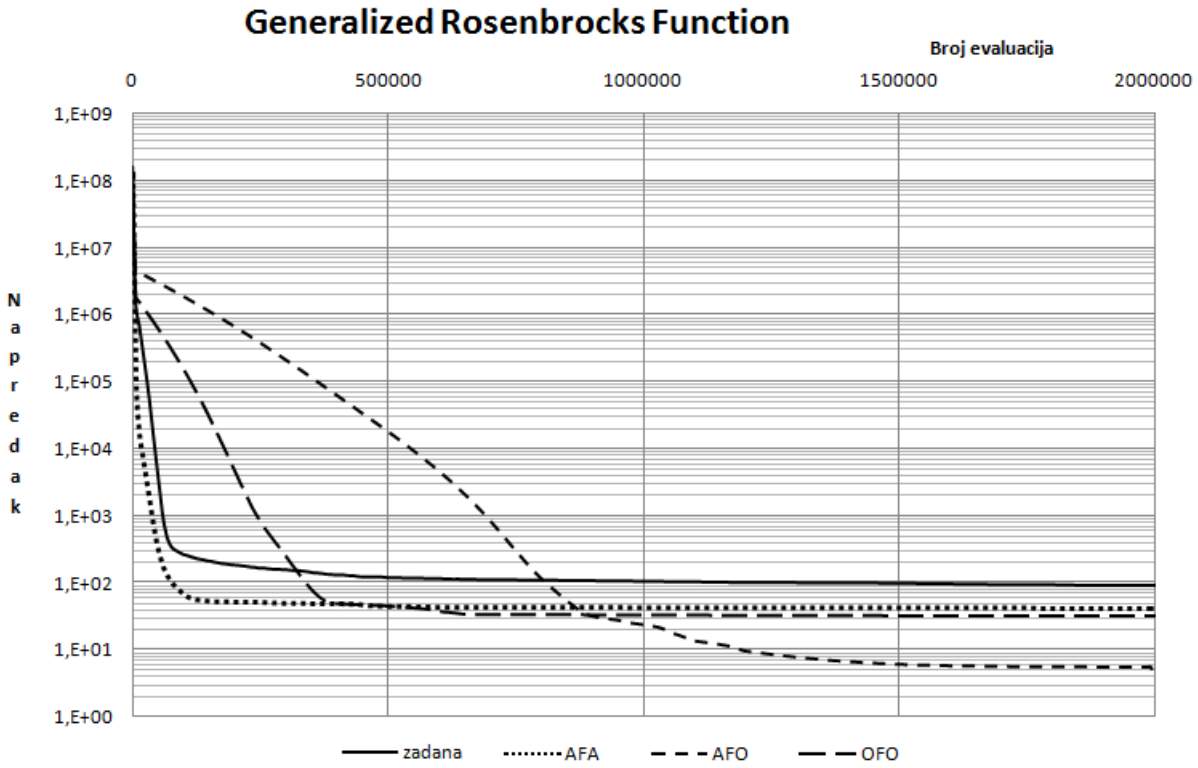
Slika 5.2: Najbolje rješenje prilikom optimiranja ispitne funkcije Schwefels problem 2.22



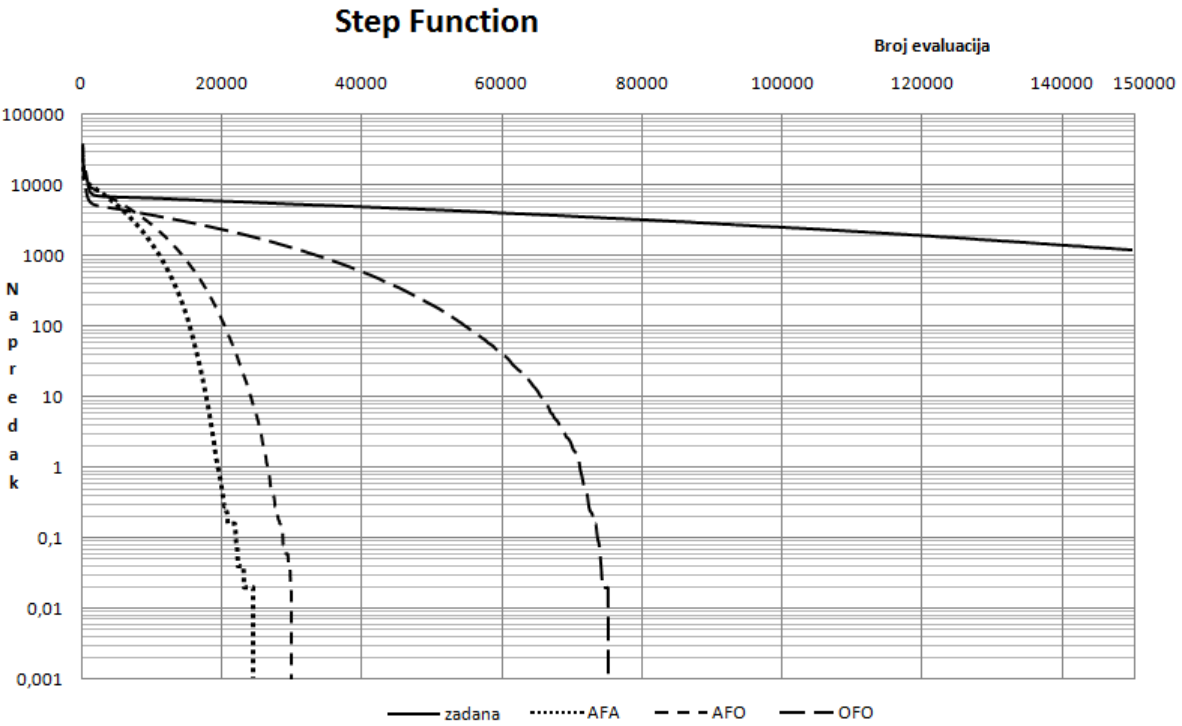
Slika 5.3: Najbolje rješenje prilikom optimiranja ispitne funkcije Schwefels problem 1.2



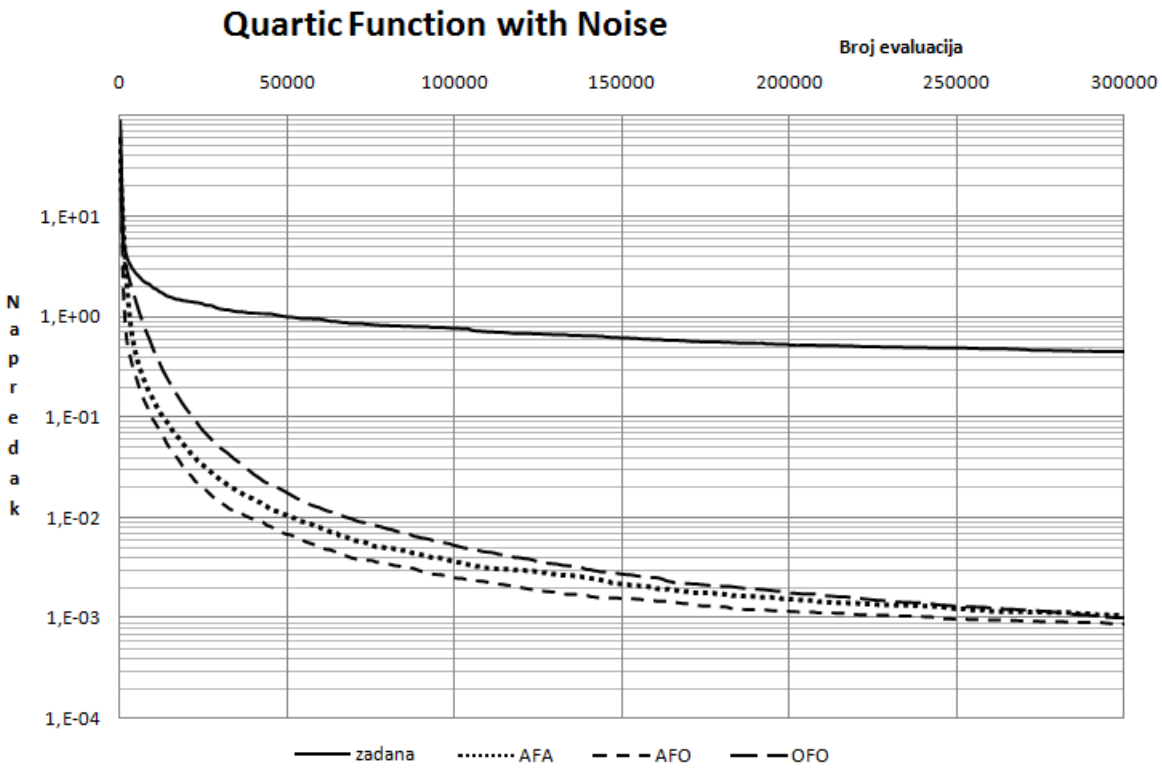
Slika 5.4: Najbolje rješenje prilikom optimiranja ispitne funkcije Schwefels problem 2.21



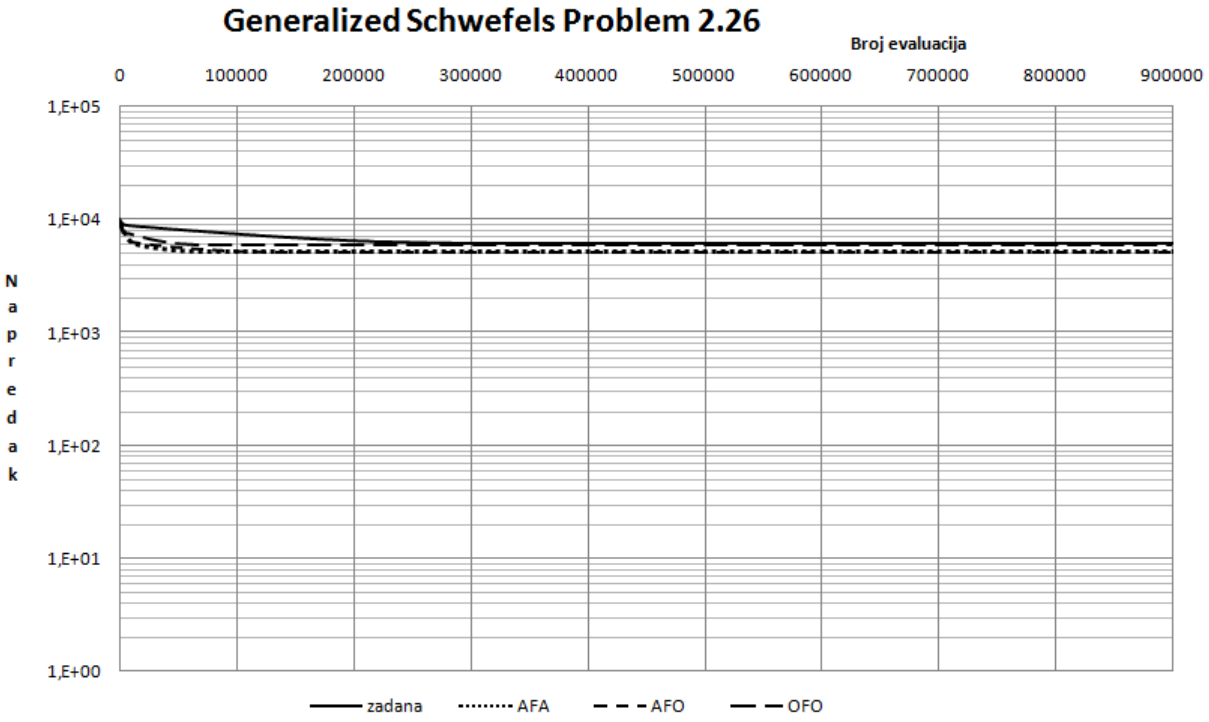
Slika 5.5: Najbolje rješenje prilikom optimiranja ispitne funkcije Generalized Rosenbrocks Function



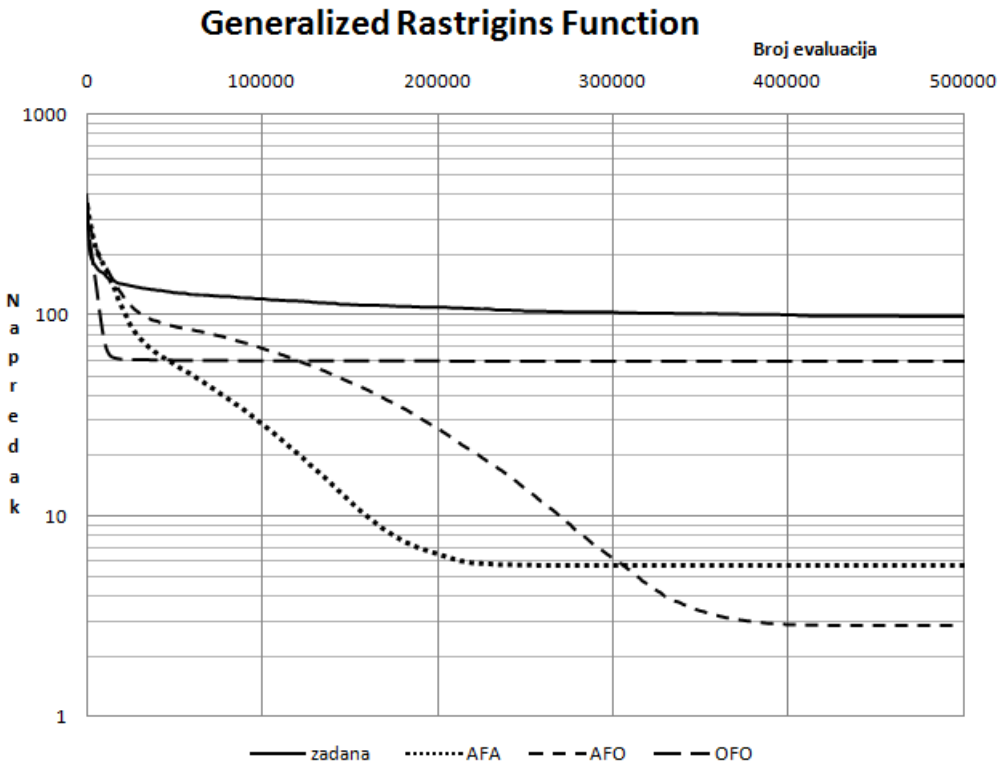
Slika 5.6: Najbolje rješenje prilikom optimiranja ispitne funkcije Step Function



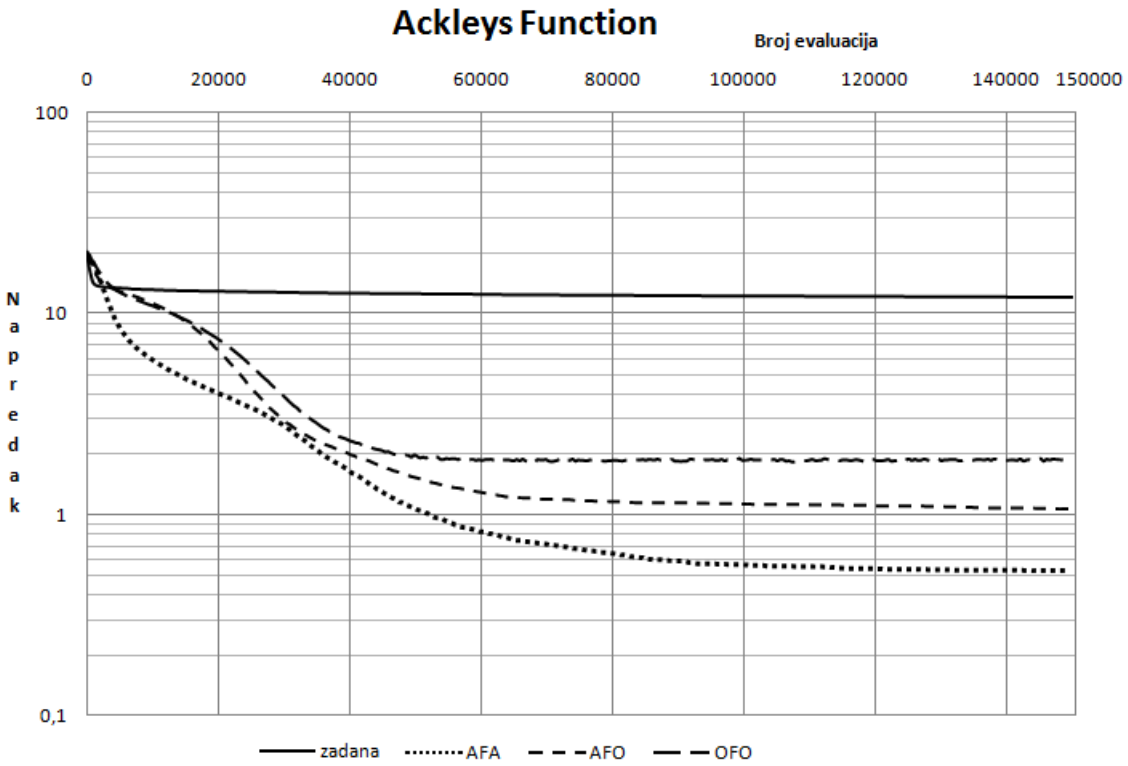
Slika 5.7: Najbolje rješenje prilikom optimiranja ispitne funkcije Quartic Function with Noise



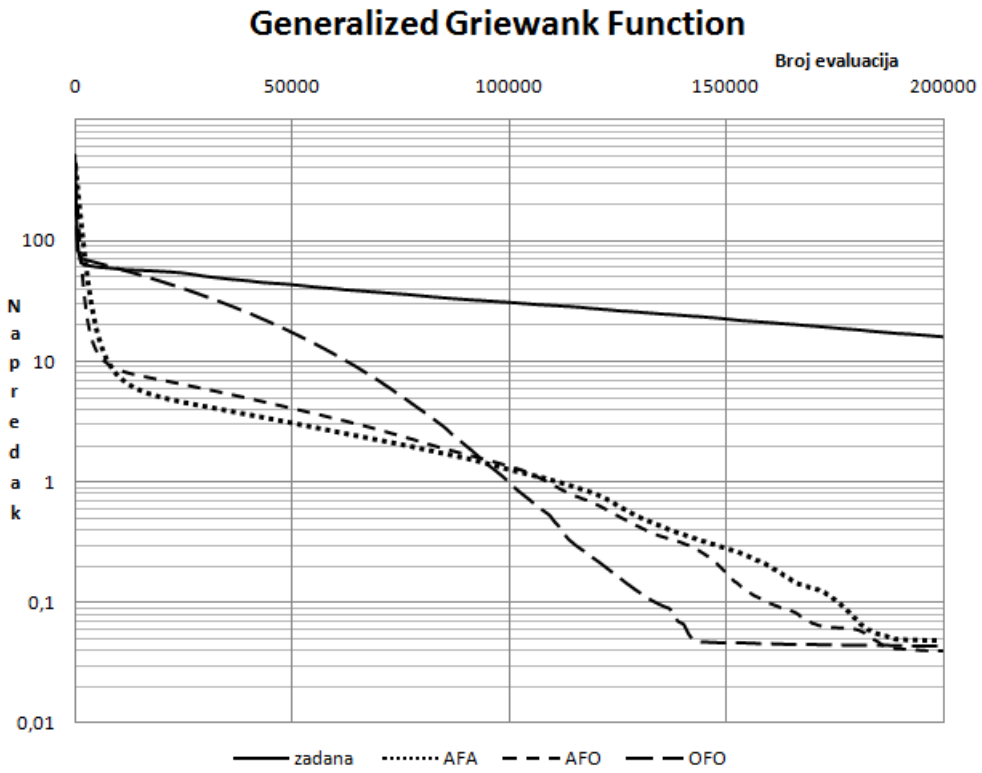
Slika 5.8: Najbolje rješenje prilikom optimiranja ispitne funkcije Generalized Schwefels Problem 2.26



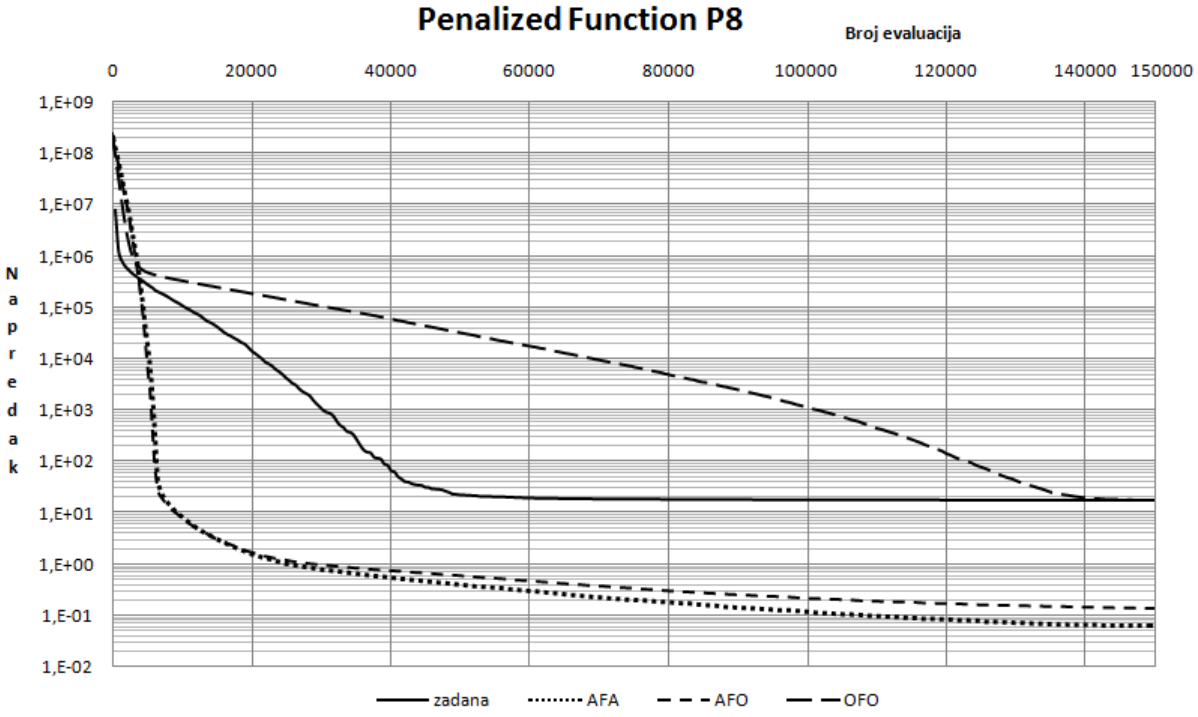
Slika 5.9: Najbolje rješenje prilikom optimiranja ispitne funkcije Generalized Rastrigins Function



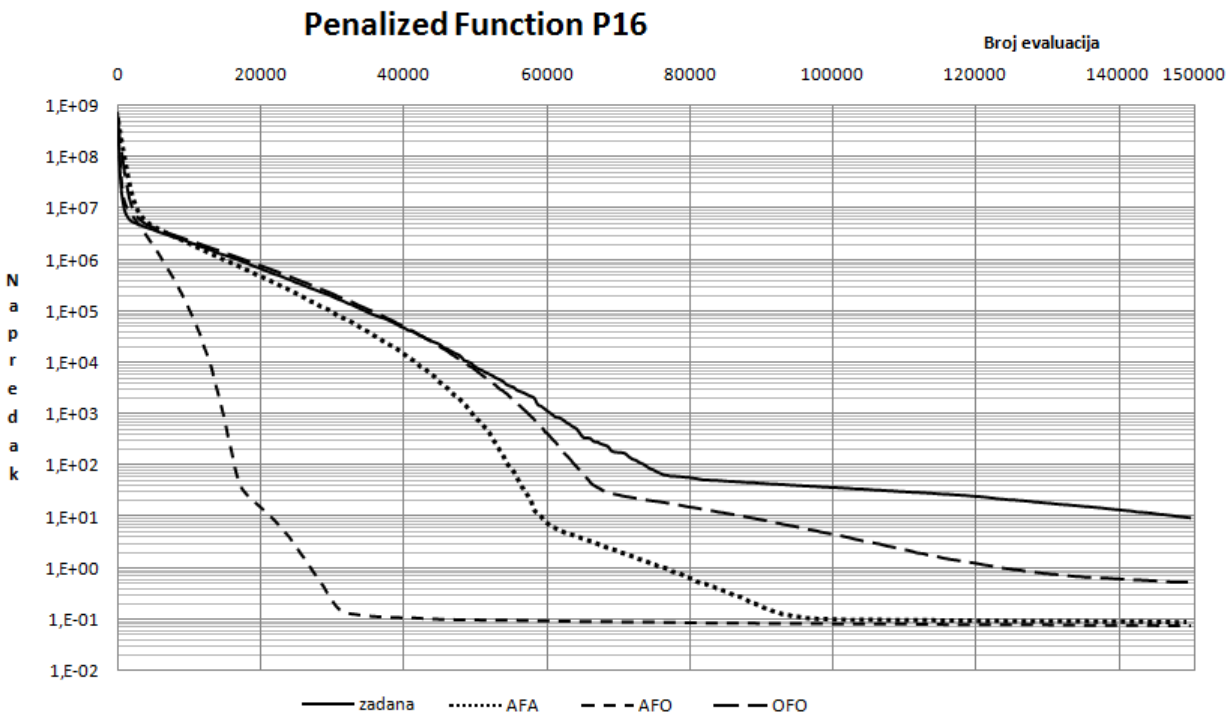
Slika 5.10: Najbolje rješenje prilikom optimiranja ispitne funkcije Ackleys Function



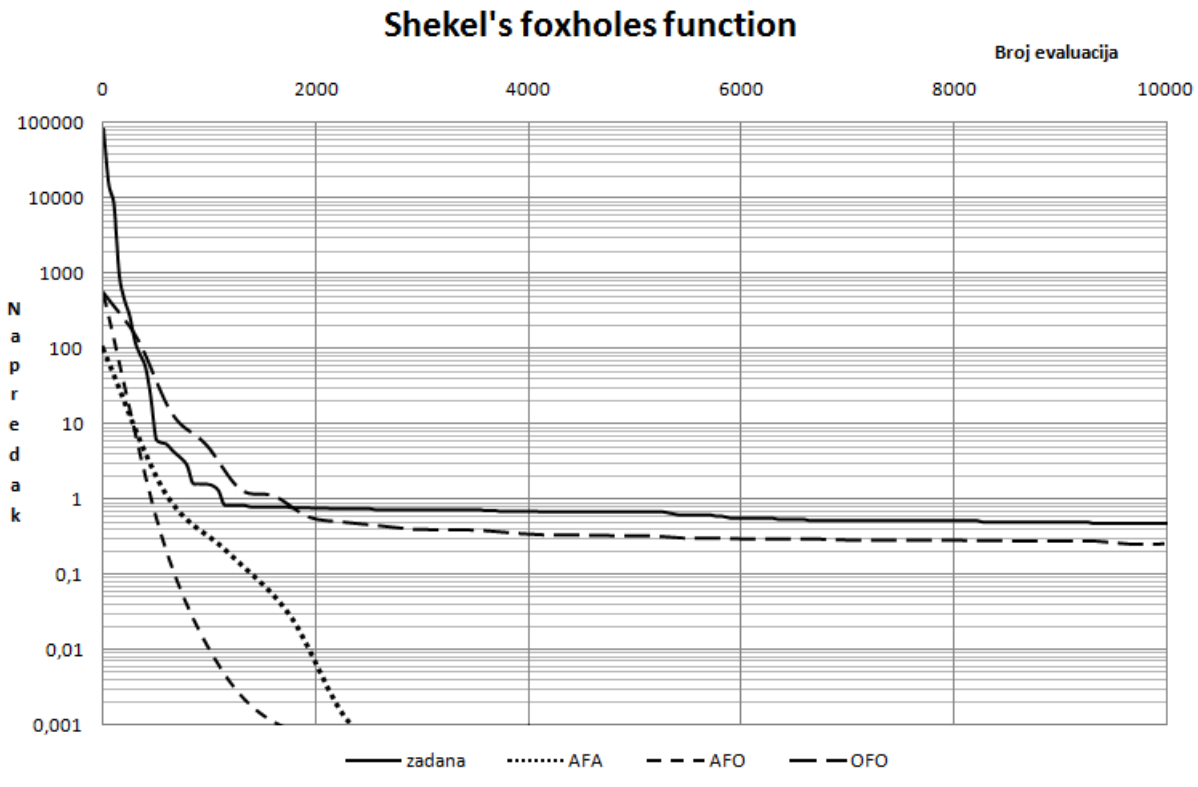
Slika 5.11: Najbolje rješenje prilikom optimiranja ispitne funkcije Generalized Griewank Function



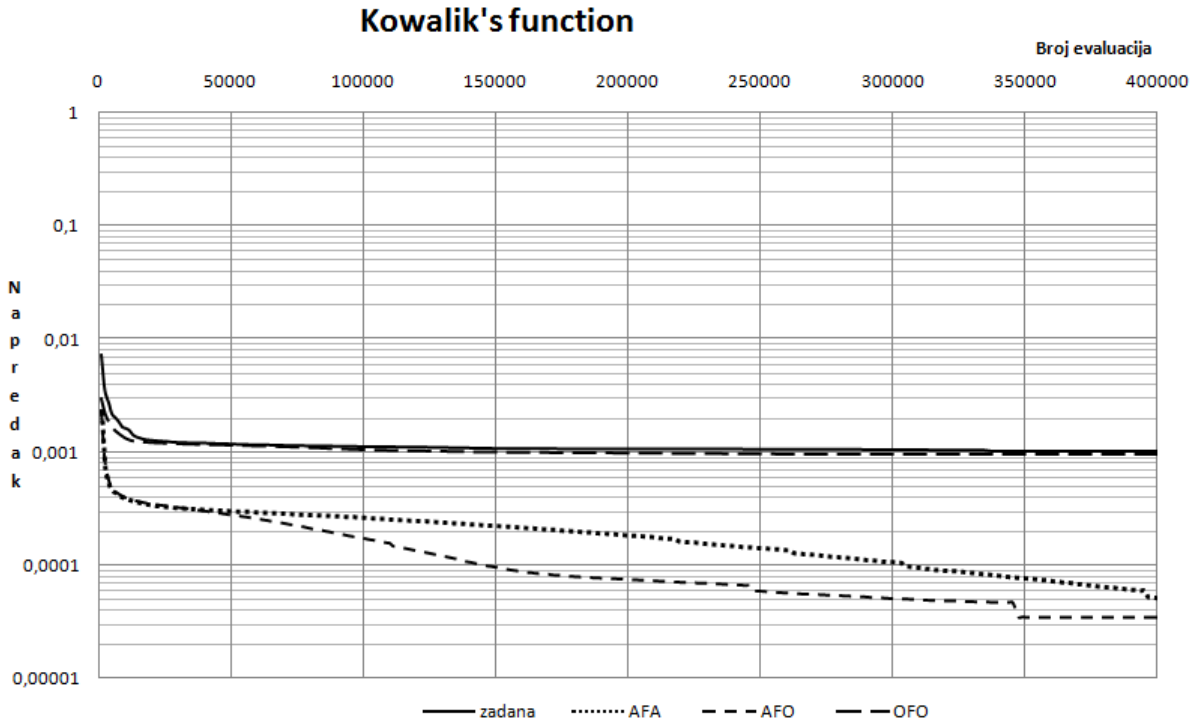
Slika 5.12: Najbolje rješenje prilikom optimiranja ispitne funkcije Penalized Function P8



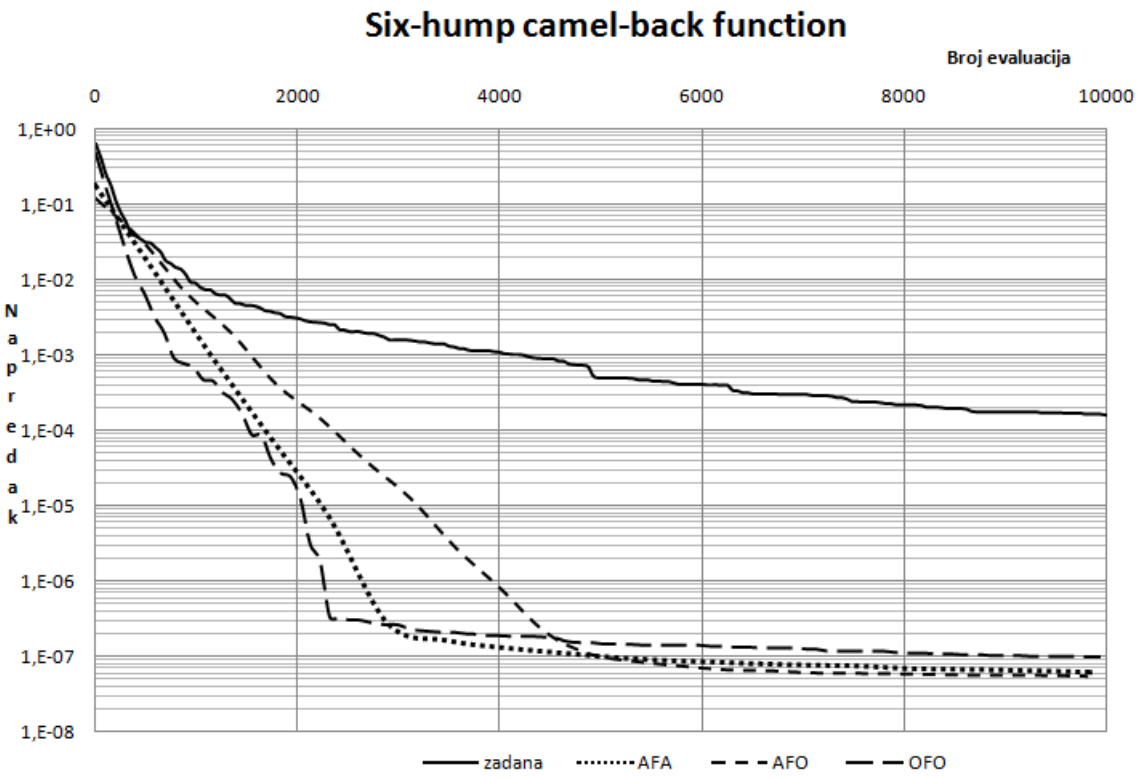
Slika 5.13: Najbolje rješenje prilikom optimiranja ispitne funkcije Penalized Function P16



Slika 5.14: Najbolje rješenje prilikom optimiranja ispitne funkcije Shekel's foxholes function



Slika 5.15: Najbolje rješenje prilikom optimiranja ispitne funkcije Kowalik's function



Slika 5.16: Najbolje rješenje prilikom optimiranja ispitne funkcije Six-hump camel-back function

5.3. Ispitivanje ponovljivosti

Svrha ovog ispitivanja je utvrđivanje koji je način odabira konfiguracije parametara iz domene bolji, slučajan ili način opisan u poglavlju 4.2 koji će se za svrhu ovog ispitivanja nazivati deterministički načinom. No pošto oba načina prema formuli (4.2) teže k pravom iznosu može se pretpostaviti da će oboje davati jednako dobre rezultate. No ono što možda neće biti isto kod obje metode je ponovljivost optimiranja. Jer slučajan odabir unosi dodatnu slučajnost u sustav koju deterministički način ne unosi.

Za test ponovljivosti se koristi AFA algoritam koji optimira parametre generacijskog genetskog algoritma koji optimizira 14. testnu funkciju. Razlog odabira 14. testne funkcije je što je relativno ravna što bi trebalo dodatno raspršiti smjerove optimiranja što će dovesti do većeg izražaja ponovljivosti. Konfiguracija AFA algoritma i domena optimiranja je ista kao u poglavlju5.2. Jedina razlika je vrijednostima $maxDubina$ koje su za svrhu ovog ispitivanja postavljene na vrijednost 2.

Promatra se mjera sličnosti optimiranja p, definirana kao 5.1

$$p\left(S_1, S_2\right) = \frac{|S_1 \cap S_2|}{|S_1| + |S_2|} \tag{5.1}$$

gdje su S_1i S_2 skupovi područja koja su vratila zasebna izvođenja AFA algoritma kao rezultat optimiranja. Optimiranje AFA algoritmom se izvela 10 puta sa slučajnim odabirom i 10 puta sa deterministčkim odabirom. Prosječna sličnost se sada definira kao 5.2.

$$T = \frac{2}{n * (n-1)} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} p\left(S_i, S_j\right) \tag{5.2}$$

Dok je u ovom slučaju $n = 10$ broj izvođenja. Što je prosječna sličnost optimiranja metode veća to je i veća ponovljivost.

Prosječne sličnosti za obje metode se nalaze u tablici 5.8.

Tablica 5.8: Prosječne sličnosti

	Slučajan odabir	Deterministički odabir
T	0.606	0.6069

Iako je vrijednost T veća kod deterministčkog odabira ta razlika je zanemariva. Temeljem rezultata ispitivanja se zaključuje da nema razlike u ponovljivosti između ta dva načina odabira.

Neuspjeh ovog testa ukazuje da bi se ipak trebali usporediti rezultati optimiranja obje metode, ali to ostaje za neki drugi rad.

6. Implementacija algoritama podijeli i usporedi za sustav ECF

6.1. Sustav ECF

ECF sustav je pisan u jeziku C++ te je namijenjen za programe evolucijskog računanja. Razvija se na FER-u od strane više studenata i profesora Domagoj Jakobović.

Sustav se konfigurira preko xml datoteke, te od korisnika još minimalno zahtjeva problem koji se optimira. Kompletni popis svih mogućnosti sustava ECF se može naći na službenoj stranici ECF-a http://gp.zemris.fer.hr/ecf/.

Sustav ECF koristi dvije eksterne biblioteke:

- boost C++ library http://www.boost.org/
- XML parser http://www.applied-mathematics.net/tools/xmlParser.html

Razvijeni PIU sustav također koristi iste dvije navedene biblioteke.

6.2. Sustav PIU

Sustav podijeli i usporedi (PIU) je namijenjen za optimizaciju parametara stohastičkih algoritama i implementira algoritme AFA, AFO i OFO. Stohastički algoritam kojemu se parametri optimiziraju se može specificirati kao funkcija ili se može iskoristiti ugrađeni parser koji je sposoban mijenjati xml datoteku. Takva promijenjena xml datoteka se može iskoristiti za konfiguriranje stohastičkog algoritma koji se testira. Takav je slučaj za sustav ECF. Sve defaultne postavke parsera sustava PIU su postavljene tako da se može s minimalnim naporom parsirati konfiguracijske datoteke sustava ECF. Način je prikazan na primjeru u poglavlju 6.3.2.

Sustav PIU se može konfigurirati direktno u kodu ili preko xml konfiguracijske datoteke, koja je detaljnije objašnjena u poglavlju 6.4.1. PIU je izveden u jeziku C++ te koristi dvije eksterne biblioteke navedene u poglavlju 6.1

6.2.1. Algoritmi podijeli i usporedi

Sustav ima realizirane algoritme AFA, AFO i OFO s par dodataka. Svi algoritmi imaju sposobnost optimiziranja po više kriterija za što se koristi dominantnost da bi se dobile Pareto fronte. S time da se može koristiti dva tipa dominantnosti:

- *slab* - ako je definirani algoritam bolji od drugog u barem jednoj karakteristici, a po svim ostalim karakteristikama su jednaki, onda prvi dominira nad drugim.
- *strog* - definirani algoritam mora biti bolji po svim karakteristikama od drugog da bi se proglasio dominantnijim.

Algoritmi AFA, AFO i OFO su izmijenjen tako da podržavaju specifikaciju maksimalnog broja iteracija algoritma. Algoritmi AFO i OFO podržavaju specifikaciju koraka u dubinu optimiranja. Odnosno kod algoritma AFO iz poglavlja 4.3.3 se u $splitPerActive$ metodi postavlja $maxDubina_t = korak$, a kod algoritma OFO iz poglavlja 4.3.2 se u $andDubina$ metodi postavlja

$$maxDubina_t = \min\left(korak, maxDubinaOrginal_t - putaOptimiziran_t\right)$$

I zadnji dodatak je kod algoritma OFO koji je spomenut na kraju poglavlja 4.3.2.

6.2.2. Opis sustava

Na slici 6.1 je prikazan pojednostavljen dijagram razreda PIU sustava.

PIU klasa je glavno sučelje za korištenje PIU sustava. Preko njega se sustav konfigurira te se provodi optimiranje. Konfigurira se prilikom stvaranja objekta klase *PIU* te pokretanjem AFA, AFO ili OFO metode se pokreće optimiranje s odgovarajućim algoritmom.

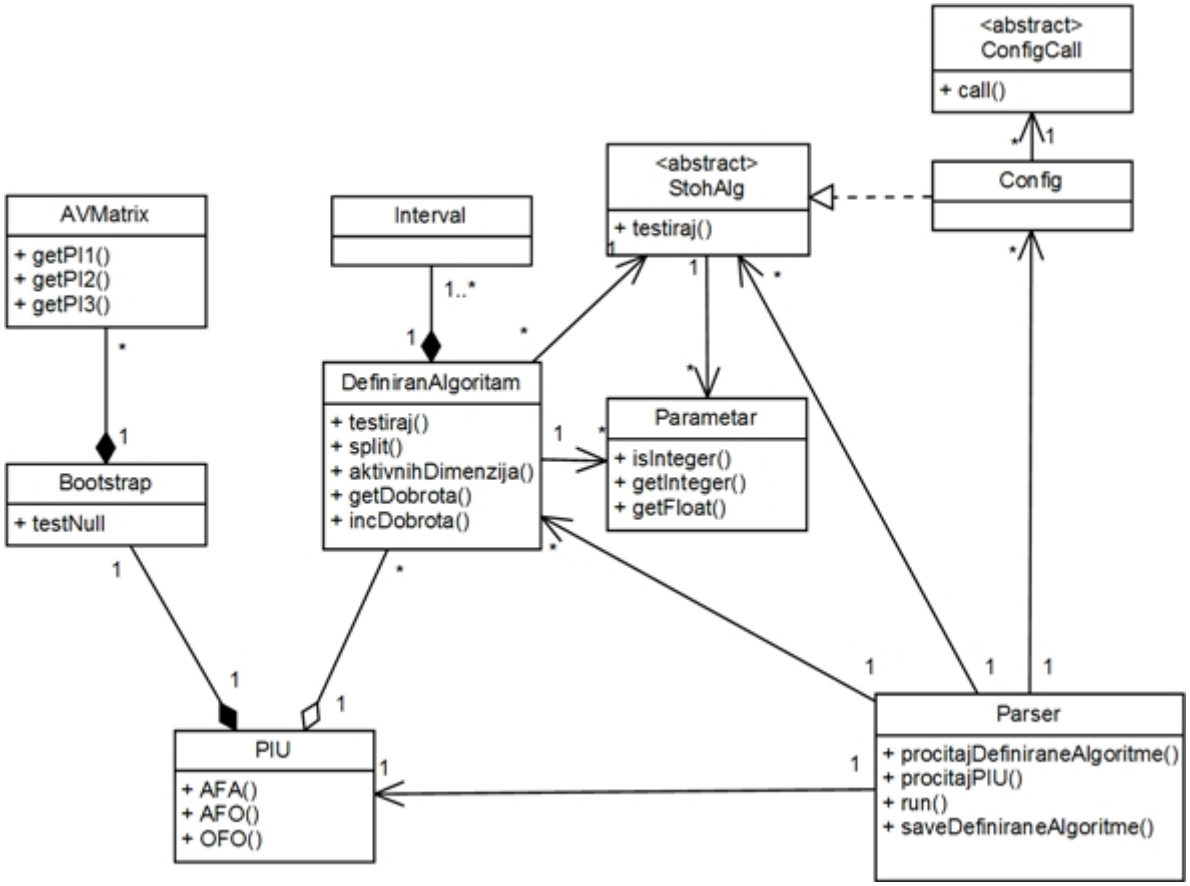
Boostrap klasa realizira bootstrap metodu koja koristi klasu *AVMatrix*, koja realizira Antihetic variates iz poglavlja 2.5, za smanjenje varijance.

Glavni entitet koji se optimizira je klasa *DefiniranAlgoritam* koja za opis svoje domene koristi klasu *Interval* te njen stohastički algoritam definiran preko apstraktne klase *StohAlg* s kojim se komunicira preko *Parametar* objekata.

Parser klasa ostvaruje mogućnost konfiguriranja i pokretanja PIU sustava preko *PIU* objekta, pomoću konfiguracijske xml datoteke. Kako?, je objašnjeno u poglavlju 6.4. Konfiguriranje drugih xml datoteka za svrhu testiranja se također obavlja u parseru. Takva izmijenjena xml datoteka se prosljeđuje kao parametar tijekom poziva metode *call()* razreda koji realizira apstraktnu klasu *ConfigCall*.

Ulazi u sustav su preko *PIU* klase i ili preko *Parsera*. Dok je potrebno definirati klasu koja realizira bilo *StohAlg* ili *ConfigCall* ovisno o željenom načinu upotrebe.

Sadašnji PIU sustav je moguće konfigurirati da podržava paralelno, odgođeno izvođenje stohastičkih algoritama. No zbog toga što još nije razvijena odgovarajuća platforma koja



Slika 6.1: UML dijagram razreda sustava PIU

može distribuirati izvođenje stohastičkih algoritama na više računala, se ne objašnjava ova funkcionalnost u ovom radu. Ali je ona zato komentirana u samom kodu.

6.3. Ispitivanje implementacije

Provedena su dva ispitivanja i jedan primjer. Prvo isptivanje ima definiran stohastički algoritam preko *StohAlg* klase, a drugi preko *ConfigCall* klase. U oba se slučajeva sustav konfigurira s xml datotekom. Sva tri primjera su priložena uz biblioteku PIU tako da se mogu isprobati.

6.3.1. Ostvarenje StohAlg klase

Svrha prvog ispita je provjerit je li sustav dobro optimizira te pokazati jednostavnost korištenja PIU sustava. Za tu svrhu se koristi deterministička funkcija 1. iz 5.1. Potrebna je baš deterministička funkcija kako bi se provjerilo i pokazalo svojstvo determinizma sustava.

Kod ima dvije datoteke:

- main.cpp , kod6.1
- TestAlg.cpp , kod 6.2

U C++ kodu 6.1 se parsira ulazna xml datoteka, izvlače se iz nje definirani algoritmi te se provodi optimiranje. Na kraju se rezultati optimiranja spremaju u novu xml datoteku koja se može ponovno koristiti za neku daljnju optimizaciju.

U C++ kodu 6.2 se realizira apstraktna klasa StohAlg koja ima samo jednu funkciju koja za parametre ima pointere na:

- $Parametar * parametri$- array parametara za koje se treba provesti testiranje.
- $double * value$- polje u koje se trebaju spremiti rezultati testiranja, dimenzija ovog polja se podrazumijeva kao broj kriterija po kojima se optimizira.
- $intdim$- dimenzija polja $parametri$.

Funkcija koja se optimira je Sphere Model koja ima minimum u točci $\{0, 0, 0, ..., 0\}$ te se očekuje da će rezultat optimiranja sadržavati tu konfiguraciju.

Definiran je definirani algoritam koja predstavlja StohAlg koji ostvaruje Sphere Model funkciju nad domenom

$$\{[-20, 20], [-200, 200], [-1, 1]\}$$

gdje su prve dvije domene nad cijelim brojevima, a zadnja je nad realnim brojevima. Rezultat optimiranja je

$$\{[0, 0], [0, 0], [0, -0.000122]\}$$

Algorithm 6.1 main.cpp

```
#include "piu.h"
#include "TestAlg.cpp"

using namespace std;

int main(int argc, char **argv){

/** Stvara se novi parser nad ulaznom xml datotekom */
Parser parser(argv[1]);

/** Učitavaju se definiraniAlgoritmi za problem TestAlg() */
TestniAlg test=new TestniAlg();
list<DefiniranAlgoritam*> *startList=parser.procitajDefiniraneAlgoritme(test);

/** Pokreće se testiranje*/
list<DefiniranAlgoritam*> *optimalni=parser.run(startList,NULL);

/** Spremaju se rezultati u izlaznu datoteku */
parser.saveDefiniraneAlgoritme(popis);

return 0;
}
```

Algorithm 6.2 TestAlg.cpp

```
#include "piu.h"

class TestniAlg: public StohAlg{

  public: void testiraj(Parametar *parametri,double *value,int dim){
    double v=0;
    for(int i=0;i<dim;i++){
      double x;
      if(parametri[i].isInteger())
        x=parametri[i].getInteger();
      else
      x=parametri[i].getFloat();
      v+=x*x;
    }

    delete[] parametri;
    *value=v;
  }
};
```

Algorithm 6.3 mainECF

```
int main(int argc, char **argv) {
  StateP state (new State);
  state->setEvalOp(new MyEvalOperator);
  state->initialize(argc, argv);
  state->run();
  return 0;
}
```

Iz toga proizlazi da je ispit zadovoljen. Za ispitivanje su korištena sva tri algoritma. Točna konfiguracija je prikazana na stranici 51. Objašnjenje xml konfiguracijske datoteke se može naći u poglavlju 6.4.1.

6.3.2. Ostvarenje ConfigCall klase

Svrha drugog ispita je ispitivanje spoja između PIU sustava i ECF sustava. Odnosno Config klase i kompletne Parser klase, koja je u prvom ispitivanju samo djelomično pokrivena.

Kako bi se pokazalo da je potreban minimalan dodatan kod za spajanje ECF i PIU sustava, upotrijebljen je prvi primjer iz tutoriala sa stranice http://gp.zemris.fer.hr/ecf/tutorial.html. Gdje je main C++ metoda 6.3.

Datoteke programa koje pripadaju PIU sustavu su:

- main.cpp
- Call.cpp 6.4

Gdje je main.cpp datoteka ista kao u prošlom primjeru s jedinom razlikom što se umjesto novog TestAlg objekta predaje novi Call objekt[1]. Dok je Call.cpp datoteka prikazana na stranici 52

Datoteka Call.cpp 6.4 se može iskoristiti kao predložak za spajanje ECF sustava sa PIU sustavom gdje je samo potrebno nadopisati kod vezan za upravljanje ECF sustavom na mjesto gdje je u Call.cpp kopirana main funkcija prikazana na stranici 50.

Kako se ne bi punilo rad s dodatne tri xml datoteke usmjeravam vas da pogledate primjer ConfigCallExample, koji dolazi uz PIU biblioteku. On je direktna realizacija ovog ispita.

Pošto se prilikom rada programa ne pojavljuje nikakva pogreška, sustav je prošao ovaj ispit.

[1] *parser.procitajDefiniraneAlgoritme(new Call());*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<PIU version="1.0" tip="StohAlg" izlazFile="RezultatS.xml">
  <Flags status="1" strog="1" />
  <Postavke
    brojKriterija="1"
    alfa="0.02"
    bootstrapUzorkovanja="3000"
    testovaPoDimenziji="3"
    dijecePoDimenziji="2" />
  <Algoritam name="allFORall">
    <Iteration value="1"/>
  </Algoritam>
  <Algoritam name="allFORone">
    <Iteration value="10"/>
    <Step value="1"/>
  </Algoritam>
  <Algoritam name="oneFORone"/>
  <DefiniraniAlgoritmi>
    <DefAlg>
      <Domena>
        <Parametar minValue="-20" maxValue="20" dataType="integer" dubina="16"
          start="1"/>
        <Parametar minValue="-200" maxValue="200" dataType="integer"
          dubina="20"
          start="0.1"/>
        <Parametar minValue="-1" maxValue="1" dataType="float" dubina="15"
          start="100"/>
      </Domena>
    </DefAlg>
  </DefiniraniAlgoritmi>
</PIU>
```

Slika 6.2: PIUStohAlg xml konfiguracijska datoteka

Algorithm 6.4 Call.cpp

```
#include "ECF.h"
#include "State.h"
#include "piu.h"
#include "FunctionMinEvalOp.h"

class Call:public ConfigCall{
//Prenose se extra parametri definirani prilikom poziva PIU sustava
private: char **argv;
private: int argc;

public: Call(int argc,char** argv){
this->argv=argv;
this->argc=argc;
}

public: void call(char* configFile,double* rezultat){
  //Postavljanje puta do konfiguracijske datoteke
  argv[1]=configFile;

  //Kod direktno prekopiran iz 6.3 main funkcije
  StateP state (new State);
  // set the evaluation operator
  state->setEvalOp(new FunctionMinEvalOp);
  state->initialize(argc, argv);
  state->run();

  //Spremanje rezultata
  *rezultat=state->getHoF()->getBest().front()->fitness->getValue();

  //Čišćenje
  argv[1]=NULL;
}

};
```

6.3.3. Primjer sa problemom putujučeg putnika

Problem trgovačkog putnika je problem pronalaska najkraćeg puta koji prolazi kroz sve gradove točno jedanput. Ostvaren je problem trgovačkog putnika s 29 gradova kojeg se rješava sa steady-state genetskim algoritmom uz pomoć sustava ECF. Pojedinci populacije su zapisani u obliku niza indeksa gradova koji onda predstavljaju put. Provodi se turnirska selekcija s mutacijom koja zamjenjuje poziciju dva indeksa unutar gena pojedinca. Duljina najkraćeg puta, korištenog problema trgovačkog putnika, iznosi 2020.

Optimiziraju se parametri genetskog algoritma:

1. veličina populacija $[20, 200]$
2. vjerojatnost mutacije $[0, 1]$
3. veličina turnira $[3, 20]$

Parametri su optimirani s tri algoritma AFA, AFO i OFO gdje se išlo do dubine 4 za veličinu turnira, 8 za vjerojatnost mutacije i 5 za veličinu populacije. Ostale postavke su više manje iste kao u poglavlju 5. Za više detalja se može pogledati kod ovog primjera koji dolazi uz biblioteku sustava PIU.

Rezultati optimiranja parametara su navedeni u tablici 6.1.

Tablica 6.1: Rezultati optimiranja parametara steady state genetskog algoritma za problem trgovačkog putnika

algoritam	br. područja	br. izvođenja	veličina pop.	vjerojatnost mut.	veličina turnira
AFA	1	2772	198	0.943	17
AFO	4	1564	192	0.88	10
OFO	4	828	131	0.99	17

U tablici 6.1 je za svaki korišteni algoritam navedeno koliko je definiranih algoritama vratio kao rezultat optimiranja te preporučene vrijednosti parametara jednog od vraćenih definiranih algoritama, ako ih je bilo više. Te također koliko se puta pokrenuo genetski algoritam tijekom optimiranja, broj izvođenja.

Performanse optimiranih parametara skupa sa zadanim parametrima, koji se nalaze u tablici 6.3, su ispitane ponavljanjem izvođenja genetskog algoritma 50 puta za isti problem trgovačkog putnika. Rezultati se nalaze u tablici 6.2.

Slika 6.3: Zadani parametri za problem trgovačkog putnika

veličina populacije	vjerojatnost mutacije	veličina turnira
30	0.3	3

Tablica 6.2: Rezultati ispitivanja optimiranih parametara

algoritam	očekivanje	std. devijacija
AFA	2039.2	22.4
AFO	2040.42	17.86
OFO	2048	27.737
zadano	2086.3	40.586

Genetski algoritam pokazuje bolja svojstva s optimiranim parametrima nego sa zadanim parametrima.

6.4. Upute korištenja

Testiranje u PIU sustavu se može provesti preko realizacije dvije apstraktne klase:

- *StohAlg*
- *ConfigCall*

StohAlg apstraktna klasa je za općenitije slučajeve. Klasa se sastoji od jedne metode koja za ulazne parametre ima polje parametara za koje se treba provesti testiranje, pointer na memorijsku lokaciju na koju se treba spremiti rezultate testiranja i dimenziju to je broj parametara u danom polju. Primjer optimiranja s ovom klasom se može vidjeti u primjeru StohAlgExample koji dolazi uz PIU biblioteku. Koji se također opisuje u 6.3.1.

ConfigCall apstraktna klasa je specijalizirana za komunikaciju s drugim sustavima[2] preko xml datoteke. Klasa se sastoji od jedne metode koja ima za ulaz "put\ime" nove izmijenjene xml datoteke i pointer na memorijsku lokaciju na koju se trebaju spremiti rezultati testiranja. Primjer optimiranja s ovom klasom se može vidjeti u primjeru *ConfigCallExample* koji dolazi uz PIU biblioteku. Koji se također opisuje u 6.3.2.

Konfiguracija PIU sustava se može obaviti na dva načina:

- hard kodiranje
- xml konfiguriranje

Sve metode u PIU sustavu su komentirane[3] tako da se neće ići u detalje konfiguriranja PIU sustava hard kodiranjem nego će se detaljno opisati pristup preko xml konfiguracijske datoteke.

[2] npr. ECF

[3] Upozorenje, pravopis loš

6.4.1. XML konfiguracija

Primjer xml konfiguracijske datoteke je na stranici 51.

PIU je glavni node.

Atributi:

tip = "StohAlg" ili "ConfigCall", određuje za koje sučelje je datoteka.

izlazFile = ime datoteke u koju će se spremiti rezultati u obliku nove konfiguracijske datoteke.

Podnodovi:

Flags

Atributi:

status = 1 ili 0, gdje 1 označava uključeno prikazivanje statusa sustava u komandnoj liniji.

strog = 1 ili 0, gdje je 1 označava strog režim koji se tiče definiranja dominantnosti jednog definiranog algoritma nad drugim u slučaju više kriterijskog optimiranja.

Postavke Globalne postavke PIU sustava.

Atributi:

brojKriterija = n, $n > 0$, označava po koliko kriterija se provodi optimiranje.

alfa = f, $f\epsilon < 0, 0.5 >$ zahtijevana signifikantnost prilikom bootstrap odluke.

bootstrapUzorkovanja = n, $n > 0$ broj reuzorkovanja kod bootstrapa.

testovaPoDimenziji =n, $n > 1$ broj testiranja po dimenziji, parametru.

djecePoDimenziji = n, $n > 1$ broj nove djece po dimenziji.

Algoritam Ovaj se node može ponavljati jedan za drugim s istim ili različitim algoritmima i podnodovima. Gdje njihov poredak u xml datoteci će također biti i njihov poredak u izvođenju jedan za drugim.

Atributi:

name = "allFORall", "allFORone", "oneFORone". Označava koji se algoritam koristi.

Podnodovi:

Iteration - koji ima jedan atribut *value* $= n$, $n > 0$ gdje je n broj iteracija algoritma.

Step - koji ima jedan atribut *value* $= n$, $n > 0$ gdje je n korak u dubinu algoritma. Ovaj podnode nema učinka kada se radi o "allFORall" algoritmu.

DefiniraniAlgoritmi Može imati jedan ili više pod nodova DefAlg. Svaka DefAlg struktura označava jedan početni definirani algoritam. Konfiguracija ovog dijela se razlikuje između različitih tipova konfiguracijske datoteke.

ConfigCall-DefAlg

Atribut:

configFile = "put\ime", put i i ime konfiguracijske datoteke koja se konfigurira za pripadajući definirani algoritam.

Podnode:

Domena može imati jedan ili više podnodova Parametar. Gdje svaki podnode Parametar definira jedan parametar za optimizaciju.

Parametar-atribute:

dataType = "integer" ili "float", označava kojeg je podatkovnog tipa parametar.

minValue $= f_{donja}$, donja granica pretraživanja parametra. (uključivo)

maxValue $= f_{gornja}$, gornja granica pretraživanja parametra. (uključivo)

start $= n$, vrijednost koja se koristi kod allFORone i oneFORone algoritama kao početna vrijednost prije nego što se provede bilo kakva promjena domene parametra. Također će ovaj parametar uvijek biti postavljen u izlaznoj datoteci gdje tada označava preporučenu vrijednost parametra[4].

dubina $= n$, maksimalan broj optimiranja parametra, odnosno maksimalna dubina do kojeg će algoritmi otići za taj parametar prilikom optimiranja.

Parametar-podnode ima podnode Source.

Source-atribute:

[4]Ovaj atribut je opcionalan, ali ako ga ima jedan parametar u domeni onda ga moraju imati svi parametri u toj domeni

path = "s", gdje je s put, u konfiguracijskoj datoteci od definiranog algoritma navedena kod configFile, od root noda do noda koji sadrži node "Entry" koji ima atribut "key". Nodovi su odvojeni '.' .

tag = "s", s je string za koji će se tražiti poklapanje s vrijednosti "key" kod "Entry". Kod noda s kojim se ima poklapanje se tekst noda zamjenjuje s odgovarajućim podatkom.

StogAlg-DefAlg

Podnode:

Domena može imati jedan ili više podnodova Parametar. Gdje svaki podnode Parametar definira jedan parametar za optimizaciju.

Parametar-atribute:

dataType = "integer" ili "float", označava kojeg je podatkovnog tipa parametar.

minValue = f_{donja}, donja granica pretraživanja parametra. (uključivo)

maxValue = f_{gornja}, gornja granica pretraživanja parametra. (uključivo)

start = n, vrijednost koja se koristi kod allFORone i oneFORone algoritama kao početna vrijednost prije nego što se provede bilo kakva promjena domene parametra. Također će ovaj parametar uvijek biti postavljen u izlaznoj datoteci gdje tada označava preporučenu vrijednost parametra[5].

dubina = n, maksimalan broj optimiranja parametra, odnosno maksimalna dubina do kojeg će algoritmi otići za taj parametar prilikom optimiranja.

[5]Ovaj atribut je opcionalan, ali ako ga ima jedan parametar u domeni onda ga moraju imati svi parametri u toj domeni

7. Zaključak

Algoritmi zasnovani na metodi podijeli i usporedi pokazuju značajan potencijal za optimiranje parametara stohastičkih algoritama iz područja Evolucijskog računanja. Pogotovo AFO algoritam. Potrebno je još puno ispitivanja kako bi se bolje razumjelo ponašanje tih algoritama. Te je potrebno provesti usporedbu s dosadašnjim metodama optimiranja parametara. Možda najbitnija mogućnost koju ima metoda podijeli i usporedi, koja je nažalost samo spomenuta u ovom radu, je što se mogu dizajnirati takva optimiranja koja će dovesti do određivanja najboljih algoritama tijekom samog procesa optimiranja. Razvijeni sustav Podijeli I Usporedi (PIU) se može poboljšati s izvedbom platforme za distribuirano testiranje. Također se pokazala potreba da se radi manje pretpostavki, a više eksperimenata. Ideje novog eksperimentalizma su samo djelomično uklopljene u ovom radu te bi to trebalo popraviti u budućim radovima.

Literatura

Thomas Bartz-Beielstein. *Experimental research in evolutionary computation*. Springer Berlin, 2006.

Radu V Craiu. Efficient bootstrap resampling. *Scientific Bulletin-University Politehnica Bucharest, Series A*, 63:3–10, 2001.

William C. Davidon. *VARIABLE METRIC METHOD FOR MINIMIZATION*. Society for Industrial and Applied Mathematics, 1991.

Bradley Efron i Robert Tibshirani. *An introduction to the bootstrap*, svezak 57. Chapman & Hall/CRC, 1993.

Ian Hacking. *Representing and intervening: Introductory topics in the philosophy of natural science*, svezak 355. Cambridge Univ Press, 1983.

Joel L. Horowitz. The bootstrap. U *In Handbook of Econometrics*, stranice 3159–3228. Elsevier Science, 2001.

Jeffrey C Lagarias, James A Reeds, Margaret H Wright, i Paul E Wright. Convergence properties of the nelder–mead simplex method in low dimensions. *SIAM Journal on Optimization*, 9(1):112–147, 1998.

Enno Mammen i Swagata Nandi. Bootstrap and resampling. U *Handbook of Computational Statistics*, stranice 499–527. Springer, 2012.

Deborah G Mayo. *Error and the growth of experimental knowledge*. University of Chicago Press, 1996.

Michael W Trosset. Taguchi and robust optimization. *Rapport technique, Rice University*, 1997.

K Paul Yoon i Ching-Lai Hwang. *Multiple attribute decision making: an introduction*. Broj 104. SAGE Publications, Incorporated, 1995.

Abdelhak M Zoubir i B Boashash. The bootstrap and its application in signal processing. *Signal Processing Magazine, IEEE*, 15(1):56–76, 1998.

Optimizacija parametara stohastičkih algoritama uporabom metode podijeli i usporedi

Sažetak

Osnovna potreba područja Evolucijskog računanja je usporedba dvaju algoritama. Kako bi se dobila ravnopravna usporedba potrebno je prvo ustanoviti najbolje vrijednosti parametara. Optimiranje parametara algoritama iz područja Evolucijskog računanja je težak problem koji se više ili manje uspješno rješava s metodama uvedenima iz drugih grana znanosti. Razvijena metoda podijeli i usporedi ima bitna svojstva koja omogućavaju algoritmima baziranima na toj metodi značajnu sposobnost optimiranja. Koja je demonstrirana na primjerima. AFA, AFO i OFO algoritmi koji se temelje na metodi podijeli i usporedi su realizirani u obliku C++ biblioteke kompatibilne s ECF sustavom.

Ključne riječi: Evolucijsko računanje, TOPSIS, Taguchi metoda, ECF, Podijeli i usporedi metoda

Optimization of parameters of stochastic algorithms using the method divide and compare

Abstract

The basic need of the area of Evolutionary computation is to compare two algorithms. In order to achieve an equal comparison it is necessary to determine the best values of parameters. Parameter optimization of algorithms in the field of Evolutionary computation is a difficult problem that is more or less successfully solved with the methods introduced from other branches of science. Developed method divide and compare has important characteristics that allow the algorithms based on this method, a significant ability for optimization. Which is demonstrated by examples. AFA, AFO and OFO algorithms that are based on the method of divide and compare are realized in the form of C + + library which is compatible with the ECF system.

Keywords: Evolutionary computation, TOPSIS, Taguchi method, ECF, Divide and compare method