Usporedba sinkronog i asinkronog obnavljanja populacije u algoritmu diferencijalne evolucije

Veselčić, Valentin

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni prijediplomski studij Računarstvo

Usporedba sinkronog i asinkronog obnavljanja populacije u algoritmu diferencijalne evolucije

Završni rad

Valentin Veselčić

Osijek, 2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z1P: Obrazac za ocjenu završnog rada na sveučilišnom prijediplomskom studiju

Ocjena završnog rada na sveučilišnom prijediplomskom studiju

Ime i prezime pristupnika:	Valentin Veselčić
Studij, smjer:	Sveučilišni prijediplomski studij Računarstvo
Mat. br. pristupnika, god.	R4730, 28.07.2021.
JMBAG:	0165092059
Mentor:	doc. dr. sc. Dražen Bajer
Sumentor:	
Sumentor iz tvrtke:	
Naslov završnog rada:	Usporedba sinkronog i asinkronog obnavljanja populacije u algoritmu diferencijalne evolucije
Znanstvena grana završnog rada:	**Umjetna inteligencija (zn. polje računarstvo)**
Zadatak završnog rada:	Opisati algoritam diferencijalne evolucije kao vrstu evolucijskih algoritama s naglaskom na numeričku optimizaciju. Posebno se osvrnuti na način obnavljanja populacije, odnosno odabir naredne generacije. Ugraditi dvije inačice algoritma diferencijale evolucije, gdje jedna vrši sinkrono obnavljanje populacije, a druga vrši asinkrono obnavljanje populacije. Eksperimentalno ispitati učinkovitost i ponašanje ugrađenih inačica algoritma na nekoliko standardnih testnih funkcija. Rezervirano za: Valentin Veselčić

Datum prijedloga ocjene završnog rada od strane mentora:	15.09.2024.
Prijedlog ocjene završnog rada od strane mentora:	Izvrstan (5)
Datum potvrde ocjene završnog rada od strane Odbora:	25.09.2024.
Ocjena završnog rada nakon obrane:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije završnog rada čime je pristupnik završio sveučilišni prijediplomski studij:	26.09.2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O IZVORNOSTI RADA

Osijek, 26.09.2024.

Ime i prezime Pristupnika:	Valentin Veselčić
Studij:	Sveučilišni prijediplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	R4730, 28.07.2021.
Turnitin podudaranje [%]:	8

Ovom izjavom izjavljujem da je rad pod nazivom**: Usporedba sinkronog i asinkronog obnavljanja populacije u algoritmu diferencijalne evolucije**

izrađen pod vodstvom mentora doc. dr. sc. Dražen Bajer

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis pristupnika:

SADRŽAJ

1. UVOD 1

1.1. Zadatak završnog rada 2

2. ALGORITAM DIFERENCIJALNE EVOLUCIJE 3

2.1. Struktura algoritma 3

2.2. Obnavljanje populacije 7

2.3. Sinkrono i asinkrono obnavljanja populacije u literaturi 8

3. OSTVARENO PROGRAMSKO RJEŠENJE 10

3.1. Način rada programskog rješenja 10

3.2. Prikaz i način upotrebe programskog rješenja 11

4. EKSPERIMENTALNA ANALIZA 13

4.1. Postavke eksperimenta 14

4.2. Rezultati 15

5. ZAKLJUČAK 23

LITERATURA .....

SAŽETAK .....

ABSTRACT .....

ŽIVOTOPIS .....

PRILOZI .....

1. UVOD

Problem kontinuirane optimizacije (engl. *continuous optimisation*) je problem koji se često pojavljuje u različitim inženjerskim zadacima, kao i problemima prisutnim kod strojnog učenja. Ti problemi zahtijevaju pronalazak rješenja unutar kontinuiranog prostora, odnosno prostora definiranog nad realnim skupom gdje se broj vrijednosti unutar nekog zadanog intervala smatra neprebrojivim. Prvi korak u procesu optimizacije takvog problema je definiranje funkcije cilja (engl. *objective function*). Takve funkcije često nemaju derivaciju ili su isprekidane što otežava klasičan pristup optimizacije poput matematičkog jer u nekim slučajevima funkcija cilja i nije poznata u analitičkom obliku, kao što je to slučaj u mnogim simulacijama zbog čega se takvi problemi nazivaju i problemi crne kutije (engl. *black-box optimization*).

Jedan od načina koji se pokazao vrlo uspješnim pri optimizaciji ovakvih problema je korištenjem evolucijskih algoritama (engl. *evolutionary algorithms,* EAs). EAs imaju značajne prednosti nad klasičnim iterativnim algoritmima za numeričku optimizaciju jer ne zahtijevaju informacije o derivacijama funkcije, te su robusniji u radu sa kompliciranim i nepravilnim funkcijama cilja. Ipak nisu bez nedostataka, osjetljivost performansi na postavke parametara je prisutna, te mogu se pokazati lošijim izborom u vidu spore konvergencije i zahtjeva nad računalnim resursima za funkcije koje imaju derivaciju i blag nagib. Također zbog potrebe za velikim brojem procjena funkcije cilja često znaju biti vremenski zahtjevni. Za postupak optimizacije koriste informacije iz populacije koja se sastoji od potencijalnih rješenja. Jedan od značajnijih predstavnika EAs je algoritam diferencijalne evolucije (engl. *differential evolution,* DE)

Algoritam DE je inicijalno stvoren sa namjernom rješavanja problema kontinuirane optimizacije. DE se ističe od ostalih EAs po svom načinu provođenja postupka mutacije, radi čega je ovaj element često tema razmatranja pri predstavljanju novih inačica algoritma. Uz mutaciju, način podešavanja parametara algoritma koji imaju direktan utjecaj na performanse algoritma je također često razmatran, dok je način obnavljanja populacije algoritma često zapostavljen u literaturi.

U standardnoj implementaciji algoritma, kao i u značajnom broju inačica u literaturi koristi se sinkrono obnavljanje populacije. Kod ovakvog načina obnavljanja populacije postoji točno definirana granica između članova populacije (vektora) trenutne i sljedeće generacije i ne dolazi do njihovog međudjelovanja. Nasuprot sinkronoj postoji i asinkrona implementacija algoritma gdje nema podjele vektora prema generacijama, što dovodi do njihovog međudjelovanja pri provođenju postupka optimizacije. Također dolazi do manje prostorne složenosti i bržeg reagiranja na promjene jer se pri obnavljanju populacije jedan od vektora koji se razmatra odmah odbacuje a

drugi dodjeljuje sljedećoj generaciji. Ove karakteristike ga ipak čine privlačnim za daljnje proučavanje.

U drugom poglavlju detaljno je opisan algoritam diferencijalne evolucije, s fokusom na postupak obnavljanja populacije. Također je dan pregled zastupljenosti oba načina obnavljanja populacije u literaturi. Treće poglavlje se sastoji od opisa programskog rješenja, njegovih ulaza, izlaza, te način upotrebe. Četvrto poglavlje predstavlja prikaz provedene eksperimentalne analize korištenjem programskog rješenja nad obje inačice algoritma. Eksperimentalna analiza je provedena na deset testnih funkcija pri dvije različite dimenzionalnosti i tri postavke parametra veličine populacije algoritma DE, sa rezultatima prikazanim tablicama i grafovima.

1.1. Zadatak završnog rada

Opisati algoritam diferencijalne evolucije kao vrstu evolucijskih algoritama s naglaskom na numeričku optimizaciju. Posebno se osvrnuti na način obnavljanja populacije, odnosno odabir naredne generacije. Ugraditi dvije inačice algoritma diferencijale evolucije, gdje jedna vrši sinkrono obnavljanje populacije, a druga vrši asinkrono obnavljanje populacije. Eksperimentalno ispitati učinkovitost i ponašanje ugrađenih inačica algoritma na nekoliko standardnih testnih funkcija.

2. ALGORITAM DIFERENCIJALNE EVOLUCIJE

Problem kontinuirane optimizacije je definiran kao problem u kojemu svaka od varijabli problema ima domenu u skupu $\mathbb{R}$, odnosno $x_i \in \mathbb{R}$, $\forall\, i = 1, \dots, n_x$. Pojam optimizacije predstavlja traženje rješenja za ovakav problem, koji tada predstavlja funkciju cilja (engl. *objective function*) za koju je potrebno pronaći njen maksimum ili minimum pri definiranim ograničenjima [1]. Često se definira bar ograničenje u vidu donje i gornje granice prostora pretrage. Moguće je se često susresti sa stvarnim problemima kontinuirane optimizacije kod kojih je ovakvo ograničenje prisutno, primjerice problem minimiziranja veličine rešetkastih struktura [2], problem raspodjele aktivne i reaktivne električne energije [3] optimizacija modela prometnog toka [4], optimizacija parametara modela u svrhu učenja kvalitetne reanimacije [5].

Upotreba evolucijskih algoritama je jedan od načina optimizacije kontinuiranih problema. Evolucijski algoritmi su grupa stohastičkih optimizacijskih metoda zasnovanih na modelu stvarnih evolucijskih procesa, gdje se svaka jedinka prilagođava svojoj okolini kako bi preživjela i tada se njene karakteristike dalje nasljeđuju.

Diferencijalna evolucija je jedan od evolucijskih algoritama inicijalno stvoren za rješavanje problema kontinuirane optimizacije [6]–[8], te se pokazao kao učinkovit za rješavanje istih. Posjeduje iste elemente kao i ostali evolucijski algoritmi, populaciju jedinki, način procjene kvalitete jedinki, proces selekcije i proces reprodukcije po kojemu se i ističe.

Standardna varijanta algoritma je kroz godine modificirana i nove strategije su predstavljene koje najčešće utječu na način kroz koji se proces mutacije provodi. U većini njih zadržava se standardni, odnosno sinkroni način obnavljanja populacije. Međutim istražene su i druge metode obnavljanja populacije, te postoje i asinkrone inačice algoritma, ali su znatno manje zastupljene i istražene u odnosu na one sa sinkronim.

2.1. Struktura algoritma

Diferencijalna evolucija (DE) je evolucijski algoritam predstavljen od strane Storne-a i Price-a 1997. godine, originalno stvoren sa namjerom primjene za probleme kontinuirane optimizacije [6]–[8]. Glavna razlika kojom se DE ističe od ostalih evolucijskih algoritama je to da za smjer daljnje pretrage DE koristi razlike, odnosno udaljenosti između jedinki trenutne populacije. Također novonastala jedinka se uspoređuje samo sa svojim roditeljem za odlučivanje koja će biti član sljedeće generacije, tzv. pohlepna shema (engl. *greedy scheme*), što često nije slučaj kod ostalih evolucijskih algoritama [1], [9].

DE se pokazao se kao vrlo kvalitetan način provođenja globalne optimizacije i upotrijebljen je na velikom rasponu takvih problema [2]–[5]. Metoda kojom algoritam izvršava postupak optimizacije sastoji se od četiri osnovna koraka: inicijalizacija populacije, mutacija, križanje i selekcija. Algoritam se sastoji od populacije jedinki gdje svaka jedinka predstavlja potencijalno rješenje, jedinke se iterativno poboljšavaju postupcima mutacije, križanja i selekcije. Jedinke iste generacije se međusobno kombiniraju u procesu mutacije i križanja nakon čega se stvara nova jedinka. Novonastale jedinke preuzimaju mjesto u sljedećoj generaciji populacije ako su kvalitetnije od prethodnih.

Populacija $P(g)$ algoritma DE se sastoji od broja jedinki definiranih parametrom NP, D-dimenzionalnih jedinki $x_j(g)$, koje se još nazivaju i vektori:

$$x_i(g) = \left(x_{i,j}(g)\right), i = 1, \dots, NP, j = 1, \dots, D \ , \tag{2.1}$$

gdje g predstavlja indeks generacije, i indeks jedinke i j indeks dimenzije.

Inicijalizacija početne populacije se najčešće izvršava postupkom biranja slučajne vrijednosti unutar zadanog prostora pretrage za svaku dimenziju svih jedinki populacije $P(g)$ veličine NP. Početna populacija može utjecati na performanse algoritma u sljedećim generacijama, slučajnim postupkom inicijalizacije se osigurava da su jedinke jednoliko raspoređene, odnosno raznolike, čime se omogućuje kvalitetno istraživanje prostora pretrage i smanjuje rizik od rane stagnacije. Iako su istraženi i drugi načini inicijalizacije [10], ovaj ostaje najčešće korišten.

Mutacija je proces koji se smatra ključnim za DE i po kojemu je DE dobio i ime [11]. Za svaki vektor cilja (engl. *target vector*) $x_i(g)$ populacije $P(g)$ slučajno se bira bazni vektor (engl. *base vektor*) x_{i_1} i upotrjebljuje se vektor razlike dva slučajno odabrana vektora x_{i_2}, x_{i_3} iz populacije i to takva da:

$$i \ \neq i_1 \neq i_2 \ \neq i_3 \tag{2.2}$$

Tako izabrani vektori koriste se za stvaranje mutant vektora (engl. *mutant vector*) $u_i(g)$ prema formuli:

$$u_i(g) = x_{i_1}(g) + \ F * \left(x_{i_2}(g) - x_{i_3}(g)\right), \tag{2.3}$$

gdje parametar $F \in (0, \infty)$ predstavlja faktor skaliranja i kontrolira snagu utjecaja diferencijalne varijacije. što u principu znači da za vrijednosti faktora skaliranja $F = 0$ mutant vektor će biti u potpunosti jednak baznom vektoru, dok sa većim vrijednostima utjecaj vektora razlike na bazni

vektor se povećava i on se više modificira. Faktor skaliranja F utječe na to koliko će se krajnji mutant vektor razlikovati od baznog vektora.

Pri procesu mutacije mutant vektor $u_i(g)$ može se stvoriti van prethodno definiranih granica prostora pretrage, te se treba izvršiti korekcija. Postoje dvije različite generalne tehnike korekcije, kaznom (engl. *penalty*) i resetiranjem (engl. *resetting*) [8], [9]. Kod prve tehnike novonastalom vektoru se dodjeljuje "kazna" pomoću nekog dodatnog kriterija pri procjeni kvalitete vektora, čime se nastoji ostvariti nepovoljna vrijednost procjene kvalitete kako bi se odvratilo od biranja tog vektora pri procesu selekcije. Kod resetiranja vektor se direktno modificira i njegove vrijednosti koje prekoračuju postavljene granice se mijenjaju na različite načine ovisno o tehnici.

Istraženo je više različitih implementacije DE, primarno sa različitim načinima provođenja procesa mutacije [12]. Radi ovoga različite implementacije moguće je razaznati prema notaciji za njihov bazni vektor. Za dvije najpoznatije implementacije, *rand/1* i *best/1*, to bi bilo:

$$u_i(g) = x_{rand}(g) + F * \left(x_{i_2}(g) - x_{i_3}(g)\right), \tag{2.4}$$

$$u_i(g) = x_{best}(g) + F * \left(x_{i_2}(g) - x_{i_3}(g)\right), \tag{2.5}$$

gdje *rand* predstavlja slučajno izabrani vektor iz populacije, a *best* najkvalitetniji. Postoje i implementacije u kojima se koristi više vektora razlike [1], [12], [13].

Proces križanja algoritma DE koristi kombinaciju mutant vektora $u_i(g)$ i vektora cilja $x_i(g)$ za stvaranje novog vektora $x_i'(g)$. Postupak biranja komponenti mutant vektora ili vektora cilja implementiran je na sljedeći način:

$$x'_{j,i}(g) = \begin{cases} u_{j,i}(g)\ ako\ (rand\ (0,1) \leq CR)\ ili\ (j = randbr(d)) \\ x_{i,j}(g)\ u\ suprotnom \end{cases},\ j = 1, \dots, D\ , \tag{2.6}$$

gdje $randbr(d)$ predstavlja slučajno izabrani indeks dimenzije iz skupa $\{1, 2, \dots, D\}$ pomoću kojega se osigurava da novi vektor sadrži bar jedan element mutant vektora, a $rand(0,1)$ predstavlja slučajno izabranu vrijednost zadanog intervala [0,1] iz skupa $\mathbb{R}$. Za proces križanja koristi se slučajno generirana vrijednosti iz intervala [0,1], koja u slučaju da je manja od vrijednosti parametra CR, dovodi do toga da će se komponenta mutant vektora dodijeliti novom vektoru, inače se novom vektoru dodjeljuje komponenta vektora cilja. Parametar CR predstavlja stopu križanja (engl. *crossover-rate*) čija je vrijednost u rasponu [0,1]. Parametrom CR direktno se utječe na količinu komponenti koje novonastali vektor nasljeđuje od roditelja (vektora cilja) ili mutant

vektora. Opisana metoda križanja naziva se binomnom (engl. *binomial*), ali postoji i eksponencijalna (engl. *exponential*) koja se rjeđe koristi [1], [8].

Utjecaj parametara $NP, F\ i\ CR$ na algoritam DE je značajan, što se može primijetiti i u performansama algoritma. Radi toga pojavljuje se potreba za njihovom kalibracijom u svrhu ostvarivanja zadovoljavajućih performansi, što je dovelo do razmatranja strategija za njihovo podešavanje [14]–[16].

Tijekom procesa selekcije bira se koji će od dva vektora (novonastalog vektora i roditelja) postati dio sljedeće generacije prema navedenom:

$$x_i(g+1) = \begin{cases} x_i'(g)\ ako\ f\left(x_i'(g)\right) \leq f\left(x_i(g)\right) \\ x_i(g)\ u\ suprotnom \end{cases}, \tag{2.7}$$

gdje f predstavlja funkciju cilja (engl. *objective function*), za čiju manju ostvarenu vrijednost sa jednim od vektora se odlučuje koji od njih će se prenijeti u sljedeću generaciju. Procesom selekcije omogućuje se stvaranje nove generacije čija kvaliteta neće biti lošija od prethodne. U većini implementacija proces selekcije se provodi tek nakon što su svi novi vektori trenutne generacije generirani, odnosno sinkrono, ali postoji i asinkroni način koji je manje zastupljen. Glavna razlika je u tome da, za razliku od sinkronog, kod asinkronog načina selekcija se provodi odmah kada je novi vektor stvoren. Radi toga ima mogućnost brže reakcije na promjene, novi vektori se odmah koriste u daljnjim procesima i ne postoji generacijski jaz kao što je slučaj kod sinkronog načina obnavljanja populacije.

Navedeni procesi algoritma DE se ponavljaju sve dok se ne ostvari uvjet završetka koji je definiran. Neki od mogućih uvjeta završetka su prekid kada je zadovoljavajuće rješenje pronađeno, prekid kada nema daljnjeg napretka odnosno došlo je do stagnacije, prekid nakon određenog broja generacija, prekid kada je dostignut maksimalni dozvoljeni broj evaluacija funkcije cilja [1], [8].

Radi različitih implementacija algoritma DE [12], [13] u literaturi je općeprihvaćena notacija DE / *x* / *y* / *z*, gdje *x* predstavlja način biranja baznog vektora, *y* broj korištenih vektora razlike i *z* korištenu metodu križanja. Na slici 2.1 prikazan je pseudokôd standardnog algoritma *DE/rand/1/bin*.

```
definiraj parametre NP,F,CR
inicijaliziraj brojač generacije g = 0
stvori početnu populaciju P(g) koja se sastoji od NP slučajno generiranih jedinki x(g)

sve dok uvjet završetka nije ostvaren radi:
      za svaki vektor cilja x_i(g) populacije P(g) radi:
            izaberi tri slučajna vektora x_i1 ≠ x_i2 ≠ x_i3 ≠ x_i iz populacije P(g) i stvori mutant vektor u_i(g) = x_i1 + F * (x_i2 − x_i3)
            izvrši korekciju u_i(g) po potrebi
            križanjem u_i(g) i x_i(g) stvori novi vektor x_i'(g)
            ako f(x_i'(g)) je bolje od f(x_i(g)) onda:
                  x_i(g + 1) = x_i'(g)
            u suprotnom:
                  x_i(g + 1) = x_i(g)
      g += 1
vrati najbolje ostvareno rješenje x_i(g)
```

Slika 2.1 *Pseudokôd algoritma DE/rand/1/bin*

2.2. Obnavljanje populacije

Pojam obnavljanja populacije kod algoritma DE se odnosi na stvaranje nove generacije, odnosno stvara se nova generacija koju čine vektori odabrani procesom selekcije, takvi da sveukupna kvaliteta nove generacije ne degradira u odnosu na prethodnu. Novu generaciju mogu sačinjavati i vektori prethodnih generacija uz uvjet da su bili bolji od svojih potomaka (engl. *offspring*).

U originalnom algoritmu DE provodi se sinkroni način obnavljanja populacije [6], [7], što znači da proces selekcije se provodi tek nakon što su generirani svi novi vektori za postojeće vektore. Ovako provedenim obnavljanjem populacije dovodi se do toga da potencijalno kvalitetniji novostvoreni vektori se ne koriste u procesima DE odmah u trenutku kada postanu dostupni, već se održavaju dvije zasebne populacije, čiji se pripadajući roditelji i njihovi potomci uspoređuju.

U nekim implementacijama DE predložen je i asinkroni način obnavljanja populacije, gdje se proces selekcije provodi odmah kada je novi vektor generiran [17], [18]. Kvalitetniji vektor se zadržava i uvrštava u sljedeću generaciju, a preostali vektor se odmah odbacuje. S obzirom da se novostvoreni vektor uspoređuje samo sa roditeljem, a ne sa cijelom populacijom i sama implementacija ovakvog načina obnavljanja populacije je relativno jednostavna. Asinkroni način obnavljanja populacije je također manje zahtjevan na računalne resurse radi svoje manje prostorne složenosti jer nije potrebno držati dvije različite populacije u memoriji radi kasnije usporedbe njihovih jedinki kao što je slučaj kod sinkronog načina obnavljanja populacije. Pri asinkronom obnavljanju populacije kako se uklanja standardni ciklus izvršavanja procesa DE tako i pojam

generacije postaje zanemariv s obzirom da se trenutna generacija u bilo kojem trenutku izvođenja algoritma može sastojati od jedinki različitih generacija koje bi u sinkronoj implementaciji pripadale tek sljedećoj generaciji radi kasnijeg procesa selekcije do kojeg dolazi tek kada su generirane sve nove jedinke trenutne generacije.

Sličnu strategiju odabira jedinki sljedeće generacije koristi i podgrupa genetskih algoritama, eliminacijski genetski algoritmi (engl. *steady state genetic algorithms)* gdje do selekcije dolazi odmah kada je novi vektor stvoren. Ovakav pristup je dinamičniji i također nosi manju prostornu složenost u usporedbi sa generacijskim genetskim algoritmima (engl. *generational genetic algorithms*) gdje se proces selekcije provodi tek kada su svi novi vektori stvoreni [1], [9]. Ipak razlikuju se od većine DE implementacija u načinu provođenja selekcije, koji se kod DE provodi direktnom usporedbom roditelja i novonastale jedinke, kod GA se koristi više različitih strategija poput turnirske selekcije (engl. *tournament selection*) gdje se najlošija jedinka iz proizvoljne grupe jedinki zamjenjuje novonastalom, zamjena najlošijeg je još jedna od opcija, gdje se najlošija jedinka populacije zamjenjuje sa novom jedinkom, ili pak kao i kod DE provodi se direktna usporedba roditelja i nove jedinke.

2.3. Sinkrono i asinkrono obnavljanja populacije u literaturi

Iako su brojne inačice algoritma DE predložene u literaturi u većini njih radi se o standardnom, sinkronom načinu obnavljanja populacije. Znatno češće razmatraju se modifikacije ostalih elemenata algoritma, poput podešavanja parametara ili mutacije u odnosu na sam način obnavljanja populacije.

Jedan od brojnih primjera gdje se koristi sinkrono obnavljanje populacije je [15], gdje Draa i drugi predlažu varijantu algoritma diferencijalne evolucije OCSinDE sa dodatnim mehanizmima u svrhu poboljšavanja jedne prethodno predstavljene DE inačice. Iako su u ovom i radovima [14], [16], [19] razmatrana poboljšanja u vidu podešavanja parametara F i CR, način obnavljanja populacije ostaje onaj standardni. Slično tome, radovi [11], [19] u kojima su proučene inačice sa modificiranim procesom mutacije također koriste sinkrono obnavljanje populacije.

S druge strane ipak postoji nekolicina radova koji razmatraju asinkroni način obnavljanja populacije. Qing [17] u svom radu navodi način primjene asinkronog DE u problemima elektromagnetskog inverznog raspršenja. Eksperimentalnom analizom provodi usporedbu performansi standardnog DE algoritma i asinkrone inačice (DDE). Primjećuje superiornost DDE u vidu brže konvergencije i robusnosti za različite postavke eksperimentalne analize, kao i manje zahtjeve po pitanju računalnih resursa radi same prirode takvog načina obnavljanja populacije.

Peng i drugi [18] su razmotrili asinkronu inačicu algoritma DE/*neighbor*/1, koja također koristi i adaptivnu strategiju za podešavanje parametra CR. Za bazni vektor se bira slučajni vektor iz susjedstva vektora cilja. U eksperimentalnoj analizi uspoređuju predloženi algoritam sa standardnim DE/*rand*/1 i DE/*best*/1 gdje se pokazuje boljim u vidu performansi. Također predloženu varijantu uspoređuju sa nekoliko kompleksnijih varijanti DE algoritma, gdje se ponovno pokazao kao kvalitetniji. Ipak povoljni rezultati se ne mogu pripisati samo dinamičnoj prirodi obnavljanja populacije algoritma uzimajući u obzir ostale mehanizme koji su i detaljnije proučeni u radu.

U radu [4], Strofylas i drugi koriste sinkronu i asinkronu inačicu paralelnog algoritma DE, za koju se umjesto računalno zahtjevnog izračunavanja vrijednosti funkcije cilja koriste modeli koji određuju njenu približnu vrijednost. Algoritam upotrjebljuju u svrhu optimizacije modela prometnog toka koji koristi stvarne podatke. Zaključuju da su obje inačice algoritma došle do jednakih rezultata i kao takve su dobar izbor za optimizaciju problema, s tim da asinkrona brže konvergira i bolje upotrjebljuje računalne resurse.

3. OSTVARENO PROGRAMSKO RJEŠENJE

Programski jezik Python je korišten u ostvarivanju programskog rješenja. Programsko rješenje se sastoji od algoritma DE/*rand*/1/bin pri dvije različite implementacije načina obnavljanja populacije (sinkrono i asinkrono obnavljanje populacije).

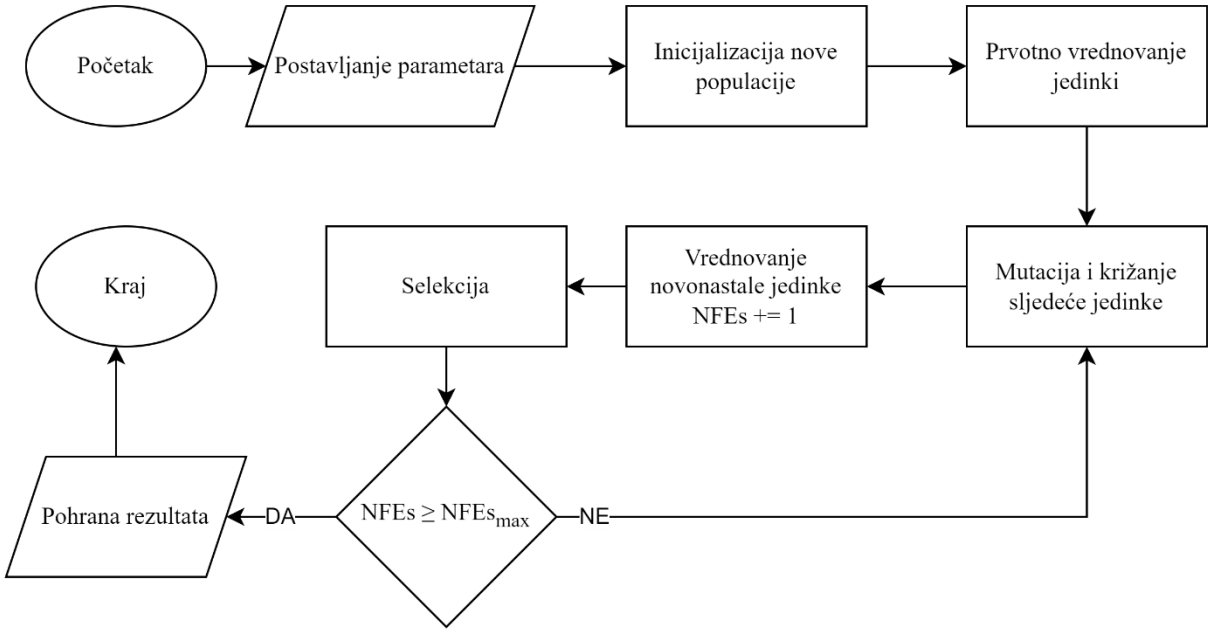
Omogućeno je spremanje podataka u CSV (engl. *comma separated values*) formatu tijekom samog izvršavanja programa, kao i automatsko imenovanje datoteka za jednostavnije snalaženje. Od biblioteka korištene su samo standardne biblioteke, *csv* za pohranjivanje podataka i *random* zbog svojih funkcija koje daju slučajne uniformne vrijednosti unutar zadanog intervala.

3.1. Način rada programskog rješenja

Ulaz programskog rješenja su parametri algoritma, veličina populacije NP, faktor skaliranja F, stopa križanja CR kao i broj puta koliko će se algoritmi izvršiti, dimenzionalnost, maksimalni dozvoljeni broj evaluacija funkcije cilja $NFES_{max}$, funkcija cilja i njena pripadajuća gornja i donja granica prostora pretrage. U ostvarenoj automatiziranoj okolini programskog rješenja omogućen je unos svih navedenih varijabli putem konzole osim funkcije cilja i gornje i donje granice koje su predefinirane. Moguć je višestruk unos varijabli dimenzionalnosti i veličine populacije.

Po završetku unosa varijabli započinje se kombinacija svake funkcije sa svakom od zadanih dimenzionalnosti i veličina populacija. Svaka od jedinstvenih kombinacija se koristi pri izvršavanju zadanog broja ponavljanja algoritama.

Implementacija samog algoritma DE započinje sa inicijalizacijom početne populacije sa prethodno zadanim brojem jedinki nakon čega se izvršava vrednovanje svih jedinki. Započinje se proces mutacije i korekcije u slučaju da je mutirani vektor izvan granica, križanja i selekcije sve dok zadani $NFES_{max}$ budžet nije iscrpljen. Nakon iskorištavanja budžeta prikupljeni rezultati se spremaju. Ovaj proces koji se ponavlja za svako ponavljanje algoritma radi prikupljanja većeg broja podataka je prikazan na dijagramu toka na slici 3.1.



Slika 3.1. *Dijagram toka programskog rješenja na visokoj razini za jedno izvođenje*

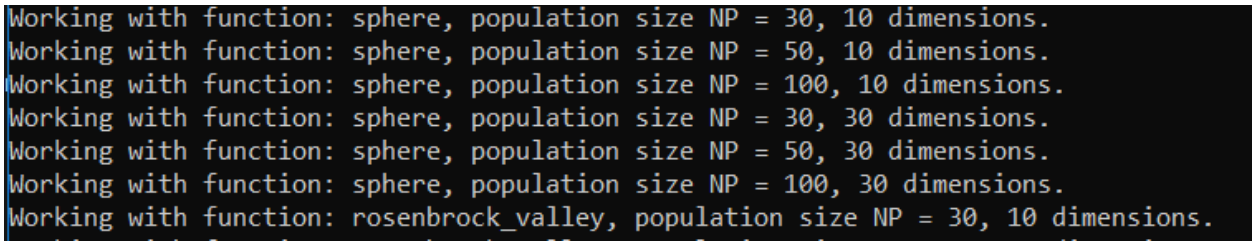
3.2. Prikaz i način upotrebe programskog rješenja

Korištenje programskog rješenja započinje sa unošenjem broja ponavljanja, dimenzije, veličine populacije, faktora skaliranja, stope križanja i broja dozvoljenih evaluacija funkcije. U slučaju da vrijednosti nisu unesene u ispravnom formatu (odvojene zarezom ili praznim mjestom) od korisnika se zahtijeva ponovni unos. Primjer unosa postavki prikazan je na slici 3.2.

```
Input number of iterations: 30
Input dimension/s: 10, 30
Input population size/s: 30, 50, 100
Input scaling factor F: 0.5
Input crossover rate CR: 0.9
Input number of allowed function evaluations: 100000
```

Slika 3.2. *Unos postavki programskog rješenja*

Nakon postavljanja ulaza započinje se izvođenje ponavljanja. U konzoli je omogućen prikaz funkcije koja se trenutno optimizira sa obje implementacije algoritma, kao i trenutna veličina populacije i dimenzija. Primjer ispisa tijekom izvođenja programskog rješenja je prikazan na slici 3.3.



Slika 3.3 *Ispis tijekom izvođenja programskog rješenja*

Nakon izvršavanja programskog rješenja podaci su dostupni unutar *csv* datoteka koje se nalaze u zajedničkoj mapi. Datoteke su podijeljene po tipu implementacije DE algoritma, funkcijama i dimenzionalnostima korištenim na tim funkcijama, po čemu su datoteke i imenovane. Podaci u stupcima predstavljaju nabolje rješenje u određenom trenutku izvođenja algoritma (od lijeva pa nadesno, 1%, 10%, 20%, … 100% ukupnog budžeta $NFEs_{max}$). Podaci u redcima predstavljaju pojedino ponavljanje. Prije svakog skupa navedena je i korištena konfiguracija. Primjer izlaza programskog rješenja prikazan je na slici 3.4.

1	NP = 30, F = 0.5, CR = 0.9										
2	1.642E+02	5.572E-16	7.704E-35	4.941E-55	8.088E-75	5.849E-94	6.132E-113	2.542E-113	2.541E-113	2.541E-113	2.541E-113
3	1.340E+02	1.037E-15	7.182E-34	2.563E-52	1.203E-71	4.073E-91	4.915E-110	1.215E-129	7.848E-150	2.770E-167	9.977E-178
4	8.733E+01	2.334E-15	3.391E-32	4.668E-50	4.350E-68	4.347E-87	2.758E-106	1.354E-125	6.596E-144	1.435E-163	8.341E-183
5	1.079E+02	3.320E-15	5.446E-34	2.022E-52	4.386E-71	1.599E-92	1.054E-112	2.484E-132	9.402E-151	1.921E-170	3.265E-189
6	2.626E+02	1.436E-16	1.240E-33	4.930E-53	2.719E-71	2.887E-87	1.329E-88	1.323E-88	1.323E-88	1.323E-88	1.323E-88
7	1.218E+02	5.109E-16	4.600E-35	6.975E-54	2.415E-72	5.127E-92	2.388E-111	5.781E-131	5.810E-150	1.271E-168	1.739E-185
8	3.263E+02	3.076E-11	5.800E-31	3.587E-50	4.685E-68	1.344E-87	4.118E-106	3.255E-125	1.827E-144	6.146E-164	5.397E-184
9	1.569E+02	7.299E-15	1.855E-15	1.855E-15	1.855E-15	1.855E-15	1.855E-15	1.855E-15	1.855E-15	1.855E-15	1.855E-15
10	2.581E+02	2.481E-14	2.963E-32	3.363E-50	1.828E-68	7.639E-82	1.269E-100	4.478E-120	2.993E-139	8.884E-160	3.176E-179
11	1.089E+02	1.894E-15	3.502E-36	6.990E-55	1.208E-73	1.242E-80	9.620E-82	8.992E-82	8.989E-82	8.989E-82	8.989E-82
12											
13	NP = 50, F = 0.5, CR = 0.9										
14	1165.6455	7.76E-07	3.26E-17	1.52E-27	4.85E-38	1.04E-48	2.81E-59	2.38E-69	2.02E-79	1.38E-89	1.00E-99
15	1279.0109	2.05E-06	5.89E-17	8.75E-28	8.37E-39	1.19E-48	3.23E-59	1.33E-69	1.87E-80	2.20E-91	9.13E-102
16	3083.9181	3.77E-06	1.10E-16	8.57E-28	3.29E-39	7.95E-50	8.30E-61	2.97E-71	6.49E-82	1.45E-91	1.89E-102
17	1044.8695	1.67E-07	4.31E-18	5.05E-29	1.50E-39	5.25E-50	6.34E-61	1.39E-71	2.48E-81	5.92E-92	2.97E-102

Slika 3.4. *Izlaz programskog rješenja*

4. EKSPERIMENTALNA ANALIZA

S ciljem usporedbe učinkovitosti i ponašanja implementacija algoritma DE pri sinkronom i asinkronom obnavljanju populacije provedena je eksperimentalna analiza. Iz skupa najčešće korištenih konfiguracija parametara u literaturi [14], faktor skaliranja, stopa križanja i veličina populacije, upotrijebljene su najčešće korištene. Dvije konstantne vrijednosti za faktor skaliranja F i stopu križanja CR, te skup od tri vrijednosti za veličinu populacije NP je upotrijebljen.

Često korištenih deset testnih funkcija [9], [20], [21] pri dvije različite dimenzionalnosti upotrijebljeno je za provedbu analize. Funkcije su navedene u tablici 4.1, gdje d predstavlja razmotrene dimenzionalnosti, $S \subseteq R^d$ prostor pretrage i f_{min} globalni minimum funkcije. Funkcije $f_1 - f_4$ su unimodalne, dok su funkcije $f_5 - f_{10}$ multimodalne sa velikim brojem lokalnih minimuma.

Unimodalne funkcije su povoljnije za usporedbu stopa konvergencije algoritama što čini taj podatak bitniji od same kvalitete krajnjeg rješenja. Nisu prisutni lokalni minimumi u kojima bi se algoritam mogao zaglaviti, što znači da se fokusira na što brže pronalaženje globalnog minimuma. Kod multimodalnih funkcija kvaliteta krajnjeg rješenja je bitniji podatak jer prikazuje sposobnost algoritma da izbjegne zaglavljivanje u nekvalitetnim lokalnim minimumima [20].

Korištenje različitih dimenzionalnosti daje uvid u robusnost algoritma, sa većim brojem dimenzija problem postaje kompleksniji. Prostor rješenja se povećava eksponencijalno [12], što znači da se za svaku dodatnu dimenziju problema dodaje nova varijabla. Vrijednost te varijable u kombinaciji s vrijednostima ostalih varijabli stvara novo potencijalno rješenje. Veći broj varijabli rezultira eksponencijalnim rastom broja mogućih kombinacija, algoritam mora pretražiti znatno veći prostor potencijalnih rješenja kako bi pronašao optimalno rješenje.

Tablica 4.1. *Testne funkcije korištene kao funkcije cilja za optimizaciju*

Testna funkcija	d	S	f_{min}
$f_1(x) = \sum_{i=1}^{D} x_i^2$	{10,30}	[-100, 100]^d	0
$f_2(x) = \sum_{i=1}^{D-1} (100\left(x_{i_{+_1}} - x_i^2\right)^2 + (x_i - 1)^2)$	{10,30}	[-10, 10]^d	0
$f_3(x) = \sum_{i=1}^{D} x_i^2 + (\sum_{i=1}^{D} 0.5 \cdot i \cdot x_i)^2 + (\sum_{i=1}^{D} 0.5 \cdot i \cdot x_i)^4$	{10,30}	[-10, 10]^d	0
$f_4(x) = \sum_{i=1}^{D} \lvert x_i \rvert^{i+1}$	{10,30}	[-10, 10]^d	0
$f_5(x) = -\sum_{i=1}^{D} x_i \sin\left(\sqrt{\lvert xi \rvert}\right) + 418.9828872724337 \cdot D$	{10,30}	[-500, 500]^d	0
$f_6(x) = \sum_{i=1}^{D} (x_i^2 - 10\cos(2\pi x_i)) + 10 \cdot D$	{10,30}	[-5.12, 5.12]^d	0
$f_7(x) = -20\exp(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}) - \exp\left(\frac{1}{D}\sum_{i=1}^{D} cos(2\pi x_i)\right) + e + 20$	{10,30}	[-32.768, 32.768]^d	0
$f_8(x) = \sum_{i=1}^{D} \lvert x_i \sin(x_i) + 0.1x_i \rvert$	{10,30}	[-10, 10]^d	0
$f_9(x) = \frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=i}^{D} \cos(\frac{x_i}{\sqrt{i}}) + 1$	{10,30}	[-100, 100]^d	0
$f_{10}(x) = 1 - cos\left(2\pi\sqrt{\sum_{i=1}^{D} x_i^2}\right) + 0.1\sqrt{\sum_{i=1}^{D} x_i^2}$	{10,30}	[-20, 20]^d	0

4.1. Postavke eksperimenta

Korišten je standardni algoritam DE (DE/*rand*/1/bin) u dvije implementacije, s jedinom razlikom u načinu obnavljanja populacije. Izvršeno je trideset ponavljanja za oba algoritma, za svaku od funkcija navedenih u tablici 4.1 pri $d = \{10, 30\}$. Svako ponavljanje izvršeno je sa novom, slučajno generiranom populacijom unutar zadanih granica. Za rukovanje ograničenjem granica iskorišten je jedan od najjednostavnijih načina, vektor se postavlja na gornju granicu ako je veći od nje, odnosno manju ako je manji.

Kao uvjet završetka izvođenja algoritma korišten je maksimalni dozvoljeni broj evaluacija funkcije cilja (engl. *number of function evaluations*), $NFEs_{max} = 10^5$. Tijekom izvođenja algoritma prikupljeni su podaci o najboljem rješenju u deset točaka i to pri 1%, 10%, 20%, 30%, …, 90% vrijednosti $NFEs_{max}$, kao i vrijednost nakon 100% $NFEs_{max}$. Naknadnom obradom rezultata podaci su agregirani kako bi se dobile prosječne vrijednosti za svaku od navedenih točaka 30 ponavljanja algoritma, te prosjek i pripadajuća standardna devijacija krajnje vrijednosti.

Za vrijednosti parametara faktora skaliranja F i stope križanja CR korištene su konstantne vrijednosti uzete iz skupa najčešćih vrijednosti parametara u literaturi [14], $F = 0.5$ $CR = 0.9$.

Za parametar veličine populacije NP korišten je skup vrijednosti {30, 50, 100}, s ciljem usporedbe performansi implementacija algoritma DE pri različitim veličinama populacije. Korištenjem više različitih vrijednosti veličine populacije moguće je proučiti kako broj jedinki utječe na učinkovitost pretraživanja u vidu stope konvergencije i kvalitete krajnjeg rješenja, ali i robusnost algoritma. Sa većim vrijednostima parametra NP zadani prostor pretrage moguće je bolje istražiti, dok sa manjom vrijednosti NP bolje se iskorištavaju i dalje poboljšavaju one jedinke koje su najbolje. Korištenjem različitih veličina populacije pri različitim načinima obnavljanja populacije moguće je proučiti ponašanje algoritma u vidu ravnoteže njegovih sposobnosti istraživanja i iskorištavanja. S obzirom da se uspoređuju načini obnavljanja populacije ispituju se različite postavke parametara NP koji direktno utječu na njenu veličinu.

4.2. Rezultati

Rezultati optimizacije prethodno navedenih testnih funkcija sa dvije različite implementacije obnavljanja populacije algoritma, asinkrono (DDE) i sinkrono (DE), prikazani su u tablici 4.2 i tablici 4.3. Analizirani podaci su grupirani u tablice po dimenzionalnosti. Najbolji rezultati su prikazani podebljano. U grafovima prikazani su prosjeci prikupljenih najboljih rješenja tijekom izvođenja, na slici 4.1 za vrijednosti dimenzionalnosti $d = 10$, a na slici 4.2 pri $d = 30$.

Pri $\mathrm{d} = 10$ i $NP = 30$ rezultati u vidu stope konvergencije i prosječne kvalitete rješenja sa DDE su bolji na 2 funkcije (f_1 i f_8), lošiji na također 2 (f_4 i f_{10}) i neznatno bolji od DE na 5 od preostalih 6 funkcija sa sličnim rezultatima. Na funkcijama sa sličnim, te neznatno boljim ili lošijim rezultatima, algoritmi imaju identične krivulje stope konvergencije, a prosjeci rješenja su u istim redovima veličine, s blagom prednošću jednog algoritma nad drugim. Ipak, razlike u kvaliteti rješenja nisu značajne.

DDE za vrijednosti veličine populacije $NP = 50$ se pokazuje boljim na 4 funkcije (f_1, f_2, f_3, f_8), dok je DE bolji samo na 1 (f_4), slični podaci su vidljivi za funkcije $f_5, f_6, f_7, f_9, f_{10}$ s tim da za 3 funkcije (f_5, f_6, f_{10}) rezultati su neznatno bolji za DE, dok za preostale 2 DDE je kvalitetniji. Za $NP = 100$ DE je bolji izbor samo na f_5 i to nedaleko od DDE koji je bolji na 5 funkcija. Slični rezultati su za funkcije f_6, f_7, f_9, f_{10} od kojih je samo za f_9 DDE bolja opcija.

Tablica 4.2 *Prikaz rezultata DDE i DE pri* $d = 10$ *za deset testnih funkcija*

F	$NP = 30$		$NP = 50$		$NP = 100$	
	DDE	DE	DDE	DE	DDE	DE
	avg. ± std dev	avg. ± std dev	avg. ± std dev	avg. ± std dev	avg. ± std dev	avg. ± std dev
f_1	**8.795E-29 ± 4.74E-28**	8.420E-15 ± 4.53E-14	**3.282E-101 ± 8.63E-101**	3.408E-83 ± 1.06E-82	**1.201E-43 ± 1.50E-43**	8.543E-37 ± 7.99E-37
f_2	**5.525E+00 ± 1.92E+00**	5.872E+00 ± 1.20E+00	**4.269E-01 ± 7.63E-01**	2.774E+00 ± 1.39E+00	**7.840E-18 ± 1.68E-17**	6.318E-08 ± 3.34E-07
f_3	**2.121E-03 ± 1.07E-02**	5.908E-03 ± 2.72E-02	**5.758E-65 ± 1.58E-64**	4.392E-57 ± 8.15E-57	**8.903E-27 ± 8.93E-27**	7.332E-24 ± 8.31E-24
f_4	6.803E-03 ± 2.88E-02	**8.299E-09 ± 3.88E-08**	7.453E-47 ± 4.01E-46	**2.346E-172 ± 0.00E+00**	**4.754E-91 ± 1.43E-90**	3.235E-78 ± 6.19E-78
f_5	**4.679E+02 ± 2.56E+02**	4.799E+02 ± 3.10E+02	1.727E+02 ± 1.71E+02	**1.432E+02 ± 1.46E+02**	1.165E+02 ± 1.86E+02	**9.740E+01 ± 1.32E+02**
f_6	**1.733E+00 ± 1.62E+00**	2.192E+00 ± 1.63E+00	2.047E+00 ± 3.71E+00	**1.469E+00 ± 2.56E+00**	1.794E+01 ± 2.63E+00	**1.761E+01 ± 3.48E+00**
f_7	**3.079E-15 ± 1.21E-15**	3.434E-15 ± 6.38E-16	**3.434E-15 ± 6.38E-16**	3.553E-15 3.94E-31	3.553E-15 ± 3.94E-31	**3.434E-15 ± 6.38E-16**
f_8	**1.166E-17 ± 6.28E-17**	4.883E-16 ± 2.57E-15	**1.033E-99 ± 3.71E-99**	4.563E-80 2.41E-79	**9.609E-42 ± 3.97E-41**	2.486E-35 ± 8.48E-35
f_9	2.883E-02 ± 1.75E-02	**2.035E-02 ± 1.74E-02**	**1.166E-02 ± 9.99E-03**	1.567E-02 1.39E-02	**1.922E-01 ± 8.54E-02**	2.314E-01 ± 7.44E-02
f_{10}	1.032E-01 ± 1.80E-02	**9.987E-02 ± 2.04E-09**	9.987E-02 ± 5.53E-09	**9.987E-02± 5.55E-10**	9.987E-02 ± 2.35E-09	**9.987E-02 ± 1.36E-09**

Pri $d = 30$ i $NP = 30$ DDE se pokazao bolji na f_3, sličan i neznatno bolji od DE na 4 funkcije (f_5, f_6, f_7, f_{10}), sličan i neznatno lošiji za 2 funkcije (f_1, f_9) i lošiji od DE na 3 funkcije (f_2, f_4, f_8). Za postavku parametra $NP = 50$ DDE je bolji na 3 funkcije (f_1, f_3, f_7), slični i neznatno bolji za 3 (f_5, f_6, f_9), slični i neznatno lošiji na 2 (f_2, f_{10}), i lošiji na 2 funkcije (f_4, f_8). Za posljednju postavku parametra $NP = 100$ DDE se pokazao kao bolja opcija od DE dajući bolje rezultate na 8 funkcija, od kojih su 4 slični (f_2, f_3, f_7, f_{10}). Bolje rezultate DE je ostvario samo na dvije funkcije (f_5, f_6), a i tada su samo neznatno bolji od DDE.

Tablica 4.3 *Prikaz rezultata DDE i DE pri* $d = 30$ *za deset testnih funkcija*

F	$NP = 30$		$NP = 50$		$NP = 100$	
	DDE	DE	DDE	DE	DDE	DE
	avg. ± std dev	avg. ± std dev	avg. ± std dev	avg. ± std dev	avg. ± std dev	avg. ± std dev
f_1	8.669E+00 ± 3.19E+01	**1.374E+00 ± 4.67E+00**	**6.608E-31 ± 3.47E-30**	2.122E-27 ± 4.27E-27	**2.668E-09 ± 2.26E-09**	9.439E-08 ± 5.68E-08
f_2	1.806E+02 ± 5.50E+02	**6.367E+01 ± 5.66E+01**	3.152E+01 ± 1.80E+01	**2.680E+01 ± 8.12E+00**	**1.949E+01 ± 1.15E+00**	2.161E+01 ± 7.78E-01
f_3	**6.482E-03 ± 1.83E-02**	7.447E-02 ± 3.43E-01	**7.625E-04 ± 8.31E-04**	1.866E-03 ± 3.02E-03	**1.683E+01 ± 5.75E+00**	2.106E+01 ± 7.00E+00
f_4	3.150E+11 ± 1.45E+12	**3.647E+10 ± 1.33E+11**	1.245E+02 ± 5.60E+02	**3.119E+01 ± 1.54E+02**	**6.097E-26 ± 1.41E-25**	2.722E-20 ± 9.58E-20
f_5	**2.536E+03 ± 6.63E+02**	2.978E+03 ± 4.91E+02	**2.520E+03 ± 7.38E+02**	2.717E+03 ± 8.79E+02	6.630E+03 ± 5.26E+02	**6.554E+03 ± 4.18E+02**
f_6	**2.027E+01 ± 5.93E+00**	2.121E+01 ± 5.55E+00	**5.963E+01 ± 2.76E+01**	6.291E+01 ± 3.36E+01	1.862E+02 ± 1.09E+01	**1.853E+02 ± 1.44E+01**
f_7	**1.582E+00 ± 1.10E+00**	1.649E+00 ± 9.55E-01	**7.461E-15 ± 1.91E-15**	1.279E-14 ± 5.99E-15	**1.419E-05 ± 5.62E-06**	8.855E-05 ± 2.36E-05
f_8	1.031E-02 ± 2.82E-02	**9.853E-03 ± 3.36E-02**	1.516E-16 ± 2.95E-16	**2.332E-17 ± 8.72E-17**	**7.399E-09 ± 5.52E-09**	1.748E-07 ± 2.35E-07
f_9	9.634E-02 ± 1.42E-01	**8.455E-02 ± 1.37E-01**	**1.479E-03 ± 3.40E-03**	1.561E-03 ± 3.68E-03	**2.639E-10 ± 3.37E-10**	8.863E-09 ± 8.79E-09
f_{10}	**3.632E-01 ± 1.11E-01**	3.699E-01 ± 1.32E-01	1.983E-01 ± 2.71E-02	**1.957E-01 ± 1.57E-02**	**2.526E-01 ± 4.32E-02**	2.581E-01 ± 4.47E-02

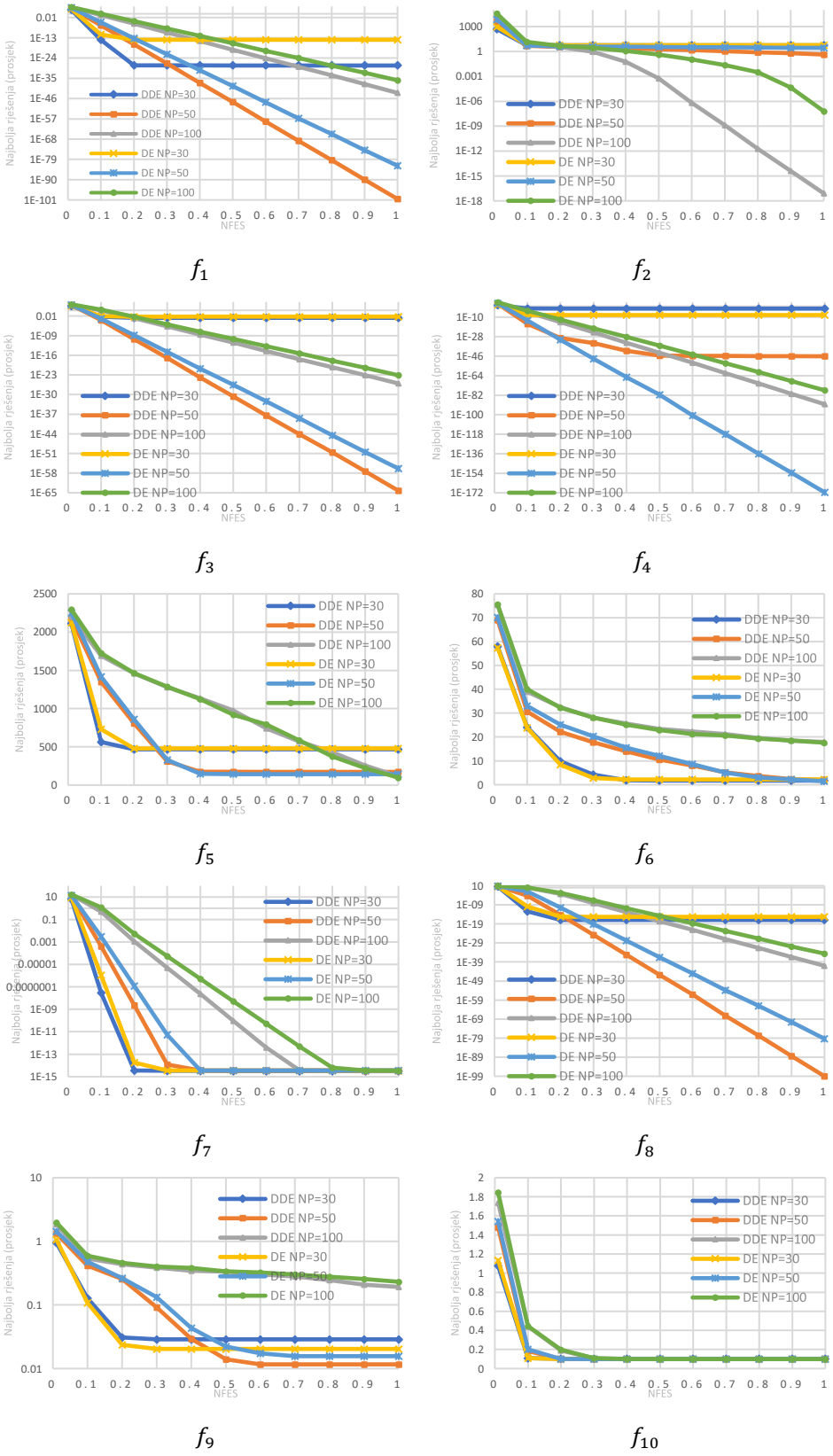
Rezultati DDE su bolji od DE na 19 od 30 ukupnih evaluacija (10 funkcija sa 3 postavke parametara) pri obje razmotrene dimenzionalnosti. Jedna od sličnosti koja se može primijetiti za obje inačice algoritma je da je veći broj kvalitetnijih rješenja ostvaren pri veličini populacije $NP = 50$ (10 od 20 za DDE i 12 od 20 za DE) dok je taj broj znatno manji pri $NP = 30$ (3 od 20 za DDE i 2 od 20 za DE) i manji pri $NP = 100$ (7 od 20 za DDE i 6 od 20 za DE) što je uočljivo i na slikama 4.1 i 4.2. Ovi rezultati sugeriraju da veličina populacije od $NP = 50$ omogućuje bolju ravnotežu između istraživanja i iskorištavanja u oba načina obnavljanja populacije na proučenim funkcijama.

Moguće je primijetiti i da je DDE generalno bolji u stvaranju veće kvalitete krajnjeg rješenja u većini slučajeva u odnosu na DE koji daje znatno bolje rješenje samo u jednom slučaju (na funkciji

f_4 pri $NP = 50$ i $d = 10$), gdje je vidljiva kontinuirana konvergencija i rješenja su više redova veličina veća od rješenja DDE. Potencijalno objašnjenje zašto DE u ovom slučaju ima znatno bolje performanse je u tome što pri sinkronom načina obnavljanja populacije algoritam ne radi odmah sa najboljim dostupnim rješenjima jer nisu još uvrštena u trenutnu populaciju, već nastavlja raditi sa "lošijim" roditeljima što mu daje značajnu prednost u ovom specifičnom slučaju.

Za navedeni slučaj sa funkcijom f_4 na kojoj DE ima bolje performanse algoritam DDE počinje da stagnira nakon otprilike 40% ukupnog $NFEs_{max}$. Kod funkcije f_4 do stagnacije može doći kada algoritam ne može pronaći optimalniji, odnosno najstrmiji put do globalnog minimuma koji algoritmu može biti težak za razaznati zbog naglog rasta nagiba što se više udaljava od globalnog minimuma. Ovaj problem postaje izraženiji sa većim brojem dimenzija, moguće ga je primijetiti za $d = 30$ kod oba algoritma, gdje razlika u performansama nije toliko drastična, ali kvaliteta rješenja je i dalje veća sa DE i algoritam se bolje nosi sa problemom. Također, što su vrijednosti bliže globalnom minimumu (odnosno nuli) područje pretrage postaje ravnije, razlika između potencijalnih rješenja je manja i teže postaje pronaći u kojem je smjeru ono optimalnije rješenje, ovo je slučaj kod rezultata sa vrijednosti dimenzionalnosti $d = 10$. Do istog problema za algoritme dolazi i kod funkcije f_1 za vrijednosti veličine populacije $NP = 30$ pri obje dimenzionalnosti, gdje se stagnacija pojavljuje već pri 30% $NFEs_{max}$. Kod f_2 i f_3 je problem u principu isti, ali se područja sa malim razlikama u nagibu nalaze unutar ravnih dolina u kojima se nalazi i globalni minimum, pronalaženje doline je jednostavno ali globalnog minimuma zahtjevno. Rezultati oba algoritma pri $NP = 30$ za ove dvije funkcije počinju rano stagnirati sa iznimkom za funkciju f_3 pri $d = 30$ gdje zbog povećane kompleksnosti funkcije algoritmi ne mogu jednako brzo da dođu do prostora u kome su za $d = 10$ stagnirali.

Kod multimodalnih funkcija ($f_5 - f_{10}$) do ranije stagnacije češće dolazi sa manjim vrijednostima veličine populacije ($NP = 30$ i $NP = 50$) nego sa većim ($NP = 100$) što se može objasniti time da multimodalne funkcije zahtijevaju drugačiju ravnotežu između istraživanja i iskorištavanja od unimodalnih radi izlaska iz neoptimalnih lokalnih minimuma. Ako čisto poredimo algoritme sa različitim NP, stagnacija u nekim od ovih slučajeva i nije loša stvar kada uzmemo u obzir da se radi o multimodalnim funkcijama i da su razlike u rješenjima pri ostale dvije veličine populacije male. U nekim slučajevima prisutni su i relativno ravni prostori oko globalnih minimuma kao i kod prethodno diskutiranih unimodalnih funkcija, funkcija f_8 je jedan od primjera gdje algoritmi za $NP = 30$ i obje dimenzionalnosti već rano i prije algoritama sa $NP = 50$ i $NP = 100$ pronalazi prostor globalnog minimuma sa malim razlikama u točkama, ali brzo stagnira i krajnje rješenje ostaje istog reda veličine.



Slika 4.1 *Grafički prikaz prosjeka najboljih rješenja prikupljenih tijekom izvođenja algoritma pri d = 10*

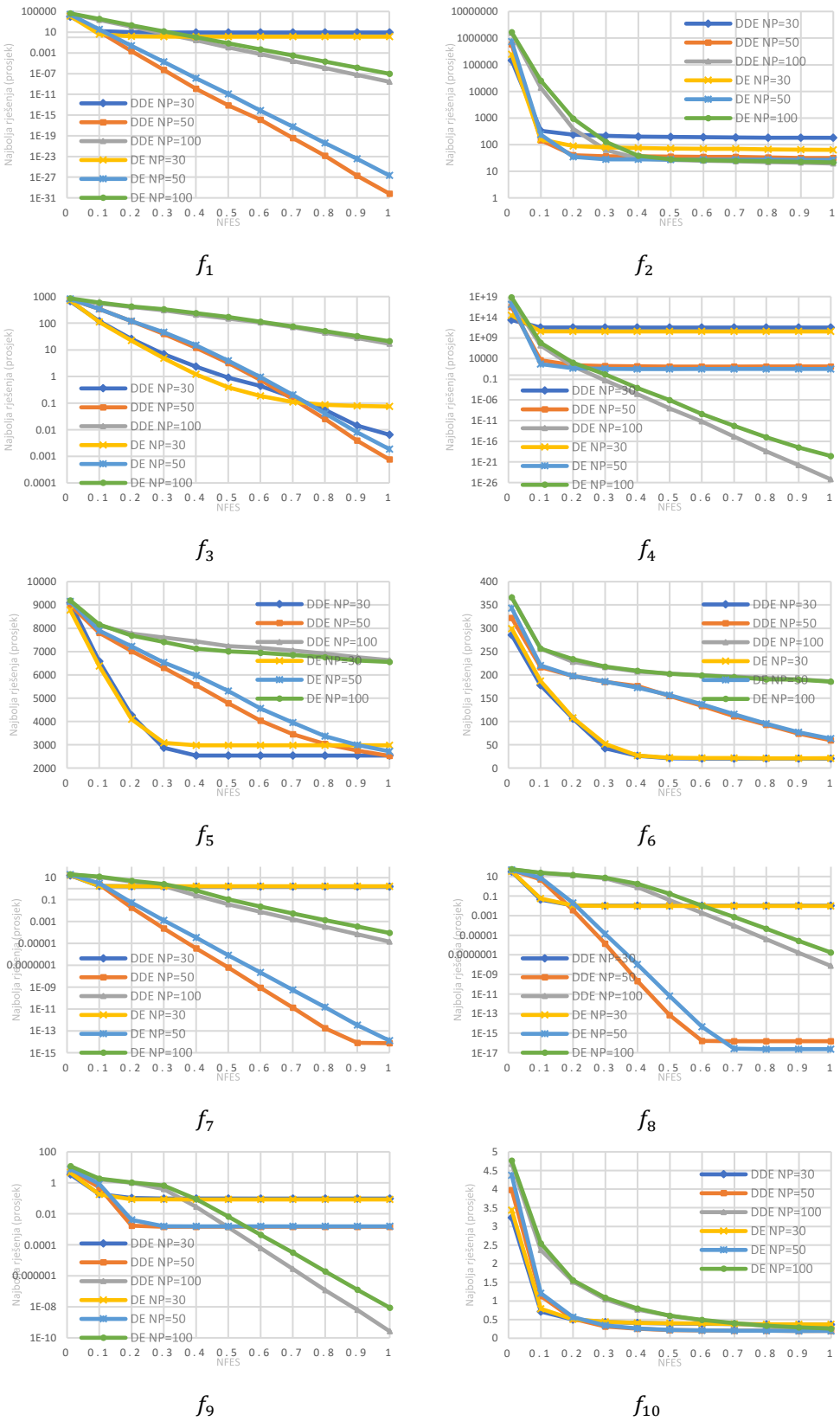
Kod funkcija f_1 i f_2 pri $NP = 30$ oba algoritma rano stagniraju, dok za $NP = 50$ i $NP = 100$ stopa stagnacija je manja i konvergencija je brža. Ipak DDE se pokazao bolji sa većom konvergencijom i blažom stagnacijom od DE.

Funkcija f_3 opet pokazuju slične trendove u konvergenciji i stagnaciji za jednake vrijednosti populacije također u korist DDE, sa jednom iznimkom pri $d = 30$ i $NP = 30$ gdje DE nakon određene točke počinje da brže konvergira od DDE, ali onda počinje stagnirati dok DDE nastavlja kontinuirano konvergirati.

Kod funkcija $f_5 - f_7$ situacija je nešto drugačija, algoritmi imaju lošije performanse pri $NP = 100$, dok sa $NP = 30$ algoritmi ostvaruju bolju stopu konvergencije i od $NP = 50$ i kvaliteta krajnjeg rješenja je slična u nekim slučajevima i bolja nego pri $NP = 50$ i $NP = 100$. Iznimka su f_5 pri $d = 10$ i f_7 pri $d = 30$ gdje algoritmi pri $NP = 30$ već rano počinju da stagniraju jer ne mogu pronaći izlaz i lokalnog minimuma. Kada promatramo individualne algoritme DDE nešto brže konvergira od DE, ali kvaliteta krajnjeg rješenja je identična.

Kod funkcija f_8 i f_9 performanse algoritma za neke od veličina populacija se čine kao da počinju stagnirati, ali nakon određene točke dolazi do ubrzavanja konvergencije, te kvaliteta krajnjeg rješenja postaje i bolja od bar jedne druge veličine populacije algoritma, uglavnom $NP = 30$ koja već rano počne stagnirati. Ovo je vidljivo kod $NP = 100$ na funkcijama f_8 i f_9 pri $d = 30$, te za vrijednost $d = 10$ na funkciji f_9 za veličinu populacije $NP = 50$. Moglo bi se zaključiti da algoritmi posjeduju bolju sposobnost izlaska iz lokalnih minimuma za ove vrijednosti populacije u ovom specifičnom slučaju. Funkcija f_9 je također specifična u tome da algoritmi proizvode kvalitetnija rješenja pri većoj dimenzionalnosti, osim za vrijednosti populacije $NP = 30$ gdje je razlika u rješenjima neznatna, dok je pri $NP = 50$ jednog reda veličine veća, a pri $NP = 100$ više redova veličine veća. DDE i ovdje ima nešto bolju konvergenciju i kvalitetu rješenja osim u dva slučaja kada DE počinje stagnirati ili jednu točku kasnije sa kvalitetnijim rješenjem(f_8 pri $d = 30$ i $NP = 50$) ili u istoj točki sa nešto kvalitetnijim rješenjem (f_9 pri $d = 10,\ NP = 50$).

Kod f_{10} algoritmi stagniraju dosta rano za sve tri postavke populacije, s nešto kasnijom stagnacijom kod većih populacija, ali postižu sličnu kvalitetu rješenja. Pri $d = 30$ stopa konvergencije opada blaže što je vidljivije kako veličina populacije raste i do stagnacije dolazi nešto kasnije nego pri $d = 10$. Performanse algoritama su identične za sve veličine populacije i dimenzionalnosti.



Slika 4.2 *Grafički prikaz prosjeka najboljih rješenja prikupljenih tijekom izvođenja algoritma pri d = 30*

Moguće je primijetiti da se algoritmi ponašaju kako je i donekle očekivano s obzirom na tip funkcije. Kod unimodalnih funkcija ($f_1 - f_4$) češće je vidljiva kontinuirana i linearna stopa konvergencije u slučajevima kada ne dolazi do rane stagnacije, dok kod multimodalnih funkcija ($f_5 - f_{10}$) primjećuju se trendovi gdje je konvergencija usporena i otežana zbog velikog broja lokalnih minimuma.

Slike 4.1 i 4.2 ukazuju na bolju konvergenciju DDE u odnosu na DE što se dalje odražava u činjenici da DDE brže reagira na poboljšanja i promjene u smjeru optimalnog prostora pretraživanja zbog svog asinkronog načina obnavljanja populacije koji mu daje veću virtualnu populaciju [17]. Veća virtualna populacija u biti znači da algoritam raspolaže sa većim brojem potencijalnih rješenja (jedinki) nego što je propisano parametrom NP. Do ovoga dolazi jer u asinkronom načinu obnavljanja populacije dolazi do zamjene sa novim kvalitetnijim rješenjima odmah u trenutku kada su dostupna što rezultira i boljim iskorištavanjem jer se nova zamijenjena rješenja dalje koriste u procesima križanja i mutacije.

5. ZAKLJUČAK

Način obnavljanja populacije može imati značajan utjecaj na performanse algoritma u vidu kvalitete krajnjeg rješenja ali i brzine konvergencije i robusnosti. U ovom radu opisane su i uspoređene dvije inačice algoritma diferencijalne evolucije sa jedinom razlikom u načinu obnavljanja populacije, jedna sa standardnim, sinkronim, a druga sa asinkronim načinom obnavljanja populacije. S obzirom da se vrši usporedba načina obnavljanja populacije, promatran je i utjecaj različitih postavki parametra koji direktno utječe na veličinu populacije. Oba algoritma su primijenjena za optimizaciju deset često korištenih testnih funkcija pri dvije različite dimenzionalnosti u svrhu usporedbe njihove robusnosti sa povećanjem kompleksnosti funkcije cilja.

Eksperimentalnom analizom utvrđeno je da, iako manje zastupljena, asinkrona inačica algoritma se pokazala boljim izborom na većini testnih funkcija. Asinkroni način obnavljanja populacije u većini slučajeva dovodi do brže konvergencije i kasnije stagnacije u usporedbi sa sinkronim. Također je prikazan utjecaj veličine populacije na obje inačice sa varirajućim rezultatima ovisno o specifičnoj testnoj funkciji, ipak uz veličinu populacije $NP = 50$ zabilježen je nešto veći broj kvalitetnijih rješenja nego za ostale dvije vrijednosti ($NP = 30\ i\ NP = 100$).

S obzirom na velik broj inačica u literaturi koje koriste različite postupke mutacije, u budućim istraživanjima bilo bi korisno usporediti utjecaj sinkronog i asinkronog načina obnavljanja populacije uz različite postupke mutacije, s ciljem utvrđivanja koji od ta dva pristupa općenito ima povoljniji utjecaj na performanse algoritma. Također, mogla bi se izravno usporediti i odrediti razlika u tome koliko učinkovito oba pristupa koriste računalne resurse, mjerenjem faktora poput vremena izvršavanja ili korištenja memorije, što bi moglo dati dodatni uvid u njihove prednosti i nedostatke u određenim slučajevima gdje su ovi faktori bitni.

LITERATURA

[1] A. P. Engelbrecht, *Computational intelligence: an introduction*, 2nd ed. John Wiley & Sons: Chichester, England ; Hoboken, NJ, 2007.

[2] V.-H. Truong, S.-E. Kim, „Reliability-based design optimization of nonlinear inelastic trusses using improved differential evolution algorithm“, *Advances in Engineering Software*, sv. 121, str. 59–74, srp. 2018.

[3] N. H. Awad, M. Z. Ali, R. Mallipeddi, P. N. Suganthan, „An efficient Differential Evolution algorithm for stochastic OPF based active–reactive power dispatch problem considering renewable generators“, *Applied Soft Computing*, sv. 76, str. 445–458, ožu. 2019.

[4] G. A. Strofylas, K. N. Porfyri, I. K. Nikolos, A. I. Delis, M. Papageorgiou, „Using synchronous and asynchronous parallel Differential Evolution for calibrating a second-order traffic flow model“, *Advances in Engineering Software*, sv. 125, str. 1–18, stu. 2018.

[5] C. Lins, D. Eckhoff, A. Klausen, S. Hellmers, A. Hein, S. Fudickar, „Cardiopulmonary resuscitation quality parameters from motion capture data using Differential Evolution fitting of sinusoids“, *Applied Soft Computing*, sv. 79, str. 300–309, lip. 2019.

[6] R. Storn, K. Price, „Differential Evolution: A Simple and Efficient Adaptive Scheme for Global Optimization Over Continuous Spaces“, *Journal of Global Optimization*, sv. 23, sij. 1995.

[7] R. Storn, K. Price, „Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces“, *Journal of Global Optimization*, br. 4, sv. 11, str. 341–359, pros. 1997.

[8] K. V. Price, R. M. Storn, J. A. Lampinen, *Differential evolution: a practical approach to global optimization*. Springer: Berlin ; New York, 2005.

[9] A. E. Eiben, J. E. Smith, *Introduction to Evolutionary Computing*. Springer Science & Business Media, 2007.

[10] D. Bajer, G. Martinović, J. Brest, „A population initialization method for evolutionary algorithms based on clustering and Cauchy deviates“, *Expert Systems with Applications*, sv. 60, str. 294–310, lis. 2016.

[11] D. Bajer, „Adaptive k -tournament mutation scheme for differential evolution“, *Applied Soft Computing*, sv. 85, str. 105776, pros. 2019.

[12] S. Das, P. N. Suganthan, „Differential Evolution: A Survey of the State-of-the-Art“, *IEEE Transactions on Evolutionary Computation*, br. 1, sv. 15, str. 4–31, velj. 2011.

[13] S. Das, S. S. Mullick, P. N. Suganthan, „Recent advances in differential evolution – An updated survey“, *Swarm and Evolutionary Computation*, sv. 27, str. 1–30, tra. 2016.

[14] D. Bajer, „Parameter control for differential evolution by storage of successful values at an individual level“, *Journal of Computational Science*, sv. 68, str. 101985, tra. 2023.

[15] A. Draa, K. Chettah, H. Talbi, „A Compound Sinusoidal Differential Evolution algorithm for continuous optimization“, *Swarm and Evolutionary Computation*, sv. 50, str. 100450, stu. 2019.

[16] J. Brest, S. Greiner, B. Boskovic, M. Mernik, V. Zumer, „Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems“, *IEEE Transactions on Evolutionary Computation*, br. 6, sv. 10, str. 646–657, pros. 2006.

[17] Anyong Qing, „Dynamic differential evolution strategy and applications in electromagnetic inverse scattering problems“, *IEEE Transactions on Geoscience and Remote Sensing*, br. 1, sv. 44, str. 116–125, sij. 2006.

[18] H. Peng, Z. Guo, C. Deng, Z. Wu, „Enhancing differential evolution with random neighbors based strategy“, *Journal of Computational Science*, sv. 26, str. 501–511, svi. 2018.

[19] X. Zhao, G. Xu, L. Rui, D. Liu, H. Liu, J. Yuan, „A failure remember-driven self-adaptive differential evolution with top-bottom strategy“, *Swarm and Evolutionary Computation*, sv. 45, str. 1–14, ožu. 2019.
[20] X. Yao, Y. Liu, G. Lin, „Evolutionary programming made faster“, *IEEE Transactions on Evolutionary Computation*, br. 2, sv. 3, str. 82–102, srp. 1999.
[21] M. Jamil, X.-S. Yang, „A literature survey of benchmark functions for global optimisation problems“, *International Journal of Mathematical Modelling and Numerical Optimisation*, br. 2, sv. 4, str. 150, srp. 2013.

SAŽETAK

U radu su razmotrene dvije inačice algoritma diferencijalne evolucije, jednog od najpopularnijih evolucijskih algoritama koji se pokazao vrlo učinkovit za probleme kontinuirane optimizacije. Inačice se razlikuju samo u načinu obnavljanja populacije, jedna sa sinkronim gdje se populacija obnavlja generaciju po generaciju, a druga sa asinkronim gdje se populacija obnavlja odmah kada su nova kvalitetnija rješenja dostupna. Upotrebom ostvarenog programskog rješenja provedena je eksperimentalna analiza u kojoj su dvije inačice algoritma primijenjene na deset često korištenih testnih funkcija pri dvije različite dimenzionalnosti. Također je razmotren utjecaj parametra veličine populacije korištenjem tri različite vrijednosti istog. Ostvareni rezultati daju uvid u to kako način obnavljanja populacije direktno utječe na performanse algoritma. Daljnjom usporedbom dvije inačice primijećeno je da ona sa asinkronom implementacijom obnavljanja populacije ostvaruje bolje performanse u usporedbi sa standardnom, sinkronom inačicom u većini slučajeva.

Ključne riječi: asinkrona implementacija, diferencijalna evolucija, kontinuirana optimizacija, obnavljanje populacije, sinkrona implementacija.

ABSTRACT

Comparison of synchronous and asynchronous population update in differential evolution

The thesis discusses two implementations of the differential evolution algorithm, one of the most popular evolutionary algorithms, which has proven to be highly effective in problems of continuous optimization. The implementations differ only in the method of population updating: one using a synchronous method, where the population is updated generation by generation, and the other using an asynchronous method, where the population is updated immediately when new, better solutions become available. Using the implemented software solution, an experimental analysis was conducted in which the two implementations of the algorithm were applied to ten commonly used benchmark functions at two different dimensionalities. The impact of the population size parameter was also examined by using three different values of it. The obtained results provide insight into how the method of population updating directly affects the algorithm's performance. Further comparison of the two variants revealed that the asynchronous implementation achieves better performance compared to the standard synchronous implementations in most cases.

Keywords: asynchronous implementation, differential evolution, continuous optimization, population update, synchronous implementation.

ŽIVOTOPIS

Valentin Veselčić rođen je 15. rujna 2002. godine u Brčkom, Bosna i Hercegovina. Tehničku srednju školu pohađa u Brčkom, smjer elektrotehničar računarstva, koju završava 2021. godine. Iste godine upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, prijediplomski sveučilišni studij Računarstvo, izborni blok Programsko inženjerstvo.

PRILOZI

Git repozitorij sa kodom ostvarenog programskog rješenja: gitlab.com