SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD

Evolucija evolucijskih algoritama

Jan Tomljanović

Voditelj*: prof.dr.sc. Domagoj Jakobović*

Zagreb, svibanj, 2015.

Sadržaj

1. Uvod

Evolucijski algoritmi se danas primjenjuju na mnogim područjima u slučajevima kada je do optimalnog rješenja nemoguće doći u prihvatljivom vremenu. Oni mogu brzo pronaći rješenja koja su dovoljno dobra za primjenu i često vrlo blizu optimalnima. Postoji mnogo različitih načina na koji se ostvaruju. Pri tome, čak i kada je odabran jedan određen način ostvarenja, evolucijski algoritam i dalje ovisi o mnogim parametrima (veličina populacije, broj generacija nakon kojih će se zaustaviti, različite konstante vezane uz vjerojatnosti određenih događaja). Uz toliko mogućnosti, ne zna se koja kombinacija svih tih varijabli dovodi do optimalnog evolucijskog algoritma. Optimalan evolucijski algoritam je onaj koji bi za svaki problem bio bolji od bilo kojeg drugog evolucijskog algoritma napravljenog za taj problem. Međutim, prema teoremima "nema-besplatnog-ručka" (engl. *no free lunch theorems)*(Wolpert, Macready, 1996) čini se da takav algoritam ne postoji. Dakle, evolucijski algoritam može biti optimalan samo za određenu vrstu problema, ne za sve. To i dalje znači da treba otkriti koji je algoritam optimalan za pojedinu vrstu problema.

Ovaj rad će pokušati pružiti način na koji se može ostvariti takav evolucijski algoritam, agnostično prema vrsti problema. Cilj je razviti algoritam koji će moći evoluirati evolucijski algoritam. Idejno, ako postoji neki opći evolucijski algoritam, i ako se uzastopno primjenjuje na jednoj vrsti problema, možda ga se može evoluirati da postane bolji za tu vrstu problema. Isto tako, ako se počne primjenjivati na drugu vrstu problema, evolucija bi ga trebala usmjeriti u nekom drugom smjeru, tako da se razvija prikladno za taj problem. Taj algoritam koji će vršiti evoluciju evolucijskog algoritma kao mehanizam će koristiti genetsko programiranje, poznatu tehniku evolucije progama.

S obzirom na mogućnost zabune pri razumijevanju o kojem će se evolucijskom algoritmu u nekom trenutku pričati, onom nad kojim će se vršiti evolucija ili onom koji vrši evoluciju, koristit će se ovo pravilo: na algoritam nad kojim će se vršiti evolucija referira se samo kao na 'evolucijski algoritam', a na algoritam koji će vršiti evoluciju referira se kao na 'GP algoritam' (algoritam koji koristi genetsko programiranje).

2. Ideja evolucije evolucijskih algoritama

Ideja je razviti GP algoritam koji će imati populaciju evolucijskih algoritama i njih evoluirati kako bi kao rezultat dobio što bolji evolucijski algoritam. Načelo rada je jednako kao i u bilo kojem drugom slučaju genetskog programiranja gdje se evoluiraju programi, samo što je u ovom slučaju i sam program koji se evoluira evolucijski algoritam.

Prethodno već postoje radovi koji se bave evolucijom evolucijskih algoritama, međutim ima ih jako malo. U početku su postojali pokušaji evolucije genetskih operatora koji se koriste u evolucijskim algoritmima (Edmonds, 2001) i oni su bili primjenjivani u nepromijenjivom evolucijskom algortimu. Prva dva rada koji se bave evoluiranjem cijelog evolucijskog algoritma (Oltean 2003, Oltean 2005) prikazuju algoritam kao niz instrukcija koji se ponavlja:

```
for (int i = 0; i < brojGeneracija; i++) {
      instrukcija1();
      instrukcija2();
      instrukcija3();
      ...
}
```

Taj niz instrukcija je podložan evoluciji, tj. operatori mutacije i križanja ga mijenjaju. Ovaj način prikaza evolucijskog algoritma će se primjenjivati i u ovom radu.

U novijem radu (Oltean 2007), umjesto gore prikazanog niza instrukcija, koji obično sadržava veći broj instrukcija, evolucijski algoritam se svodi na jedan uzorak koji sadržava vrlo mali broj instrukcija. Kroz taj se niz također iterira mnogo puta, međutim te

instrukcije dobivaju nove argumente nad kojima rade pri svakoj iteraciji, za razliku od gornjih instrukcija koje se ne mijenjaju, tj. ne ovise ni o kakvim argumentima tijekom izvršavanja algoritma. Na taj način prostor pretraživanja znatno se smanjuje i brže je naći kvalitetan uzorak, ali se s druge strane gubi količina slobode u definiranju konačnog niza instrukcija koji će algoritam izvršiti.

Ovaj rad će se nastaviti na gore spomenute, evoluirati će niz instrukcija algoritma. Međutim, kako je u uvodu navedeno da svojstva i kvaliteta evolucijskog algoritma ovise o načinu ostvarenja, ali nekada i o mnogim parametrima koji se danas većinom ručno namještaju, u obzir će se uzeti oba činitelja. Način ostvarenja odgovara odabiru niza instrukcija, a namještanje parametara odvijati će se neovisno, također evolucijom.

2.1. Linearno genetsko programiranje

Pri evoluciji algoritma koristit će se genetsko programiranje. Najćešće se u tom slučaju za predstavljanje programa koristi stablo kojemu su listovi ulazni argumenti, a ostali čvorovi predstavljaju operacije. Ovdje nije korišten takav koncept, već koncept linearnog genetskog programiranja gdje se program (evolucijski algoritam) predstavlja kao niz instrukcija. Ovdje se objašnjavaju operacije koje će GP algoritam provoditi nad evolucijskim algoritmom.

U ostvarenju evolucijskog algoritma broj instrukcija daleko je veći nego na sljedećim primjerima, koji služe samo za konceptualni prikaz.

Primjer mutacije jednog programa prikazan je ovdje:

Program prije:		Program poslije:
`instrukcija1();`		`instrukcija1();`
`instrukcija2();`	→	`novaInstrukcija2();`
`instrukcija3();`	`mutiraj()`	`instrukcija3();`
`instrukcija4();`	→	`instrukcija4();`
`instrukcija5();`		

Mutacija može zamijeniti jednu instrukciju drugom, ili je modificirati. U oba slučaja nastaje nova instrukcija. Mutacija također može obrisati (gornji slučaj) ili nadodati novu instrukciju. Sve ove operacije obavljaju se nasumično: odabir koja operacija će se provesti, koja instrukcija će biti odabrana bilo za dodavanje, brisanje, zamjenu ili modifikaciju te kakva će modifikacija biti provedena nad odabranom instrukcijom. Za svaku instrukciju originalnog programa moguća je svaka od navedenih promjena: zamjena, brisanje i modifikacija.

Križanje je ostvareno tako da prima dva programa (niza instrukcija) i kao rezultat daje jedan novi program. Križanje dva progama obavlja se na ovaj način:

Program 1:	Program 2:
`instrukcija1();`	`instrukcija23();`
`instrukcija2();`	`instrukcija25();`
`instrukcija3();`	`instrukcija27();`
`instrukcija4();`	`instrukcija14();`

Križanjem Programa 1 i Programa 2 nastaje novi program:

```
instrukcija1();
instrukcija2();
instrukcija27();
instrukcija14();
```

Križanje se obavlja tako da se odredi jedna točka prekida u nizu instrukcija, tj. odredi se granica prije koje će se kopirati instrukcije prvog programa, a nakon koje će se kopirati instrukcije drugog programa. Mjesto točke prekida određuje se nasumičnim odabirom. U navedenom primjeru granica je između druge i treće instrukcije programa.

2.2. GP algoritam

Razvijen je GP algoritam koji prima evolucijski algoritam u odgovarajućem prikazu kao niz instrukcija. Točan izgled instrukcija analiziran je u sljedeće dvije cjeline, budući da su korištena dva različita načina prikaza instrukcija, a time i algoritma. Međutim, GP algoritam načelno radi jednako u oba slučaja, neovisno o tome kakve su točno instrukcije u pitanju.

Pseudokod GP algoritma:

```
1. gpAlgoritam(evolucijskiAlgoritam) {
2.       populacija <- napraviPopulaciju(evolucijskiAlgoritam);
3.       za svaki i od 0 do KONST {
4.            održiTurnir(populacija);
5.       }
6.       vrati nađiNajbolji(populacija);
7. }
```

Pseudokod funkcije *održiTurnir*:

```
1. održiTurnir(populacija) {
2.      sudionici <- odaberiSudionike(populacija);
3.      (najlošiji, bolji, najbolji) <-
   sortirajPoDobroti(sudionici);
4.      najlošiji <- primijeniOperator(najbolji, bolji);
5.}
```

GP algoritam od predanog algoritma stvara populaciju raznolikih algoritama. Predani algoritam ne mora imati definiran niz instrukcija, a i ako već ima, to nije važno, GP algoritam će svakom evolucijskom algoritmu u populaciji pridjeliti novi, nasumični niz instrukcija. Predani evolucijski algoritam je važan utoliko što definira na koji će način on izvršavati funkcije mutacije, križanja i selekcije te zna kako izračunati dobrotu jedinke u populaciji. Zatim se neki određeni broj puta izvršava funkcija *održiTurnir.* Ta funkcija odabere neki broj sudionika u liniji 2. Zatim u liniji 3 sortira sudionike po dobroti. Izračunavanje dobrote evolucijskog algoritma radi se tako da se algoritam pokrene određen broj puta, zapamte se rezultati koje je vraćao i iz medijana vrijednosti rezultata izvede se dobrota algoritma (detaljnije pod 2.5.). U liniji 4 najlošiji sudionik zamjenjuje se s rezultatom primjene operatora na bolje sudionike. Operator koji će se primijeniti je mutacija ili križanje. U slučaju mutacije mutirat će se algoritam *najbolji*, u slučaju križanja križat će se *bolji* i *najbolji*. Povratna vrijednost GP algoritma je najbolja pronađena verzija evolucijskog algoritma.

2.3. Detalji GP algoritma

U poglavlju 2.2. naveden je jednostavni pseudokod GP algoritma. Međutim rad algoritma ovisi o još mnogo parametara. Iznosi parametara koji su mijenjani za različita ispitivanja su izneseni uz podatke ispitivanja pa ovdje oni neće biti opisani.

Budući da GP algoritam nasumično generira početni niz instrukcija za svaki evolucijski algoritam u populaciji potrebno je odrediti vjerojatnosti stavljanja nekog tipa instrukcije u algoritam. Odabrani su ovi iznosi: mutacija 34%, selekcija 33% i križanje 33%. Dakle, praktično je podjednaka vjerojatnost da se izabere neki tip instrukcije. Indeksi koji će biti korišteni za pojedinu instrukciju biraju se nasumično, svi s podjednakom vjerojatnosti.

GP algoritam u metodi *održiTurnir* opisanoj u poglavlju 2.2. na mjesto najgoreg algoritma u turniru stavlja rezultat primjene operatora mutacije ili križanja na neke od ostalih algoritama u turniru. Kao parametri algoritma unesene su vjerojatnosti da se dogodi mutacija (50%) ili križanje (50%). Međutim, koristi se i sljedeći mehanizam: nakon što je odigran određen postotak ukupnog broja turnira (85%), smanjuje se vjerojatnost da se dogodi križanje na 15%. To je implementirano zato što križanje u populaciji stvara izmijenjeniji algoritam nego mutacija, pa se u zadnjih 15% turnira dopušta mutaciji da usavršava dosad stvorene algoritme.

Također, broj sudionika turnira mijenja se kroz vrijeme, tj. postotak odigranih turnira. Budući da pritisak selekcije raste s brojem sudionika turnira i time algoritam snažnije preferira trenutno najbolje jedinke (Xie, 2012), broj sudionika je povećavan s brojem odigranih turnira. U razdoblju 0-60% broja odigranih turnira broj sudionika je tri, 60-80% četiri, 80-90% pet i 90-100% šest.

Pri mutaciji evolucijskog algoritma potrebno je odlučiti koliko će se

promjena odviti na njegovim instrukcijama. Taj parametar postavljen je na vrijednost od 25%, dakle mutacija mijenja četvrtinu instrukcija algoritma.

2.4. Jednostavan prikaz evolucijskog algoritma

Nakon što je izabrano predstavljati evolucijski algoritam nizom instrukcija, struktura i ponašanje algoritma ovisi o tome kako će te instrukcije biti definirane. Za jednostavniju verziju koristit će se iste instrukcije kao i u radu *Evolving Evolutionary Algorithms using Linear Genetic Programming*, Oltean 2005. To znači da će svaka instrukcija na neko mjesto u populaciji zapisati rezultat jedne od sljedećih operacija: selekcija, mutacija, križanje. Dakle, mogući oblici instrukcije su:

(1) `pop[indeks₁] = mutiraj(pop[indeks₂]);`

(2) `pop[indeks₁] = selektiraj(pop[indeks₂], pop[indeks₃]);`

(3) `pop[indeks₁] = križaj(pop[indeks₂], pop[indeks₃]);`

Oznaka *pop* označava polje u koje su spremljene jednike populacije.

Sama funkcionalnost mutacije, selekcije i križanja ovisi o evolucijskom algoritmu koji će biti predan GP algoritmu.

Kod mutacije (koju provodi GP algoritam) prethodno spomenuta mogućnost modifikacije jedne instrukcije u ovom će slučaju mijenjati indekse. Ostali načini mutacije koje provodi GP algoritam su općeniti i neovisni o tome kako izgleda instrukcija (dodavanje, brisanje ili zamjena instrukcije).

Ovaj način prikaza evolucijskog algoritma dopušta GP algoritmu da kontrolira (kroz evoluciju) kojim tijekom će se odvijati selekcija, mutacija i križanje. Također, kontrolira na kojim dijelovima populacije će se ti operatori primjenjivati. Suprotno tome, svaki ručno napisan evolucijski algoritam ima već definiran redoslijed odvijanja instrukcija.

Ovim pristupom traži se najbolji redoslijed bez ručnog pisanja različitih verzija algoritma, pušta se GP algoritmu da optimalni redoslijed pronađe evolucijom.

2.5. Složeniji prikaz evolucijskog algoritma

Gore naveden jednostavan prikaz evolucijskog algoritma ima sljedeću manu: tijekom cijelog izvršavanja evolucijskog algoritma indeksi koji određuju na koji će član populacije neki operator biti primijenjen i na koje će se mjesto u populaciji zapisati rezultat se ne mijenjaju. Operator selekcije je taj koji filtrira bolje od lošijih, međutim ne uklanja lošije, a i indeksi u populaciji na koje se primjenjuje su stalni. Suprotno tome, ručno napravljen evolucijski algoritam koji koristi npr. turnirsku selekciju će prilikom svakog turnira mijenjati mjesto gdje će biti zapisan rezultat (obično na mjesto najgoreg člana (onog s najmanjom dobrotom) populacije koji sudjeluje u turniru) i koji će članovi populacije biti uzeti za primjenu operatora (obično najbolji (onaj s najvećom dobrotom) koji sudjeluju u turniru). Ovo znači da se jednostavan prikaz evolucijskog algoritma tromo prilagođava informacijama o dobroti jedinki koje može dobiti tijekom izvođenja i ima slab selekcijski pritisak. Time je ručno pisani algoritam u velikoj prednosti, koristi više informacija.

Iz tog razloga se uvodi složeniji oblik instrukcija:

(1) pop[najlošiji(pop[in_1], pop[in_2], ... , pop[in_K])] =

mutiraj(najbolji(pop[in_1], pop[in_2], ... , pop[in_K]));

(2) pop[najlošiji(pop[in_1], pop[in_2], ... , pop[in_K])] =

križaj(najbolji(pop[in_1], pop[in_2], ... , pop[in_K]),
drugiNajbolji(pop[in_1], pop[in_2], ... , pop[in_K]));

Oznaka *pop* predstavlja isto što i prije, polje gdje je zapisana populacija. Funkcije *najlošiji, najbolji i drugiNajbolji* primaju više članova populacije i vraćaju indeks najgoreg, najboljeg i drugog najboljeg, respektivno. Najgori je onaj s najmanjom dobrotom, najbolji onaj s najvećom, a drugi najbolji onaj s drugom najvećom dobrotom. U svakoj pojedinačnoj instrukciji za sve se funkcije koristi isti skup od *K* indeksa. *K* je nasumičan broj između 3 i 6 uključno. Time se daje još jedan stupanj slobode evoluciji koja sama određuje koje instrukcije će biti efikasnije. Unutar jednog algoritma instrukcije mogu imati različiti broj indeksa koji koriste.

Kao i u prethodnom slučaju, kod mutacije modifikacija znači mijenjanje indeksa koji se koriste u instrukciji.

Ovakav pristup omogućuje konstantno prilagođavanje algoritma trenutnom stanju populacije tijekom izvršavanja. Mana ovog pristupa je potreba za češćim određivanjem dobrote jedinki koje je nužno pri izvršavanju svake instrukcije.

2.6. Izračunavanje i usporedba dobrota evolucijskih algoritama

Dobrota evolucijskog algoritma mjeri se u njegovoj uspješnosti pri rješavanju nekog problema, tj. u kvaliteti rješenja koje pronađe za dani problem. Budući da GP algoritam mora raditi bez informacije za koji je problem evolucijski algoritam namijenjen, potrebno je pronaći univerzalni način mjerenja kvalitete rješenja koju daje evolucijski algoritam.

Ovdje je pretpostavljeno da je izlaz evolucijskog algoritma realni broj, što i jest istina za većinu evolucijskih algoritama. Međutim, evolucijski algoritam ne daje uvijek isto rješenje i zato se dobrota izračunava uzimanjem medijana svih rezultata evolucijskog algoritma pokrenutog

određen broj puta, što pruža pouzdaniju ocjenu algoritma. Odabrano je da će se algoritam pokretati 200 puta. Međutim kako bi dobrota u potpunosti predstavljala kvalitetu evolucijskog algoritma, uz ovako izračunat broj, potrebno je znati traži li algoritam minimum ili maksimum. Zato GP algoritam kao dobrotu evolucijskog algoritma uzima obje informacije. Kao primjer, u tablici 2.1. prikazani su rezultati izvođenja evolucijskog algoritma. Algoritam koji traži minimum funkcije pokrenut je 7 puta umjesto 200 kako bi se lakše opisao način izračuna dobrote.

REDNI BROJ IZVOĐENJA	*1.*	*2.*	*3.*	*4.*	*5.*	*6.*	*7.*
REZULTAT	19.1	19.5	23.2	20.0	18.9	22.4	25.5

Tablica 2.1. Rezultati izvođenja evolucijskog algoritma

Medijan rezultata je 20.0, što znači da je ukupna dobrota algoritma par vrijednosti (20.0, traži se minimum).

Usporedba dobrota dva evolucijska algoritma koji rješavaju jedan problem je jednostavna. Budući da se u svakom trenutku međusobno uspoređuju samo algoritmi koji rade na istom problemu, uvijek će ili oba tražiti minimum ili oba maksimum. Tada je bolji onaj koji daje manji ili veći rezultat, respektivno.

U napisanom pseusokodu GP algoritma (poglavlje 2.2) nije specificirana jedna dodatna funkcionalnost koja je ostvarena u implementaciji. Pri pozivanju GP algoritma, predani evolucijski algoritam sadržava informaciju o tome kako se izračunava dobrota jedinke. Međutim, on može sadržavati i više od jednog načina izračuna i u tom slučaju algoritam provodi izračun dobrote evolucijskog algoritma za svaku od ponuđenih načina izračuna dobrote jedinki. Time se evolucijski algoritam evoluira s obzirom na njegovu uspješnost na različitim problemima (koji su zapravo predstavljeni načinom izračuna dobrote jedinke). Kao rezultat dobiva se evolucijski algoritam koji je

najuspješniji u rješavanju skupa problema. U tom se slučaju pri usporedbi dva algoritma uspoređuju sve njihove izvedbe za različite načine izračuna dobrote jedinki. Pri optimizaciji funkcija, ovime se postiže da evolucijski algoritam evoluira prema tome da daje što bolja rješenja za različite funkcije.

S obzirom na ovu navedenu funkcionalnost, potrebno je opisati usporedbu evolucijskih algoritama koji rješavaju skup problema. Pri usporedbi dva evolucijska algoritma boljim se proglašava onaj koji ima više boljih pojedinačnih dobrota, tj. onaj koji bolje rješava više problema nego drugi. Npr. neka evolucijski algoritmi daju dobrote prikazane u tablici 2.2.

	FUNKCIJA 1	*FUNKCIJA 2*	*FUNKCIJA 3*	*FUNKCIJA 4*
Evolucijski algoritam A	10	4	3.3	0.3
Evolucijski algoritam B	11	1.44	3.9	0.9

Tablica 2.2. Dobrota evolucijskih algoritama primjenjenih na više funkcija

Uz pretpostavku da se za sve funkcije traži minimum funkcije, bolje rezultate ima algoritam A koji daje manju vrijednost za tri od četiri funkcije i zato se proglašava boljim. U slučaju izjednačenog rezultate boljim se proglašava nasumični algoritam.

2.7. Evolucija parametara

Uz dosad opisanu evoluciju niza instrukcija evolucijskog algoritma, u GP algoritmu koji koristi jednostavan prikaz instrukcija provodit će se i evolucija parametara algoritma. Parametar je definiran kao realni broj. GP algoritam neće znati što taj parametar predstavlja u evolucijskom algoritmu, ali ponekad će ga mutirati. Unutar evolucijskog algoritma može biti korišten proizvoljan broj parametara za različite svrhe i sve njih će GP algoritam tretirati na jednak način.

3. Programsko ostvarenje

Implementacija GP algoritma i svih popratnih detalja zamišljena je kao paket koji se pruža korisniku koji želi evoluirati svoj evolucijski algoritam. Pažnja je bila usmjerena na odvajanje funkcionalnosti evolucijskog algoritma koju mora pružiti korisnik i funkcionalnosti koju GP algoritam pruža za evoluciju tog algoritma. Na taj način postignuto je da GP algoritam samostalan i radi za bilo kakav predani evolucijski algoritam.

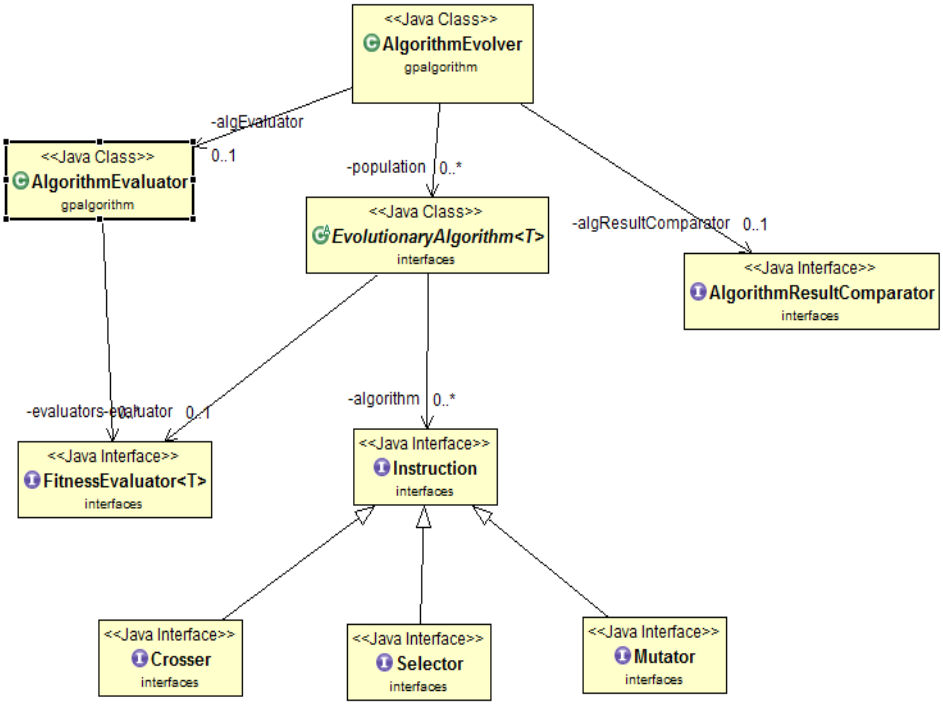
Sav kod napisan je u programskom jeziku Java. Imena razreda i metoda su na engleskom jeziku.

3.1. Programsko ostvarenje GP algoritma koristeći jednostavan prikaz evolucijskog algoritma

Implementiran je GP algoritam koji radi s evolucijskim algoritmom koji je predstavljen kao niz instrukcija oblika navedenog u poglavlju 2.4.

Na slici 3.1. prikazan je dijagram razreda programa. Centralna jedinica programa gdje GP algoritam provodi evoluciju evolucijskog algoritma je u razredu *AlgorithmEvolver*. On kao pomoćni razred koristi *AlgorithmEvaluator* (za izračunavanje dobrote pojedinog evolucijskog algoritma) i sučelje *AlgorithmResultComparator* (za uspoređivanje dobrota evolucijskih algoritama). Apstraktni razred koji predstavlja evolucijski algoritam nad kojim se provodi evolucija je *EvolutionaryAlgorithm<T>.* Taj razred korisnik mora dopuniti metodama koje definiraju detalje evolucijskog algoritma. Također, on koristi sučelja *FitnessEvaluator<T>* i *Instruction*. *FitnessEvaluator<T>* definira kako se izračunava dobrota jedinke u populaciji evolucijskog algoritma. To sučelje također korisnik mora ispuniti svojim razredom. Sučelje

Instruction predstavlja jednu instrukciju u algoritmu, kao što je objašnjeno u poglavlju 2. Ona može biti mutacija (sučelje *Mutator*), selekcija (sučelje *Selector*) ili križanje (sučelje *Crosser*). Korisnik mora također svojim razredima ispuniti zahtjeve ovog sučelja i mora ih koristiti pri implementaciji metoda razreda *EvolutionaryAlgorithm<T>*.



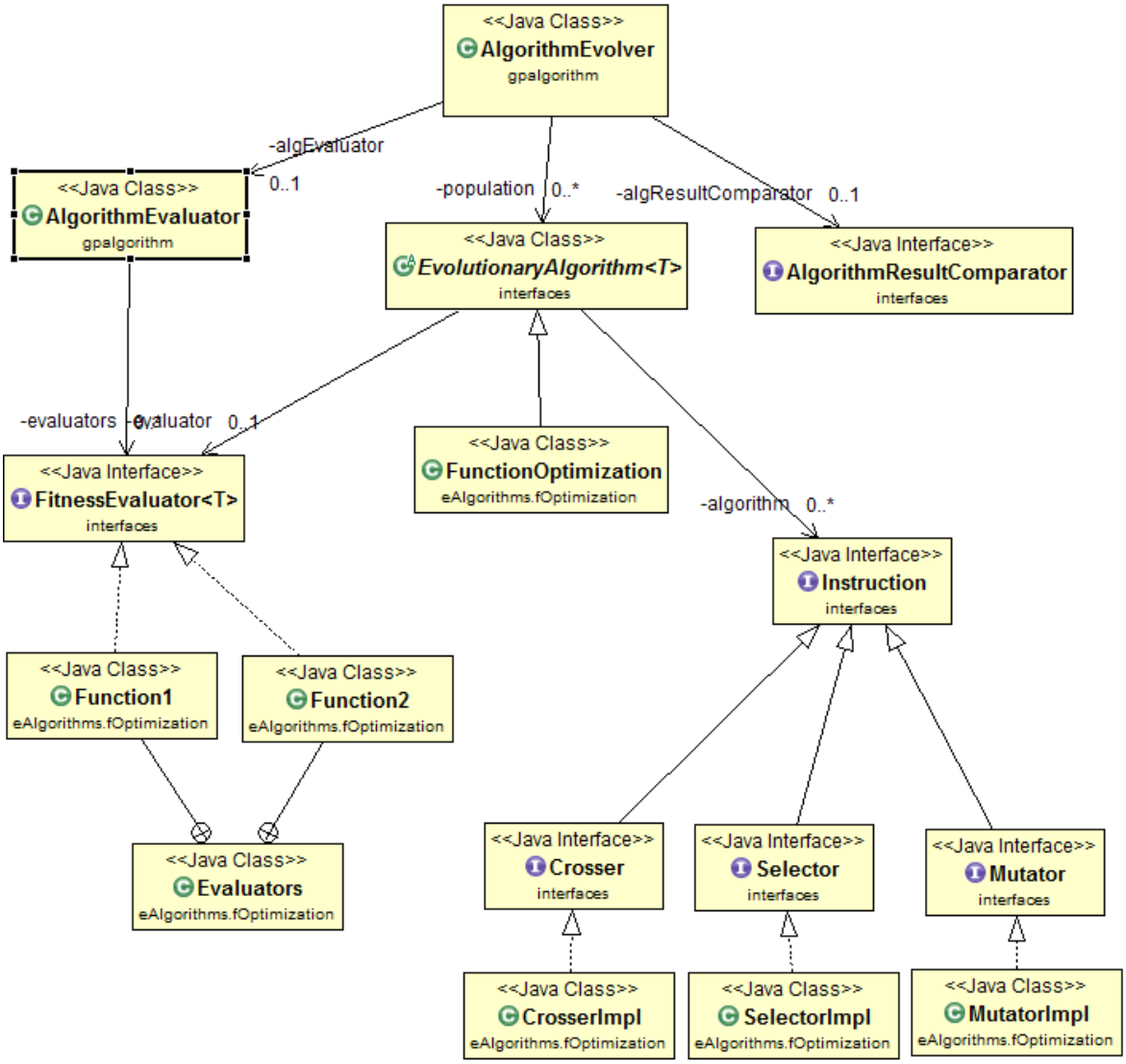
Slika 3.1. Dijagram razreda programa

Sve zajedno korisnik mora implementirati razrede koji zadovoljavaju sljedeća sučelja (EvolutionaryAlgorithm<T> nije sučelje, no korisnik svejedno mora definirati određene metode) :

- *EvolutionaryAlgorithm<T>*
- *FitnessEvaluator<T>*
- *Crosser*
- *Selector*
- *Mutator*

Na slici 3.2. prikazan je dijagram razreda programa nakon implementacije svih potrebnih razreda za ispitivanje cijele funkcionalnosti GP algoritma (tj. napravljen je i dio koji implementira korisnik). Razred *FunctionOptimization* nasljeđuje *EvolutionaryAlgorithm<T>* i implementira potrebne metode. Razredi *CrosserImpl*, *MutatorImpl* i *SelectorImpl* implementiraju sučelja *Crosser*, *Mutator* i *Selector*, respektivno. U statičkom razredu *Evaluators* su napravljeni razredi koji implementiraju sučelje *FitnessEvaluator<T>*. Na slici su prikazana samo dva takva razreda, *Function1* i *Function2* koji zapravo predstavljaju funkciju koju evolucijski algoritam pokušava optimizirati. Pri ispitivanju je korišteno više različitih funkcija.

Korisnik pri definiranju metoda razreda *EvolutionaryAlgorithm<T>* određuje i koje parametre želi evoluirati. Te parametre može koristiti pri implementaciji ostalih nužnih razreda i na taj način postići željeni učinak.



Slika 3.2. Dijagram razreda programa s implementiranim korisničkim razredima

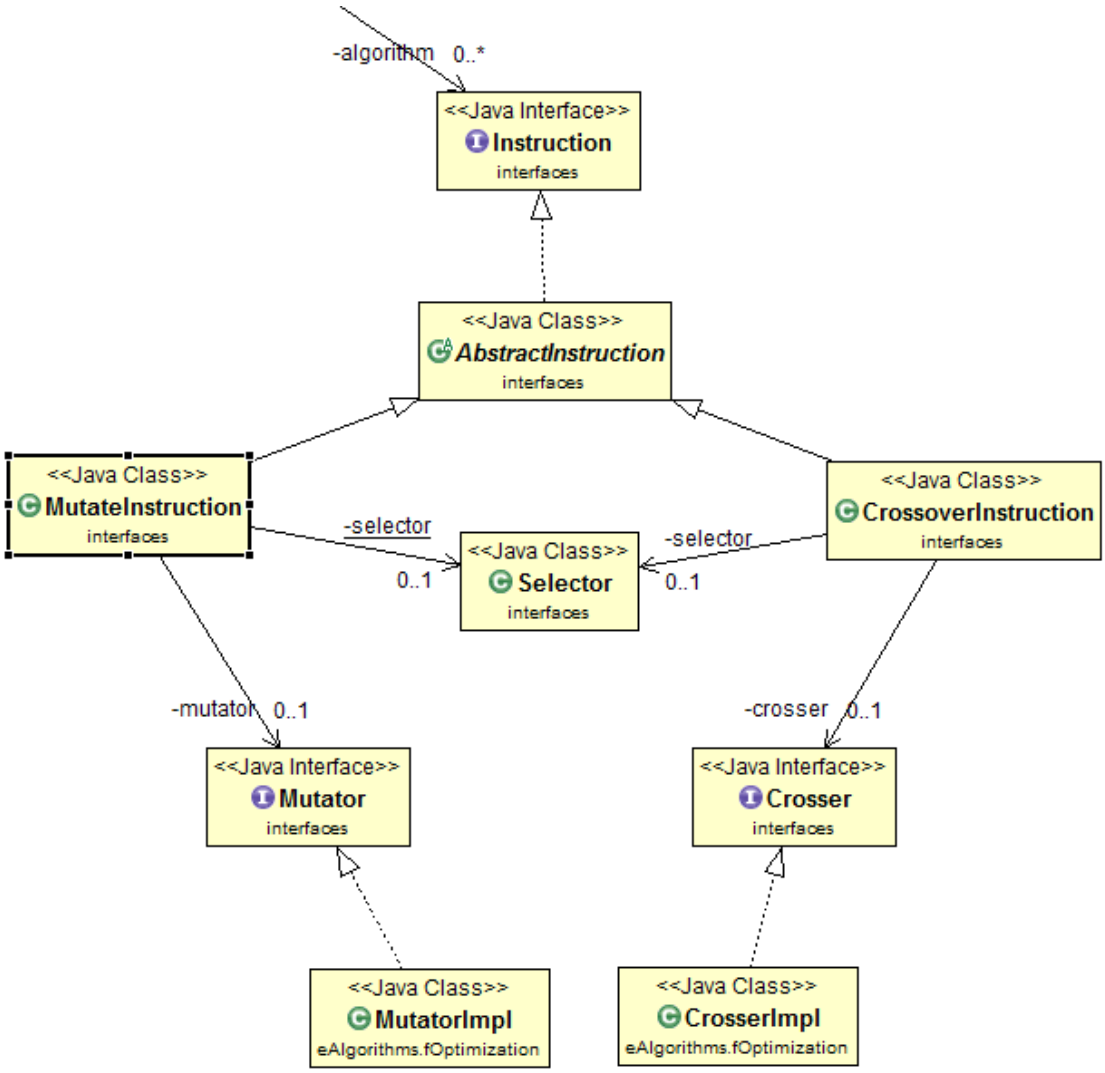
3.2. Programsko ostvarenje GP algoritma koristeći složeniji prikaz evolucijskog algoritma

Implementiran je GP algoritam koji vrši evoluciju nad evolucijskim algoritmom čije su instrukcije opisane u poglavlju 2.5. Cjelokupna struktura razreda vrlo je slična kao i ona na slici 3.1. koja odgovara GP algoritmu koji koristi jednostavan prikaz algoritma. Nadalje su opisane

samo razlike u implementaciji, sve što nije spomenuto ostaje jednako kao u poglavlju 3.1.

Na slici 3.3. prikazan je dijagram razreda kojime se postižu složenije instrukcije. Ostatak razrednog dijagrama je nepromijenjen i povezan sa sučeljem *Instruction* na jednak način. Ovdje su već prikazani razredi koje implementira onaj tko formulira evolucijski algoritam, teoretski korisnik, i to su razredi *MutatorImpl* i *CrosserImpl*. Sučelja *Mutator* i *Crosser* jednostavnija su nego prije, većina funkcionalnosti već je napisana u razredima *MutateInstruction* i *CrossoverInstruction*. U tim sučeljima potrebno je definirati samo način na koji se mutiraju i križaju jedinke, a time će se koristiti *MutateInstruction* i *CrossoverInstruction*. *Selector* više nije instrukcija, nego samo pomoćni razred kojim se vrši selekcija unutar instrukcija, kako je prethodno opisano.

U ovoj verziji GP algoritma nije omogućena evolucija parametara.



Slika 3.3. Dijagram razreda složenije instrukcije

3.3. Programsko ostvarenje evolucijskog algoritma za optimizaciju funkcija

U svrhu ispitivanja GP algoritma implementiran je algoritam za optimizaciju funkcija, prethodno spomenut kao razred *FunctionOptimization* koji nasljeđuje *EvolutionaryAlgoritm<T>*. On je neovisan o funkciji koju optimizira, ona je definirana kao način izračuna dobrote, tj. kao *FitnessEvaluator<T>*.

Jedinka koju algoritam evoluira je vektor. Pri inicijalizaciji populacije svi

su vektori popunjeni nasumičnim vrijednostima u određenom rasponu.

Budući da samo izvršavanje algoritma nije potrebno definirati, već je određeno u apstraktnom razredu *EvolutionaryAlgorithm<T>*, za algoritam optimizacije funkcija potrebni su još samo razredi koji opisuju instrukcije.

Ovisno radi li se o jednostavnom ili složenijem načinu prikaza instrukcija, na odgovarajući način implementiraju se razredi *CrosserImpl*, *MutatorImpl* i u prvom slučaju *SelectorImpl*. U njima se definira kako će se mutirati, križati i selektirati vektori. Mutacija je implementirana kao Gaussova mutacija, a parametar sigma se evoluira u slučaju jednostavnih instrukcija, u suprotnom je postavljen na $\sigma = 0.75$. Kod križanja je korišteno konveksno križanje (Oltean, 2005) s parametrom $\lambda = 0.5$. Selekcija odabire vektor s većom dobrotom.

4. Ispitivanje

Ispitivanje funkcionalnosti GP algoritma provedeno je koristeći implementirani evolucijski algoritam za optimizaciju funkcija. Korištene funkcije opisane su u tablici 4.1. Uvijek se traži minimum funkcije.

FUNKCIJA	*DOMENA*	*MINIMUM*
$f_1(x)=\sum_{i=1}^{n}(i*(x_i)^2)$	$[-10,10]^n$	$f_1(0,\ldots,0)=0$
$f_2(x)=\sum_{i=1}^{n}x_i^2$	$[-100,100]^n$	$f_2(0,\ldots,0)=0$
$f_3(x)=\sum_{i=1}^{n-1}100*(x_{i+1}-(x_i)^2)^2+(1-x_i)^2$	$[-30,30]^n$	$f_3(0,\ldots,0)=0$
$f_4=$ *Ackleyjeva funkcija* [1]	$[-32,32]^n$	$f_4(0,\ldots,0)=0$
$f_5=$ *Griewankova funkcija* [2]	$[-512,512]^n$	$f_5(0,\ldots,0)=0$
$f_6=$ *Lunacekova funkcija* [3]	$[-100,100]^n$	nepoznato
$f_7=\sum_{i=1}^{n}(\sum_{j=1}^{i}(x_j-j)^2)$	$[-10,10]^n$	$f_7(1,2,\ldots,n)=0$
$f_8=$ *Rastriginova funkcija* [4]	$[-5,5]^n$	$f_8(0,\ldots,0)=0$

Tablica 4.1. Funkcije za optimizaciju

Ispitivano je evoluiranje evolucijskog algoritma koji je primijenjen samo na jednu funkciju, ali i onoga kojemu se dobrota mjeri s obzirom na njegovu uspješnost na različitim funkcijama.

Za ocjenu i usporedbu dobivenih rezulata implementirana je jednostavna standardna verzija evolucijskog algoritma za optimizaciju funkcija.

1 http://www.cs.unm.edu/~neal.holts/dga/benchmarkFunction/ackley.html
2 http://www.cs.unm.edu/~neal.holts/dga/benchmarkFunction/griewank.html
3 http://www.cs.unm.edu/~neal.holts/dga/benchmarkFunction/lunacek.html
4 http://www.cs.unm.edu/~neal.holts/dga/benchmarkFunction/rastrigin.html

4.1. Standardni algoritam korišten za ocjenu rezultata

Implementiran je evolucijski algoritam koji koristi turnirsku selekciju, turnir veličine 3. Uzima dva najbolja sudionika turnira, križa ih, rezultat križanja mutira i krajnji rezultat stavlja na mjesto najgoreg sudionika turnira. Na ovaj algoritam od sad nadalje referira se uvijek kao na 'standardni algoritam'.

Pseudokod standardnog algoritma:

```
1. standardniAlgoritam() {
2.       populacija <- inicijalizirajPopulaciju();
3.       dok je (brojIzračunaDobrote < KONST) {
4.            održiTurnir(populacija);
5.       }
6.       vrati nađiNajbolji(populacija);
7. }
```

Pseudokod funkcije *održiTurnir*:

```
1. održiTurnir(najboljiGlobalno, populacija) {
2.       sudionici <- odaberiSudionike(populacija);
3.       (najgori, bolji, najbolji) <-
   sortirajPoDobroti(sudionici);
4.       najgori <- mutiraj(križaj(bolji, najbolji));
5.       }
```

Parametri standarnog algoritma pri svakoj će usporedbi biti namješteni

tako da budu jednaki ili što sličniji onima koje će imati evolucijski algoritam koji se evoluira.

4.2. Ispitivanje koristeći evolucijski algoritam primijenjen na jednu funkciju

Rezultati ispitivanja izneseni u sljedeća dva potpoglavlja dobiveni su na ovaj način: GP algoritmu predan je evolucijski algoritam s jednim načinom izračuna dobrote, a taj je način jedna od funkcija iz tablice 4.1. Dakle, za svaku funkciju kojoj se traži minimum evolucijom dobiven je poseban evolucijski algoritam i predstavljeni su njegovi rezultati.

Uz rezultate evoluiranog evolucijskog algoritma, za usporedbu su prikazani rezultati standardnog algoritma i evolucijskog algoritma koji je dobiven nasumičnim odabirom instrukcija, u trenutku kada GP algoritam stvara populaciju evolucijskih algoritama. Razlika između tog rezultata i rezultata evoluiranog evolucijskog algoritma govori o tome koliki je konačan učinak evolucije, koliko je najbolji dobiveni algoritam poboljšan u odnosu na početni.

4.2.1. Ispitivanje koristeći jednostavan prikaz evolucijskog algoritma

Za svaku funkciju iz tablice 4.1. provedena su dva ispitivanja koja se razlikuju u veličini populacije i broju turnira (evaluacija) koje je održao GP algoritam. Parametri dani u tablici 4.3. jednaki su u oba ispitivanja. Korištena je verzija GP algoritma koja radi s evolucijskim algoritmom s jednostavnim instrukcijama, opisanim u poglavlju 2.4.

Prvo ispitivanje:

Pri prvom ispitivanju, namješteni su nešto manji iznosi parametara GP algoritma u tablici 4.2., nego u drugom ispitivanju. U tablici 4.3.

navedeni su parametri evolucijskog i standardnog algoritma i ti podaci ostaju jednaki i za drugo ispitivanje. Broj izračuna dobrote jednike u tim algoritmima određuje trajanje izvođenja algoritma. Kad se dosegne postavljen broj, algoritam se prekida i samo se još jednom prije kraja u populaciji potraži nova najbolja jedinka. To znači da je postavljen broj izračuna dobrote jednike manji od stvarnog, s malom razlikom. Izračun dobrote jednike vrši se samo kada je potreban i jednom izračunata vrijednost dobrote jedinke se pamti sve dok je jedinka nepromijenjena.

Iz ponovnih pokretanja mnogih testova, zaključeno je kako rezultati dobivenih evolucijskih algoritama mogu znatno varirati. Odstupanja ovise o veličini konačnog rezultata. Što je rezultat bliži stvarnom minimumu funkcije, to su veća i odstupanja koja mogu biti i preko 100%.

Parametar	*Iznos*
Veličina populacije	50
Broj turnira	200

Tablica 4.2. Parametri GP algoritma

Parametar	*Evolucijski alg.*	*Standardni alg.*
Veličina popluacije	20	20
Broj izračuna dobrote jedinke	200	200
Veličina vektora (n)	5	5
Parametar Gaussove mutacije σ	Početni 0.75, podložan evoluciji	0.75
Parametar konveksnog križanja λ	0.5	0.5
Broj pokretanja prije uzimanja medijana kao ukupnog rezultata	200	200

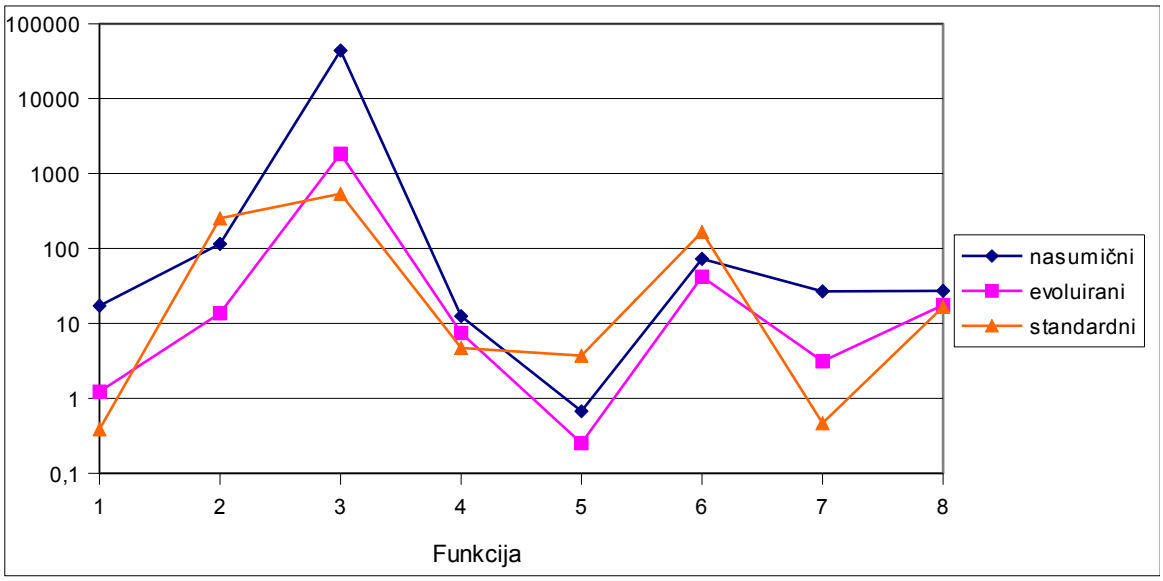
Tablica 4.3. Parametri evolucijskog i standardnog algoritma

U tablici 4.4. predstavljeni su dobiveni rezultati prvog ispitivanja.

Rezultati su zapravo dobrote algoritama, tj. medijan 200 rezultata dobivenih uzastopnim izvođenjem algoritma. Iz njih se prvo može zaključiti da je postupak evolucije uspješan barem u određenoj mjeri, rezultati evoluiranog algoritma uvijek su vidljivo bolji od rezultata nasumično stvorenog algoritma. Teže je usporediti evoluirani algoritam sa standardnim, za neke funkcije bolji je jedan, za neke drugi. Očito evolucijom stvoren algoritam može biti usporediv sa standardnim.

Funkcija	*Evolucijski algoritam s nasumičnim instrukcijama*	*Evoluirani evolucijski algoritam*	*Standardni algoritam*
f_1	17.186	1.2264	0.38650
f_2	115.12	13.654	252.99
f_3	44054	1846.5	538.94
f_4	12.521	7.4748	4.6945
f_5	0.67240	0.25180	3.7074
f_6	72.84	41.927	166.90
f_7	26.722	3.1250	0.46647
f_8	27.148	17.644	16.605

Tablica 4.4. Rezultati prvog ispitivanja



Slika 4.1. Rezultati prvog ispitivanja

Na slici 4.1. grafički su prikazani rezultati prvog ispitivanja.

Drugo ispitivanje:

Sva pravila i podaci ostaju isti kao i u prvom ispitivanju, osim parametara GP algoritma. Oni korišteni za ovo ispitivanje su prikazani u tablici 4.5. Ovim parametrima pokušava se izvući maksimalan utjecaj GP algoritma. S obzirom na vrlo velik broj turnira, trajanje izvođenja za svaku funkciju je znatno, proteže se i do par sati.

Parametar	*Iznos*
Veličina populacije	500
Broj turnira	20000

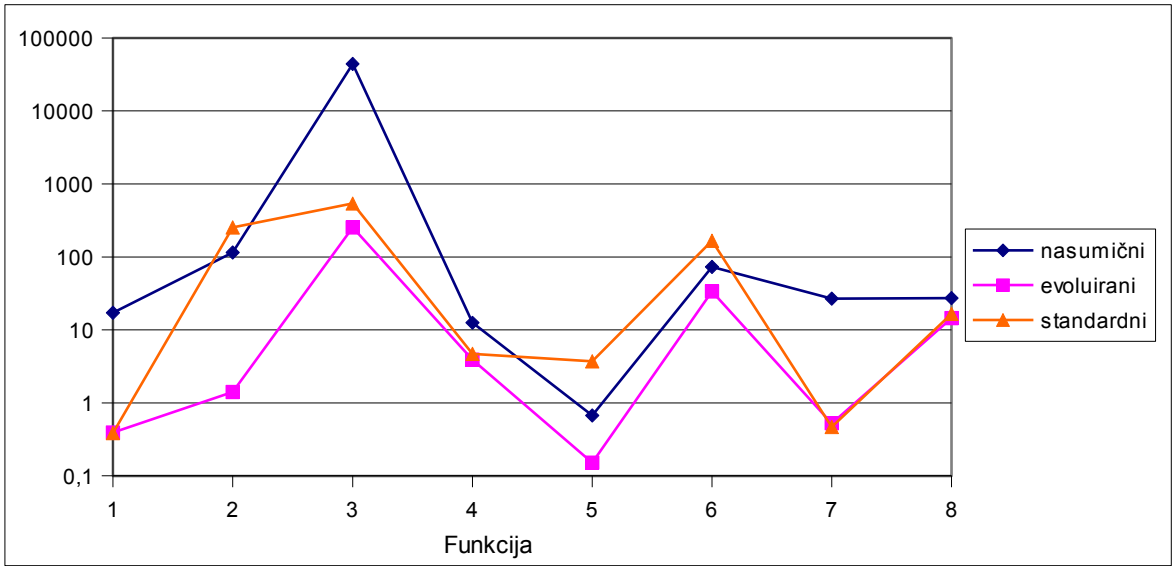
Tablica 4.5. Novi parametri GP algoritma

Funkcija	*Evolucijski algoritam s nasumičnim instrukcijama*	*Evoluirani evolucijski algoritam*	*Standardni algoritam*
f_1	17.186	0.39110	0.38650
f_2	115.12	1.4133	252.99
f_3	44054	254.25	538.94
f_4	12.521	3.8941	4.6945
f_5	0.67240	0.15210	3.7074
f_6	72.84	33.568	166.90
f_7	26.722	0.52401	0.46647
f_8	27.148	14.517	16.605

Tablica 4.6. Rezultati drugog ispitivanja

Rezultati drugog ispitivanja prikazani su u tablici 4.6. Rezultati standardnog algoritma preuzeti su iz tablice 4.4. jer se niti jedan parametar tog algoritma nije promijenio. Preuzeti su i rezultati evolucijskog algoritma s nasumičnim instrukcijama jer su parametri evolucijskog algoritma također nepromijenjeni. Ovdje se vidi kako evoluirani evolucijski algoritam za skoro svaku funkciju postiže bolje rezultate od standardnoga. GP algoritam uspješan je u dobivanju

efikasnijeg evloucijskog algoritma. Grafički prikaz rezultata je na slici 4.2.



Slika 4.2. Graf rezultata drugog ispitivanja

4.2.2. Ispitivanje koristeći složeniji prikaz evolucijskog algoritma

Ispitivanje GP algoritma koji koristi evolucijski algoritam sa složenijom verzijom instrukcija (opisanih u poglavlju 2.5.) također je provedeno s dvije različite postavke parametara. U tablici 4.7. prikazani su parametri evolucijskog i standardnog algoritma koji je korišten i oni su nepromijenjeni za oba ispitivanja.

Prvo ispitivanje:

Parametri GP algoritma korišteni za prvo ispitivanje prikazani su u tablici 4.2., dakle jednaki su kao i oni za prvo ispitivanje GP algoritma koji koristi evolucijski algoritam s jednostavnim instrukcijama.

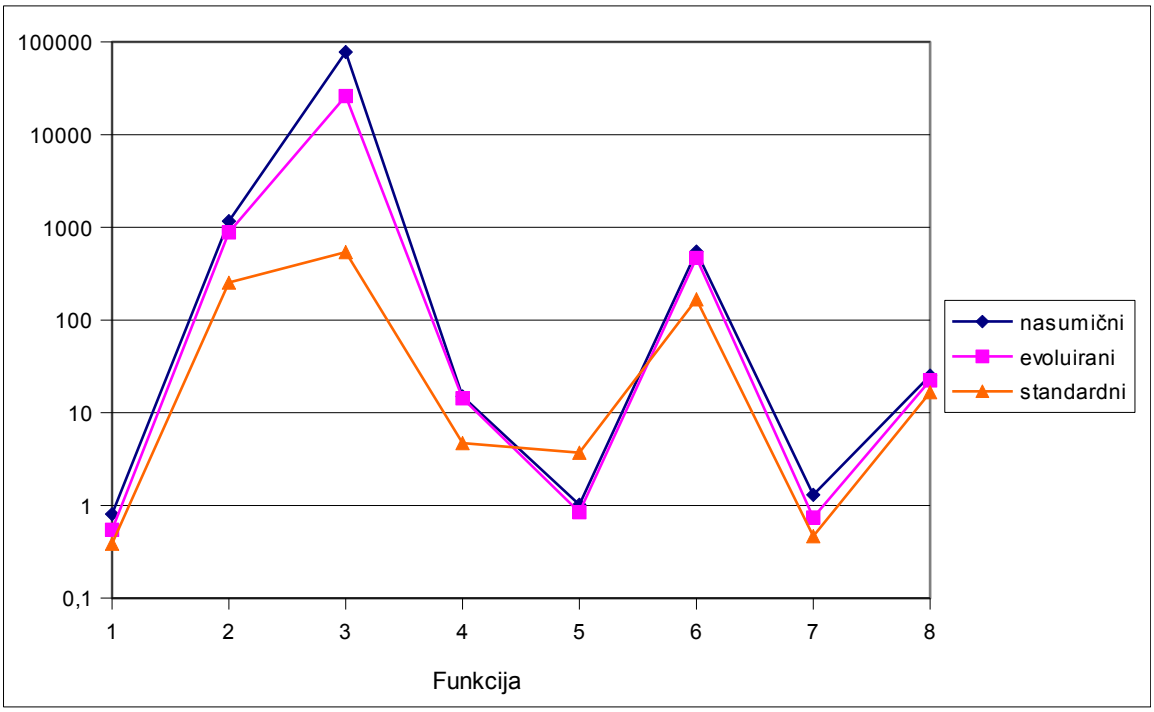
Parametar	*Evolucijski alg.*	*Standardni alg.*
Veličina popluacije	20	20
Broj izračuna dobrote jedinke	200	200
Veličina vektora (n)	5	5
Parametar Gaussove mutacije σ	0.75	0.75
Parametar konveksnog križanja λ	0.5	0.5
Broj pokretanja prije uzimanja medijana kao ukupnog rezultata	200	200

Tablica 4.7. Parametri evolucijskog i standarnog algoritma

Rezultati prvog ispitivanja prikazani su u tablici 4.8. Primjećuje se slabiji učinak evolucije, evoluirani algoritam uvijek je bolji od nasumičnog, ali često razlika nije velika. Evoluirani algoritam daje većinom lošije rezultate od standardnog, ali ponovo se može zaključiti da se evoluirani algoritam može natjecati sa standardnim, rezultati su većinom istog reda veličine. Rezultati su grafički prikazani na slici 4.3.

Funkcija	*Evolucijski algoritam s nasumičnim instrukcijama*	*Evoluirani evolucijski algoritam*	*Standardni algoritam*
f_1	0.80541	0.54475	0.38650
f_2	1165.3	878.85	252.99
f_3	77884	26018	538.94
f_4	15.054	14.301	4.6945
f_5	1.0199	0.84480	3.7074
f_6	546.32	468.47	166.90
f_7	1.3007	0.74149	0.46647
f_8	25.221	22.325	16.605

Tablica 4.8. Rezultati prvog ispitivanja



Slika 4.3. Graf rezultata prvog ispitivanja

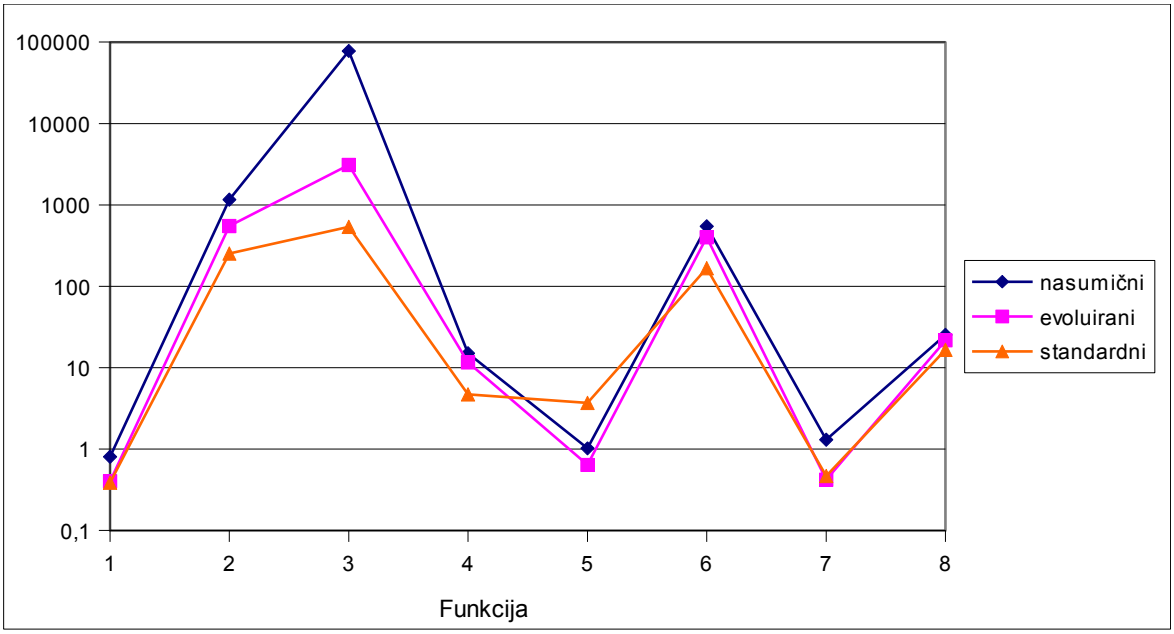
Drugo ispitivanje:

Za drugo ispitivanje su parametri GP algoritma ponovo jednaki kao i oni korišteni za drugo ispitivanje GP algoritma koji koristi evolucijski algoritam s jednostavnim instrukcijama prikazani u tablici 4.5. I u ovom je slučaju trajanje izvođenja bilo znatno.

Funkcija	*Evolucijski algoritam s nasumičnim instrukcijama*	*Evoluirani evolucijski algoritam*	*Standardni algoritam*
f_1	0.80541	0.40197	0.38650
f_2	1165.3	548.85	252.99
f_3	77884	3100.9	538.94
f_4	15.054	11.643	4.6945
f_5	1.0199	0.63930	3.7074
f_6	546.32	399.52	166.90
f_7	1.3007	0.42102	0.46647
f_8	25.221	21.687	16.605

Tablica 4.9. Rezultati drugog ispitivanja

Rezultati drugog ispitivanja prikazani su u tablici 4.9. Rezultati standardnog algoritma preuzeti su iz tablice 4.8. jer se niti jedan parametar tog algoritma nije promijenio. Preuzeti su i rezultati evolucijskog algoritma s nasumičnim instrukcijama jer su parametri evolucijskog algoritma također nepromijenjeni. Pokazuje se kako dobiveni evolucijski algoritam ovdje očekivano daje bolje rezultate nego u prethodnom ispitivanju, međutim napredak nije znatan. Standardni je algoritam i dalje bolji u većini slučajeva. Očito su mogućnosti evolucije evolucijskog algoritma s ovakvim instrukcijama ograničene. Graf rezultata drugog ispitivanja je na slici 4.4.



Slika 4.4. Graf rezultata drugog ispitivanja

4.2.3. Usporedba rezultata korištenja jednostavnog i složenijeg prikaza evolucijskog algoritma

Pažnja je usmjerena na rezultate drugog ispitivanja gdje je GP algoritam dugo evoluirao evolucijski algoritam. Iz rezultata proizlazi da je jednostavan prikaz algoritma uspješniji, ima uvjerljivo bolje rezultate od složenog prikaza. Također, kod tih algoritama značajnija je razlika između nasumično stvorenog algoritma i onog dobivenog evolucijom što govori o tome da je podložniji evlouciji. Algoritam s jednostavnim prikazom ima i većinom bolje rezultate od standardnog algoritma, dakle pokazuje da se ideja evolucije algoritma može uspješno provesti.

U nekima od evolucijom dobivenih evolucijskih algoritama s jednostavnim instrukcijama primjećeno je kako je rijetka instrukcija selekcije, iz čega se može zaključiti da je algoritmu korisnija mutacija ili križanje. Time evolucija još smanjuje selekcijski pritisak. Za razliku od složenijeg prikaza algoritma gdje je je GP algoritmu dozvoljeno regulirati selekcijski pritisak samo u broju članova populacije koji će

svaki put sudjelovati u selekciji, a on može biti samo između 3 i 6, ovdje GP algoritam s brojem instrukcija selekcije očito može bolje regulirati selekcijski pritisak. Ovo je jedna znatna prednost jednostavnog prikaza algoritma, uočena tek poslije ispitivanja.

4.3. Ispitivanje koristeći evolucijski algoritam primijenjen na više funkcija

U ovom poglavlju GP algoritam ocjenjuje evolucijski algoritam uzimajući u obzir njegovu uspješnost na različitim funkcijama. Ideja je učiti evolucijski algoritam na nekim funkcijama i onda tako dobiveni algoritam primijeniti na neku drugu funkciju. Zatim se ti rezultati uspoređuju s prethodno dobivenim rezultatima kada je evolucijski algoritam bio razvijan baš za tu funkciju.

Ovim načinom istražuje se u kolikoj mjeri evolucijski algoritam mora biti specifičan za neki problem (funkciju), i isplati li se na skupu sličnih problema koristiti ovu metodu razvoja algoritma. Kod realizacije evolucijskih algoritama za realne probleme bi ovaj pristup bio vrlo primjenjiv: evolucijski algoritam se evoluira na jednostavnijim problemima za koje znamo rješenja (možemo u nekoj mjeri procijeniti koliko je dobar) i zatim se primjeni na stvaran problem.

4.3.1. Ispitivanje koristeći jednostavan prikaz evolucijskog algoritma

Provedena su tri ispitivanja, za svako od njih su korišteni parametri GP algoritma iz tablice 4.2. i parametri evolucijskog algoritma iz tablice 4.3. Evolucijski algoritam koristi jednostavne instrukcije.

Prvo ispitivanje:

GP algoritam razvio je evolucijski algoritam koristeći funkcije f_1 i f_2

za ocjenu algoritma, dakle evolucijski algoritam učen je na ove dvije funkcije. Zatim je primijenjen na funkcije f_3 i f_4. Rezultati su prikazani u tablici 4.10.

FUNKCIJA	*EVOLUIRANI EVOLUCIJSKI ALGORITAM*	*EVOLUIRANI EVOLUCIJSKI ALGORITAM ZA POJEDINAČNU FUNKCIJU*
f_1	1.7751	1.2264
f_2	19.509	13.654
f_3	2569.2	1846.5
f_4	7.1032	7.4748

Tablica 4.10. Rezultati prvog ispitivanja

Algoritam dobiven evolucijom na f_1 i f_2 ima nešto lošije rezultate za te funkcije od algoritama koji su posebno razvijeni za svaku od tih funkcija (tablica 4.4.), što je očekivano. Zanimljivo, zatim taj algoritam ima bolje rezultate za funkciju f_4 od onog razvijenog posebno za tu funkciju. Iako je teško preciznije uspoređivati ove rezultate dobivene za manje parametre GP algoritma zbog mogućih odstupanja, može se zaključiti da evolucija na prve dvije funkcije uspješno uči algoritam kako bi bio dobar i na trećoj i četvrtoj funkciji, što se vidi i usporedbom ovih rezultata s rezultatima nasumičnog algoritma za treću i četvrtu funkciju.

Drugo ispitivanje:

GP algoritam ovdje je evoluirao evolucijski algoritam s obzirom na funkcije f_1, f_2, f_3 i f_4, i zatim je primijenjen na funkcije f_5 i f_7. Rezultati su prikazani u tablici 4.11.

FUNKCIJA	EVOLUIRANI EVOLUCIJSKI ALGORITAM	EVOLUIRANI EVOLUCIJSKI ALGORITAM ZA POJEDINAČNU FUNKCIJU
f_1	1.0382	1.2264
f_2	34.587	13.654
f_3	2992.8	1846.5
f_4	7.5749	7.4748
f_5	0.31318	0.25180
f_7	3.0564	3.1250

Tablica 4.11. Rezultati drugog ispitivanja

Rezultati su slični kao i u prethodnom ispitivanju, algoritam je uspješno učen za optimizaciju funkcija f_5 i f_7. Rezultati za funkcije na kojima je učen su ponovo većinski lošiji od rezultata algoritama koji su posebno evoluirani za svaku od tih funkcija (tablica 4.4.).

Treće ispitivanje:

U ovom slučaju GP algoritam evoluiran je s obzirom na funkcije f_1, f_2, f_3, f_4, f_5, f_7 i f_8. Zatim je primijenjen na funkciju f_6. Rezultati su prikazani u tablici 4.12.

FUNKCIJA	EVOLUIRANI EVOLUCIJSKI ALGORITAM	EVOLUIRANI EVOLUCIJSKI ALGORITAM ZA POJEDINAČNU FUNKCIJU
f_1	1.8697	1.2264
f_2	17.057	13.654
f_3	3503.9	1846.5
f_4	7.8749	7.4748
f_5	0.25484	0.25180
f_7	6.8948	3.1250
f_8	20.475	17.644
f_6	45.172	41.927

Tablica 4.12. Rezultati trećeg ispitivanja

Ovdje je pokušano dobiti dobar rezultat za jednu od kompliciranijih funkcija tako što se prethodno algoritam uči na svim drugim funkcijama. Rezultat je dobar, budući da je ovaj rezultat za funkciju f_6 sličan, iako lošiji, onom algoritma koji je razvijen samo za tu funkciju (tablica 4.4.).

4.3.2. Ispitivanje koristeći složeniji prikaz evolucijskog algoritma

Kao i u prethodnom poglavlju provedena su tri ispitivanja. GP algoritam ima parametre iz tablice 4.2., a evolucijski algoritam iz tablice 4.7. Korišten je složeniji oblik instrukcija za evolucijski algoritam. Budući da su praktično svi parametri jednaki kao i u 4.3.1. i u svakom ispitivanju korištene su iste funkcije kao i u odgovarajućem ispitivanju u 4.3.1. dalje su odmah navedeni rezultati.

Prvo ispitivanje:

FUNKCIJA	*EVOLUIRANI EVOLUCIJSKI ALGORITAM*	*EVOLUIRANI EVOLUCIJSKI ALGORITAM ZA POJEDINAČNU FUNKCIJU*
f_1	0.58126	0.54475
f_2	947.55	878.85
f_3	20057	26018
f_4	14.255	14.301

Tablica 4.13. Rezultati prvog ispitivanja

Rezultati iz tablice 4.13. pokazuju kako su rezultati algoritma razvijenog na f_1 i f_2, te zatim primjenjenog na f_3 i f_4 za svaku od tih funkcija gotovo jednaki kao i rezultati algoritama koji su razvijeni za svaku od tih funkcija posebno (tablica 4.8.).

Drugo ispitivanje:

Iz rezultata u tablici 4.14. zaključuje se da učenje nije uspjelo za funkciju f_5 čiji rezultat optimizacije je lošiji nego nekog nasumičnog algoritma (tablica 4.8.), ali je dobar rezultat ostvaren za funkciju f_7 , nešto bolji od algoritma razvijenog samo za optimizaciju te funkcije.

FUNKCIJA	*EVOLUIRANI EVOLUCIJSKI ALGORITAM*	*EVOLUIRANI EVOLUCIJSKI ALGORITAM ZA POJEDINAČNU FUNKCIJU*
f_1	0.49862	0.54475
f_2	921.43	878.85
f_3	18455	26018
f_4	13.672	14.301
f_5	1.1342	0.84480
f_7	0.64189	0.74149

Tablica 4.14. Rezultati drugog ispitivanja

Treće ispitivanje:

FUNKCIJA	*EVOLUIRANI EVOLUCIJSKI ALGORITAM*	*EVOLUIRANI EVOLUCIJSKI ALGORITAM ZA POJEDINAČNU FUNKCIJU*
f_1	0.61889	0.54475
f_2	867.44	878.85
f_3	27243	26018
f_4	14.411	14.301
f_5	0.82224	0.84480
f_7	0.87189	0.74149
f_8	24.029	22.325
f_6	457.02	468.47

Tablica 4.15. Rezultati trećeg ispitivanja

U tablici 4.15. rezultati ispitivanja pokazuju da je, kao i u slučaju s jednostavnim prikazom algoritma, učenje evolucijskog algoritma na svim funkcijama osim f_6 i zatim primjene na f_6 uspješno. Dobiven je čak i bolji rezultat od evolucijskog algoritma razvijenog samo za optimizaciju funkcije f_6 (tablica 4.8.).

4.3.3. Analiza rezultata dobivenih učenjem evolucijskog algoritma

Iz navedenih rezultata u prethodna dva potpoglavlja moguće je zaključiti kako je strategija učenja evolucijskog algoritma na nekim funkcijama i zatim primjene na neke druge funkcije većinski uspješna. Rezultati su uvijek bili istog reda veličine kao i oni dobiveni evolucijom algoritma samo za jednu funkciju. To sugerira da je optimizacija funkcija iz tablice 4.1. jedna vrsta problema, tj. da isti evolucijski algoritam može dobro raditi za optimizaciju svake od tih funkcija.

5. Zaključak

U ovom radu pokazano je kako je moguće osmisliti i implementirati evoluciju evolucijskih algoritama s obećavajućim rezultatima. Evolucija je uspješno usmjeravala algoritam tako da postaje sve bolji za probleme koje pokušava riješiti. Time se može postići željeno prilagođavanje algoritma problemu spominjano u uvodu.

Postoji mogućnost prilagodbe razvijenog algoritma kako bi mogao evoluirati evolucijski algoritam specificiran ECDL-om (Evolutionary Computation Description Language) (Đurasević, 2012).

Ovo područje ima puno potencijala za daljnje istraživanje. Razvijen GP algoritam, kao i način predstavljanja evolucijskog algoritma mogu se poboljšati. S obzirom na činjenicu da GP algoritam može vrlo slabo regulirati selekcijski pritisak evolucijskog algoritma složenijeg prikaza, moguće je uvesti instrukcije koje se izvršavaju bez prethodne selekcije te tako napraviti mješavinu dviju predstavljnih prikaza. Također, sve ovdje predstavljene instrukcije evolucijskog algoritma imale su nepromjenjive indekse. Umjesto toga, mogu se uvesti indeksi kao varijable, možda ovisni o trenutnom stanju populacije. Uvjetno izvođenje instrukcija još je jedna mogućnost. Cilj ovih predloženih poboljšanja je davanje veće slobode evolucijskom algoritmu, pokušaj da se što više mogućih činitelja uzme u obzir.

Korišteni GP algoritam vrlo je jednostavan. On se također može unaprijediti na razne načine koji danas služe poboljšavanju evolucijskih algoritama. Međutim njegove mogućnosti uvijek će biti ograničene onoliko koliko je evolucijski algoritam koji se nastoji evoluirati ograničen vlastitim prikazom.

6. Literatura

Wolpert, Macready, 1996: David H. Wolpert, William G. Macready, No Free Lunch Theorems for Optimization, 1996, IEEE Transactions on Evolutionary Computation

Edmonds, 2001: Bruce Edmonds, Meta-Genetic Programming: Co-evolving the Operators for Variation, 2001, Tubitak Journal

Oltean, 2005: Mihai Oltean, Evolving Evolutionary Algorithms using Linear Genetic Programming, 2005, Evolutionary Computation, MIT Press

Oltean, 2003: Mihai Oltean, Crina Grosan, Evolving Evolutionary Algorithms using Multi Expression Programming, 2003, The 7-th European Conference on Artificial Life

Oltean, 2007: Mihai Oltean, Evolving Evolutionary Algorithms with Patterns, 2007, Soft Computing

Xie, 2012: Huayang Xie and Mengjie Zhang, Tuning Selection Pressure in Tournament Selection, 2012, IEEE Transactions on Evolutionary Computation

Đurasević, 2012: Marko Đurasević, Metajezik za Opis Evolucijskih Algoritama, 2012, završni rad, Fakultet elektrotehnike i računarstva, Zagreb

Naslov: Evolucija evolucijskih algoritama

Sažetak:

Objašnjena je ideja evolucije evolucijskih algoritama linearnim genetskim programiranjem. Evolucijski algoritam predstavljen je kao niz instrukcija na dva načina. Razvijen je algoritam genetskog programiranja koji evoluira evolucijski algoritam s obzirom na to koliko dobro evolucijski algoritam rješava problem za koji je napravljen. Navedni algoritam ostvaren je u programskom jeziku Java. Program je ispitan na primjeru evolucijskog algoritma za optimizaciju funkcija te rezultati pokazuju da je ovako razvijen evolucijski algoritam usporediv, ponekad i bolji, od standardnog ručno napisanog evolucijskog algoritma.

Ključne riječi: evolucija evolucijskih algoritama, evolucijski algoritam, linearno genetsko programiranje, optimizacija funkcija

Title: Evolving evolutionary algorithms

Abstract:

The idea of evolving evolutionary algorithms using linear genetic programming is explained. Evolutionary algorithm is represented as a list of instructions in two ways. Genetic programming algorithm, which evolves an evolutionary algorithm taking into consideration how well the evolutionary algorithm solves the problem it was developed for, is developed. It is implemented in Java programming language. The program is tested using the evolutionary algorithm for function optimization and results show that evolutionary algorithm developed in this way has results similar, or sometimes better, than the standard manually written evolutionary algorithm.

Key words: evolving evolutionary algorithms, evoutionary algorithm, linear genetic programming, function optimization