SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD broj 517.

Optimizacijski algoritmi zasnovani na klonskoj selekciji

Krešimir Đuretec
voditelj: doc. dr. sc. Marin Golub

Zagreb , lipanj 2009

Sažetak

Optimizacijski algoritmi zasnovani na klonskoj selekciji

Umjetni imunološki sustavi su nova grana računalne inteligencije koja svoju inspiraciju pronalazi u imunološkom sustavu (točnije imunološkom sustavu kralježnjaka). Područje istraživanja umjetnih imunoloških sustava se dijeli na četiri grane (negativna selekcija, klonska selekcija, umjetne imunološke mreže i teorija opasnosti), od kojih je jedna (klonska selekcija) opisana u ovom radu.
Predstavljeni algoritmi temelje rad na klonskoj selekciji (mehanizam kojim se imunološki sustav bori protiv virusa i bakterija). Zbog velike sličnosti s genetskim algoritmima, algoritmi se mogu koristiti za optimizacijske probleme. Obrađena su četiri algoritma i testirani na problemu aproksimacije točaka određenom funkcijom. Također je u jeziku C++ implementirano radno okruženje za obrađene algoritme.

ključne riječi: umjetni imunološki sustavi, klonska selekcija, optimizacijski problemi, aproksimacijski problem, radno okruženje

Abstract

Optimization Algorithms based on Clonal Selection Principle

Artificial immune systems is a new field of computational intelligence based on immune system (more precisely vertebrate immune systems). Area of research is divided to four subfields (negative selection, clonal selection, artificial immune networks and danger theory). The subject of research in this paper is only algorithms based on clonal selection principle. Because of similarities with genetic algorithms, algorithms based on clonal selection principle are suitable for solving optimization problems. This paper presents four algorithms which are applied to approximation problem . Framework is implemented in C++ for testing presented algorithms.

key words: artificial immune systems, clonal selection, optimization problems, approximation problem, framework

Sadržaj

1. Uvod

Računalna inteligencija (eng. *Computational intelligence*) je grana umjetne inteligencije, područja koje je danas najistraživanije na području računarske znanosti i svakim danom se dolazi do novih spoznaja.
Algoritmi računalne inteligencije su zasnovani na heuristici, metodi koja pokušava doći do optimalnog rješenja na što brži način koristeći neke spoznaje o mogućim rješenjima. Heuristički algoritmi se koriste za probleme za koje ne postoji algoritam (ili još nije pronađen) koji u stvarnom vremenu daje rezultat. Možda ne pronalaze optimalno rješenje, dapače ne postoji dokaz (za većinu heurističkih algoritama) kojim bi se pokazalo da to rješenje nije proizvoljno loše, ali se najčešće kod rješavanja teških problema (za koje se i koristi heuristika) i ne traži optimalno rješenje , nego "dovoljno dobro" rješenje. Za tu svrhu su se heuristički algoritmi pokazali kao jako dobar alat. Također velika prednost im je općenitost tj. primjenjivi su na veći skup problema.
Unutar računalne inteligencije postoji nekoliko područja : evolucijsko računanje (eng. *evolutionary computation*), neuronske mreže(eng. *neural networks*) i neizraziti sustavi(eng. *fuzzy systems*). Iako se područja proučavaju zasebno, moguće su interakcije među njima.
Evolucijsko računanje je inspirirano prirodnom evolucijom. Proučavajući prirodnu evoluciju bilo je samo pitanje vremena kada će se tako nešto pokušati primijeniti u računarstvu. Začuđujuće, evolucijsko računanje se pojavilo 40-tih godina 20. stoljeća kada se počinje razvijati i samo moderno računarstvo. Do danas je razvijen veliki broj algoritama koji se temelje na evolucijskom računanju. Štoviše, svoj rad ne temelje samo na ideji evolucije iz prirode, nego i na nekim drugim zanimljivim prirodnim pojavama koje pokazuju određeni stupanj inteligencije (npr. inteligencija rojeva). Evolucijsko računanje se može podijeliti u tri cjeline: evolucijski algoritmi, algoritmi temeljeni na inteligenciji rojeva i ostalo.
Jedna od metoda koja spada pod ostale algoritme su i umjetni imunološki sustavi (eng. *Artificial Immune Systems*).

U ovom radu biti će predstavljeni ukratko umjetni imunološki sustavi kao nova metoda koja se može primijeniti na široki skup problema koji zahtijevaju od strojnog učenja pa sve do optimizacije. Također biti će predstavljene četiri grane (negativna selekcija, klonska selekcija, umjetne imunološke mreže i teorija opasnosti) umjetnih imunoloških sustava koje se mogu izučavati zasebno. Detaljnije će biti obrađena samo jedna grana(klonska selekcija) koja se također može podijeliti na podgrane (algoritme).
Cilj programskog dijela rada je izrada proširivog radnog okruženja za algoritme zasnovane na klonskoj selekciji. Uz pomoć izrađenog radnog okruženja riješiti određeni optimizacijski problem (aproksimacijski problem) i ispitati rad pojedinog algoritma.

2. Umjetni imunološki sustavi

2.1. Imunološki sustavi u prirodi

Jedan od zanimljivijih sustava u ljudskom[1] organizmu je svakako imunološki sustav. Imunološki sustav je visoko paralelni sustav koji ima mogućnost adaptacije i učenja. Glavni cilj imunološkog sustava je prepoznavanje i uklanjanje stranih organizama (tijela) iz tijela domaćina. Postoje dvije vrste imunoloških sustava : urođeni imunološki sustav i adaptivni imunološki sustav.

Urođeni imunološki sustav ima ulogu generičkog reagiranja na svaku infekciju stranim tijelom. Ne predstavlja zaštitni sustav domaćina. Uloga mu je brzo reagirati na pobudu i aktivirati adaptivni imunološki sustav.

Adaptivni imunološki sustav se sastoji od posebnih stanica koje omogućuju prepoznavanje i uklanjanje stranih organizama iz tijela. Glavne stanice od kojih se sastoji imunološki sustav su B-stanice i T-stanice. Naziv su dobile od mjesta u kojem sazrijevaju. B-stanice sazrijevaju u koštanoj srži (eng. *bonne marrow*) dok T-stanice sazrijevaju u prsnoj žlijezdi (timus ,eng. *thymus*). Jedna i druga vrsta stanica nastaju u koštanoj srži.

Glavna uloga B-stanica je stvaranje određenog antitijela protiv stranog antigena. B-stanice imaju mogućnost prelaska u dvije vrste stanica: plazma stanice i memorijske stanice. Plazma stanice imaju mogućnost stvaranja vrlo velikog broja antitijela. Memorijske stanice predstavljaju određenu vrstu memorije imunološkog sustava. One imaju mogućnost ostati u tijelu (nepromijenjene) dugi niz godina (i preko 20 godina). Upravo to svojstvo omogućuje čovjeku (i drugim organizmima) da ne obolijeva od iste vrste bolesti dva puta.

T-stanice na površini stanice imaju posebne receptore kojima se mogu vezati za antigene. Upravo ih to razlikuje od B-stanica koje takav receptor ne posjeduju. Također postoje različite vrste T-stanica s različitim ulogama. Neke od uloga su aktiviranje B-stanica ili uništavanje antigena.

2.2. Imunološki sustavi u računarstvu

Iz odlomka 2.1 se mogu uočiti neke vrlo zanimljive karakteristike imunološkog sustava. Imunološki sustav ima mogućnost raspoznavanja, pamćenja i učenja antigena. Također pojavom antigena u tijelu domaćina reagira vrlo brzo te ukloni opasnost, a da je domaći nije niti svjestan. Te karakteristike su vrlo zanimljive jer se i unutar računarstva pojavljuju mnogi problemi koji zahtijevaju metode raspoznavanja (raspoznavanje uzoraka) i učenja.

1986. godine J. Farmer, N. Packard, A. Perelson su objavili rad [1] u kojemu predlažu mreže koje temelje svoj rad na imunološkim mrežama. Cilj rada je bio predstaviti model koji će biti pogodan za simuliranje imunoloških sustava, no usporedbom modela s klasifikatorskim sustavima uočili su slično ponašanje te zaključili da se umjetni imunološki sustavi mogu primijeniti i na rješavanje problema umjetne inteligencije. H. Bersini i S. Forrest su nastavili rad i može se reći stvorili od umjetnih imunoloških sustava pravu računalsku granu. Danas se umjetni imunološki sustavi dijele na četiri područja : negativna selekcija, klonska selekcija, umjetne imunološke mreže i teorija opasnosti.

[1] govorit će se o imunološkom sustavu ljudi (kralježnjaka)

2.2.1. Negativna selekcija

Kao što je i rečeno u odlomku 2.1 T- stanice nakon nastanka u koštanoj srži dozrijevaju u prsnoj žlijezdi. Kako sama T-stanica sadrži receptor kojim se veže za strano tijelo, prilikom nastanka receptora postoji mogućnost da nastane receptor koji kao strano tijelo može prepoznati dio tijela domaćina. Rezultat toga bi bio da imunološki sustav domaćina ubije domaćina. To se ipak ne događa jer prilikom sazrijevanja T-stanica u prsnoj žlijezdi prolazi proces negativne selekcije. Negativna selekcija služi da se iz skupa T-stanica uklone one koje imaju mogućnost prepoznavanja tijela domaćina. Nakon negativne selekcije ostaju "zrele" T-stanice koje imaju mogućnost prepoznavanja samo stranog antitijela.
1994. godine S. Forrest je predložila algoritam koji radi na principu negativne selekcije. Glavna primjena algoritma je u problemima raspoznavanja uzoraka.

Slika 2.1 Umjetni imunološki sustavi

2.2.2. Umjetne imunološke mreže

Prema teoriji imunološke mreže (Niels K. Jerne 1974. godine) imunološki sustav se sastoji od niza stanica i molekula(antitijela). Antitijela ne prepoznaju samo antigene (dio tijela stranog organizma), nego već mogu prepoznati i neka druga antitijela u tijelu. Na taj način se stvara određena mreža antitijela. Kada nema pobude antigenom, mreža se nalazi u ravnotežnom stanju u kojem se broj pojedinih antitijela održava konstantnim. Prilikom pojave antigena dolazi do stimulacije cijele mreže i kao rezultat je lučenje određenog broja antitijela. Iako je ova teorija naišla na veliko protivljenje kod imunologa, u računarstvu se pokazala kao vrlo uspješna metoda za rješavanje problema skupljanja podataka u grozdove (eng. *cluster*).

2.2.3. Teorija opasnosti

Mogućnost razlikovanja vlastitog od stranog tijela ne objašnjava izostanak imunološke reakcije u situacijama kada bi se trebala pojaviti (npr. hrana u organizmu). Predložen je novi pristup koji više ne koristi razlikovanja vlastitog od stranog tijela kao deskriminatornu metodu, nego koristi razlikovanje opasnog od bezopasnog. Uvodi se novi koncept, a to je signal opasnosti. Imunološki sustav tako ne reagira na objekte, nego na signale. Ti signali potiču imunološku reakciju koja eliminira opasnost iz tijela.

3. Klonska selekcija

Kada je tijelo napadnuto stranim tijelom javlja se brza reakcija koja stimulira određene B-stanice na kloniranje (umnažanje). One stanice koje su jače stimulirane kloniraju se u većem broju. Prilikom kloniranja stanice su podvrgnute jakoj somatskoj hipermutaciji koja omogućava u vrlo kratkom roku stvaranje raznolikog skupa antitijela. Jednom kada takva B-stanica sazrije može postati ili plazma stanica ili memorijska stanica. Plazma stanice proizvode veliku količinu istovrsnih antitijela, dok memorijske stanice ostaju u tijelu vrlo dugo i tako pružaju imunološkom sustavu neki oblik memorije koja pamti već viđene antigene.
Ovaj mehanizam imunološkog sustava je bio inspiracija za algoritam klonske selekcije. Algoritam klonske selekcije je namijenjen za probleme raspoznavanja uzoraka, no vrlo ga je lako prilagoditi za optimizacijske probleme. Do danas je razvijen veliki broj algoritama koji svoji rad temelje na klonskoj selekciji, a koriste se za rješavanje optimizacijskih problema.
Algoritam koristi dva svojstva klonske selekcije. Iz skupa jedinki izabire se određen broj jedinki za kloniranje. Jedinke će biti klonirane proporcionalno svojoj dobroti (što jedinka ima veću dobrotu to je u većem broju klonirana). Mutacija svake jedinke tijekom klonske selekcije je obrnuto proporcionalna njezinoj dobroti (što jedinka ima veću dobrotu to je u manjoj mjeri mutirana).

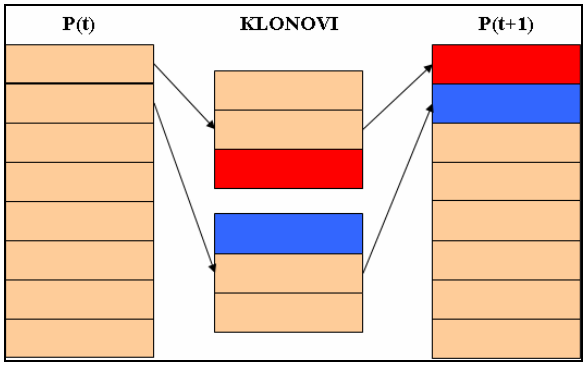
3.1. CLONALG

Leonardo N. de Castro i Fernando J. Von Zuben [2] su predložili algoritam CLONALG (eng. *CLONal selection ALGorithm*) temeljen na klonskoj selekciji. Algoritam je osmišljen za rješavanje problema raspoznavanja uzoraka, no može se primijeniti i za optimizacijske probleme.
Algoritam kod rješavanja optimizacijskih problema temelji svoj rad na jednoj populaciji (skup mogućih rješenja) koju obrađuje tijekom niza generacija. Prvo se za svaku jedinku određuje njezina dobrota. Na temelju te dobrote selektira se n jedinki za kloniranje. Prilikom kloniranja broj novonastalih klonova (N_c) se općenito računa prema jednadžbi (3.1). Ta jednadžba se prilikom rješavanja optimizacijskih problema može promijeniti u jednadžbu (3.2) bez narušavanja svojstva algoritma. Nakon kloniranja svaka jedinka prolazi proces hipermutacije (sazrijevanje jedinke) i ponovnog računanja dobrote.

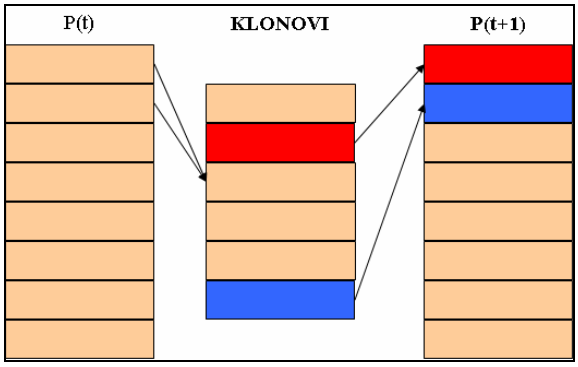
$$N_c = \sum_{i=1}^{n} round\left(\frac{\beta \cdot N}{i}\right) \tag{3.1}$$

$$N_c = \sum_{i=1}^{n} round\left(\beta \cdot N\right) \tag{3.2}$$

Selekcijom se ponovno stvara iz stare i nove generacije generacija koja ima N najboljih jedinki. Cutello i Nicosia [3] su predložili dvije vrste algoritam koje se razlikuju upravo u principu selekcije klonova.
CLONALG1 : u generaciji t svaka jedinka će biti zamijenjena svojim najboljim klonom (klon koji je nastao kloniranjem i hipermutacijom određene jedinke)
CLONALG2 : u generaciji $t + 1$ populacija će se sastojati od n najboljih klonova koji su dobiveni kloniranjem i hipermutiranjem.



Slika 3.1 CLONALG1



Slika 3.2 CLONALG2

Na kraju svake generacije u populaciju se uvodi *d* novih slučajno generiranih jedinki što omogućuje dodavanje novog korisnog materijala. Također uvođenje novih jedinki u populaciju može se izmijeniti na način da se ne uvode svake generacije nego nakon određenog broja generacija.

---

ulaz : N, n, d, β, φ
izlaz: P
P ← GenerirajRandomPopulaciju(N);
dok ¬ uvjet_zaustavljanja()
    za svaki $p_i \in P$
        Dobrota (p_i);
    kraj
    P_{select} ← Selekcija(P, n);
    za svaki $p_i \in P_{select}$
        $P_{klonovi}$ ← Kloniraj(p_i , β);
    kraj
    za svaki $p_i \in P_{klonovi}$
        HiperMutiraj(p_i , φ);
        Dobrota (p_i);
    kraj
    P ← Selekcija (P, $P_{klonovi}$, N) ;
    P_{rand} ← GenerirajRandomPopulaciju(d);
    Zamijeni (P, P_{rand});
kraj
vrati P;

Slika 3.3: Algoritam CLONALG

Kod CLONALG1 varijante algoritma ne postoji opasnost da se u jednoj generaciji pojave jedinke koje su lošije od roditelja. Kod CLONALG2 takva opasnost postoji. Da bi se to izbjeglo kod CLONALG2 algoritma može se po jedan klon od svake jedinke ostaviti nemutiran. Na taj način se uvodi vrsta elitizma jer se osigurava da u sljedeću generaciju idu jedinke koje su bolje ili jednako dobre kao jedinke iz trenutne generacije.

Tablica 3.1 Parametri algoritma

Parametar	**Opis**
P	populacija
N	veličina populacije
n	broj najboljih jedinki koje se selektiraju za kloniranje
d	broj slučajno generiranih jedinki u svakoj generaciji. Mijenjaju najlošije jedinke u populaciji.
β	faktor kloniranja
φ	faktor mutiranja

3.1.1. Hipermutacija i sazrijevanje jedinke

Hipermutacija omogućuje poboljšanje dobrote jedinki iz populacije u populaciju. Jedinke mutiraju obrnuto proporcionalno svojoj dobroti. Takav način mutacije je dobar jer omogućuje da svaka jedinka ima vlastiti faktor mutacije. Na taj način jedinke koje su vrlo daleko od optimuma koriste mutaciju za velike skokove prema optimumu dok jedinke koje su blizu koriste mutaciju za fino ugađanje.
Tu se javlja problem jer iznos optimuma nije unaprijed poznat te se ne može odrediti koja je jedinka blizu optimuma, a koja je daleko. De Castro i Timmis su predložili [4] računanje relativne dobrote pojedine jedinke u skupu klonova (jednadžba (3.4)) . Također su predložili i jednadžbu (3.3) prema kojoj se odvija hipermutacija pojedine jedinke. f je normalizirana vrijednost dobrote na interval [0,1].

$$\alpha = e^{(-\varphi * f)} \tag{3.3}$$

Normalizirana vrijednost pojedine jedine se može dobiti pomoću jednadžbe (3.4)

$$f = \frac{f_i}{f_{\max}} \tag{3.4}$$

3.2. Algoritam temeljen na B stanicama

Postoje teorije koje kažu da prilikom mutacije antitijela (receptora) B-stanice ne dolazi do slučajne mutacije pojedinih dijelova, nego se mutacija odvija na neprekinutom dijelu receptora. Inspirirani ovom idejom Kelsey i Timmis predlažu novu vrstu algoritma zasnovanog na klonskoj selekciji [5]. Algoritam je dobio ime BCA (eng. *B-Cell algorithm*).
Najvažnije svojstvo algoritma je neprekinuta mutacija koja se za razliku od CLONALG-ove hipermutacije odvija samo na neprekinutom dijelu jedinke.

```
ulaz: N, Nc, Nr, φ
izlaz: P
P ← GenerirajRandomPopulaciju( N );
dok ¬ uvjet_zaustavljanja()
        za svaki pi ∈ P
                Dobrota ( pi );
        kraj
        za svaki pi ∈ P
                Pklonovi ← Kloniraj( pi , Nc );
                Pklonovi ← GenerirajRandomPopulaciju( Nr , Nc );
                za svaki pj ∈ Pklonovi
                        HiperMutiraj( pj , φ );
                        Dobrota ( pj );
                kraj
                pi' ← OdaberiNajbolji( Pklonovi );
                ako je pi' > pi onda
                        pi ← pi';
                kraj
        kraj
kraj
vrati P;
```

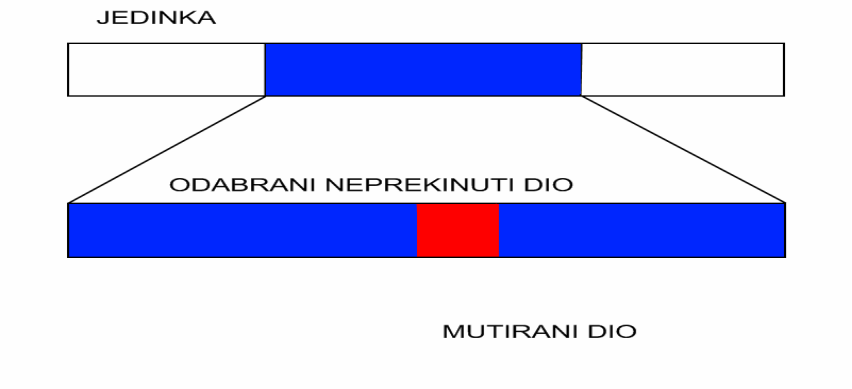
Slika 3.4: Algoritam BCA

Tablica 3.2 : Parametri BCA algoritma

Parametar	**Opis**
P	populacija
N	veličina populacije
N_c	broj klonova pojedine jedinke
N_r	broj slučajno stvoreni klonova
φ	vjerojatnost mutiranja pojedinog bita

3.2.1. Neprekinuta hipermutacija

Za razliku od hipermutacije kod CLONALG algoritma koja je uniformno razdijeljena po cijeloj jedinci (vektoru) neprekinuta hipermutacija radi na drugačijem principu. Neprekinuta hipermutacija uzima neprekinuti dio jedinke i na tom dijelu vrši mutiranje. Na jedinci se odabire mjesto i od tog mjesta se uzima neprekinuti dio (duljine *l*) jedinke.
Duljina *l* se odabire slučajno, a vjerojatnost mutacije pojedinog bita je konstantna.



Slika 3.5 Neprekinuta mutacija

3.3. Imunološki algoritam

Cutello , Nicosia i Pavone u svojim radovima ([3],[6],[7]) predstavljaju novu skupinu algoritama (temeljenu na klonskoj selekciji) koja ima neke nove operatore. Ti operatori nisu korišteni u prije navedenim algoritmima, ali su poznati u području evolucijskog računanja. Skupina je dobila ime IA (eng. *Immune Algorithms*). Najistaknutiji pripadnici skupine su opt-IA (*eng. Optimization Immune Algorithm*) i opt-IMMALG.

---

ulaz: N, β, c, τ_B
izlaz: P
P ← GenerirajRandomPopulaciju(N);
dok ¬ uvjet_zaustavljanja()
    za svaki $p_i \in P$
        Dobrota (p_i);
    kraj
    za svaki $p_i \in P$
        $P_{klonovi}$ ← Kloniraj(p_i , β);
    kraj
    $P_{mutiran}$ ← HiperMutiraj($P_{klonovi}$, c);
    Dobrota ($P_{mutiran}$);
    $P_{makromutiran}$ ← HiperMakroMutiraj($P_{klonovi}$);
    Dobrota ($P_{makromutiran}$);

    Starenje (P, $P_{mutiran}$, $P_{makromutiran}$, τ_B);
    P ← (μ + λ) - Selekcija(P, $P_{mutiran}$, $P_{makromutiran}$);

kraj
vrati P

Slika 3.6: Algoritam opt-IA

Tablica 3.3: Parametri opt-IA algoritma

Parametar	**Opis**
P	populacija
N	veličina populacije
β	faktor kloniranja
c	faktor mutiranja
τ_B	dobna granica pojedine jedinke

3.3.1. Hipermutacija i hipermakromutacija

Za pretragu prostora stanja algoritam koristi dva operatora ; hipermutacija i hipermakromutacija.

1) Hipermutacija je operator koji se izvodi nad populacijom klonova i stvara novu populaciju. Svaki klon će biti mutiran M puta gdje se M određuje prema određenom pravilu. Cutello, Nicosia, Pavone [8] su predložili tri različita načina određivanja broja mutacija na pojedinoj jedinki.
   I. Statička mutacija : Broj mutacija pojedine jedinke nije ovisan o dobroti te jedinke. Svaka jedinka prolazi kroz najviše M(x) = c mutacija.
   II. Proporcionalna mutacija : Broj mutacija pojedine jedinke je proporcionalan dobroti te jedinke. $M(x) = (E^* - f(x)) *(c*l)$
   III. Obrnuto proporcionalna mutacija : Broj mutacija pojedine jedinke je obrnuto proporcionalan dobroti te jedinke. $M(x) = (1 – E*/ f(x)) *(c*l) +(c*l)$

   Gdje je *l* duljina vektora , a E* predstavlja dobrotu najlošijeg odnosno najboljeg rješenja.
2) Hipermakromutacija je operator koji se također izvodi nad populacijom klonova i stvara novu populaciju. Broj mutacija ne ovisi niti o dobroti pojedine jedinke, niti o parametru c. Generiraju se dva slučajna broja *i* i *j*. Mora vrijediti $(i+1) \leq j \leq l$. Mutacija će se odvijati najviše $M(x) = j - i + 1$ puta na intervalu jedinke od *i* do *j* .

I u hipermutaciji i u hipermakromutaciji broj mutiranja jedinke (*M*) je zadan kao maksimalan mogući. Algoritam primjenjuje metodu prva konstruktivna mutacija (eng. *first constructive mutation*) tj. ako se nakon neke mutacije dobije jedinka koja je bolja od roditelja mutacija se prekida. Na taj način se želi spriječiti prebrza konvergencija.

3.3.2. Starenje

Proces starenja onemogućuje algoritam da se zaustavi u lokalnom optimumu. Svaka jedinka sadrži podatak o svojoj dobi (broj generacija kroz koje je prošla). Kada dosegne τ_B+1 generacija briše se iz memorije usprkos njezinoj dobroti. Prilikom kloniranja svaki klon nasljeđuje dob od roditelja. Ako klon prilikom mutacije postigne bolju dobrotu od roditelja dob mu se postavlja na 0. Na taj način se omogućuje da svaka jedinka ima mogućnost prolaska kroz čitavi životni vijek. Ako u koraku odumiranja pojedinih jedinki nova populacija ima manje jedinki od inicijalne veličine populacije tada se uvode novo generirane jedinke. Ovakav oblik starenja se naziva statičko starenje. Postoji i stohastičko

starenje gdje je smrt pojedine jedinke regulirana vjerojatnošću. Što je jedinka starija to je veća vjerojatnost pojave njezine smrti.

3.3.3. (μ,λ) i (μ + λ) selekcija

Ako se populacija sastoji od μ jedinki. Određenim postupcima iz te populacije se dobije populacija potomaka koja se sastoji od λ jedinki.
(μ,λ) – selekcija će iz populacije potomaka (koja je veličine λ) izabrati μ najboljih za novu generaciju.
(μ + λ) – selekcija će iz populacije koja je zbroj populacije roditelja i populacije potomaka izabrati μ najboljih za novu generaciju.
U postupku selekcije nije dopuštena redundancija, tj. sve μ odabrane jedinke se međusobno razlikuju.

4. Radno okruženje

4.1. Arhitektura

Radno okruženje je izgrađeno u C++ jeziku i kao takvo koristi principe objektno orijentiranog dizanja. Pretežito se koristi dinamički polimorfizam i u manjoj mjeri statički polimorfizam ostvaren preko predložaka.

Slika 4.1 Arhitektura radnog okruženja

Kao što se vidi sa slike 4.1 arhitektura je vrlo jednostavna. Glavni razred u okruženju je Sistem[2] (eng. *System*). Sistem se sastoji od šest komponenata: Algoritam(eng. *Algorithm*) , Populacija(eng. *Population*), Uvjet zaustavljanja(eng. *Stop Condition*), Statistika(eng. *Statistic*), Evaluator(*eng. Evaluator*) i Stvaratelj jedinke(eng. *Individual Creator*). Svaka komponenta je ostvarena preko apstraktne klase (sučelja) tako da je nadogradnja izuzetno jednostavna.

Rad samog Sistema se može opisati sljedećim pseudokodom:

```
dok  istina
        izvrsi generaciju (algoritam)
        izracunaj statistiku (statistika)
        provjeri uvjet zaustavljanja (uvjet zaustavljanja)
        ako je  zadovoljen uvjet zaustavljanja
                prekini izvođenje
kraj
```

Slika 4.2 Pseudokod rada Sistema

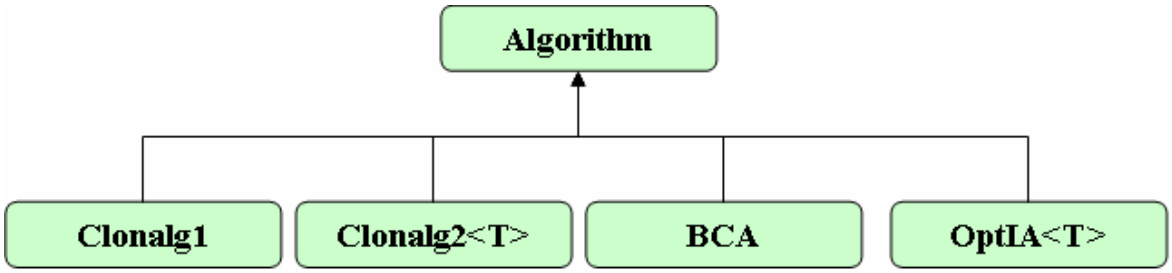
[2] napomena: u programskom rješenju se koriste engleski nazivi

Prilikom rada sa okruženjem korisnik mora programirati samo dva razreda. Razred koji će znati izračunati dobrotu jedinke (funkcija cilja) i razred koji će znati koja je jedinka od dvije ponuđene bolja.

4.2. Komponente

4.2.1. Algoritam

Algoritam je komponenta koja predstavlja jedan od algoritama iz porodice algoritama zasnovanih na klonskoj selekciji.

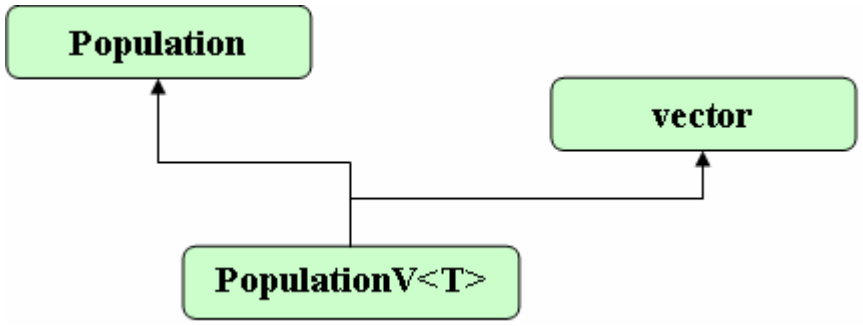


Slika 4.3 Stablo nasljeđivanja algoritama

Svaki algoritam nasljeđuje osnovnu apstraktnu klasu (Algorithm) i s njom metodu *execute()*. Ta metoda predstavlja jednu generaciju koju algoritam izvrši nad populacijom.
Kao što se vidi sa slike 4.3 dva algoritma (Clonalg2 i OptIA) su izvedena pomoću predložaka. Parametar predloška je razred koji govori koja je jedinka bolja od dvije ponuđene.

4.2.2. Populacija

Populacija se sastoji od skupa jedinki (eng. *Individual*). Hoće li jedinke biti spremljene u niz, stablo ili neku drugu strukturu ovisi o implementaciji.



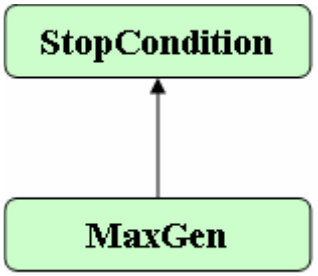
Slika 4.4 Stablo nasljeđivanja populacije

Svaka razvijena populacija nasljeđuje osnovnu apstraktnu klasu (Population) i njezino sučelje. Sučelje definira operacije (uzimanje najboljeg, uzimanje n najboljih, dodavanje jedinki itd.) nad populacijom. Ovisno o implementaciji populacije te operacije mogu biti izvedene efikasno ili manje efikasno.
Trenutno u okruženju postoji razred populacije implementiran uz pomoć vektora (PopulationV<T>). Također se sa slike 4.4 uočava da je razred implementiran pomoću predloška. Parametar predloška je razred koji govori koja je jedinka bolja od dvije ponuđene. Implementacija pomoću vektora omogućava dohvaćanje *n* najboljih jedinki (iz

populacije od N jedinki) prosječno u O(Nlog(N)) vremenu, a u najgorem slučaju u $O(N^2)$ vremenu[3]. Također dohvaćanje najbolje jedinke zahtijeva isto vrijeme što i nije zadovoljavajući rezultat, ali ga se može zanemariti jer algoritmi najčešće dohvaćaju iz populacije više od jedne najbolje jedinke.

4.2.3. Uvjet zaustavljanja

Da bi Sistem znao zaustaviti izvođenje algoritma koristi posebnu komponentu. Budući da je moguće istovremeno koristiti više uvjeta zaustavljanja Sistem uvjete zaustavljanja čuva u vektoru. Prilikom izvođenja dovoljno je da samo jedan od uvjeta bude zadovoljen i rad algoritma se prekida.

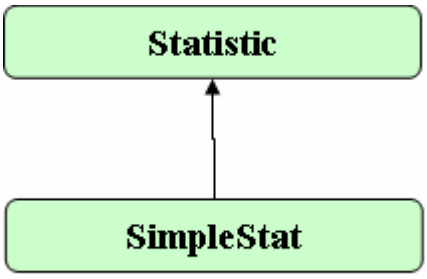


Slika 4.5 Stablo nasljeđivanja uvjeta zaustavljanja

Svaki razred koji nasljeđuje uvjet zaustavljanja mora implementirati funkciju *check()* koja vraća *true* u slučaju da je uvjet zadovoljen inače *false*. Trenutno je u okruženju implementiran samo uvjet zaustavljanja nakon određenog broja generacija.

4.2.4. Statistika

Statistika je komponenta koja se poziva u svakoj generaciji. Trenutno je implementirana samo statistika koja u svakoj generaciji računa prosjek i najbolju jedinku.

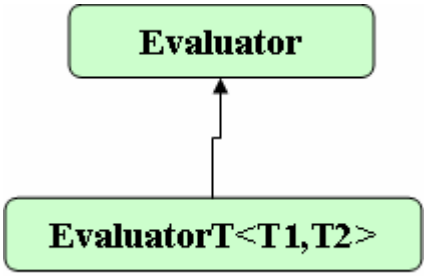


Slika 4.6 Stablo nasljeđivanja statistike

4.2.5. Evaluator

Evaluator je komponenta koja omogućuje izračun dobrote pojedine jedinke i određuje je li jedinka bolja od neke druge ili lošija.

[3] za sortiranje se koristi funkcija *sort* iz standardne biblioteke predložaka

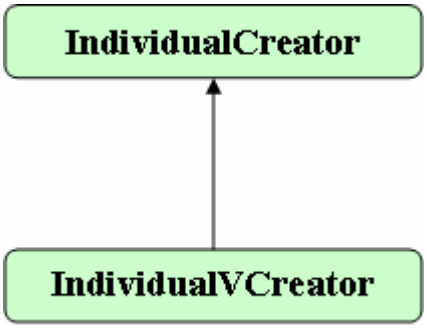


Slika 4.7 Stablo nasljeđivanja evaluatora

Trenutno je implementiran samo EvaluatorT<T1,T2> koji se temelji na predlošku. Prvi parametar predloška je razred koji zna izračunati dobrotu jedinke , a drugi razred koji zna usporediti dvije jedinke.

4.2.6. Stvaratelj jedinke

Neki algoritmi generiraju u svakoj generaciji određeni broj novih jedinki. Algoritam ne zna o kakvim se jedinkama radi i sam ih inicijalno ne zna stvoriti. To bi se moglo riješiti uz pomoć određenih parametara koji se bi predavali algoritmu, no takvo rješenje bi bilo komplicirano i neučinkovito (zbog velike količine provjera o kojim se parametrima radi). Također bi se narušio koncept radnog okruženja, jer je ideja upravo ta da algoritam ne zna s kojom vrstom jedinki radi. Drugo rješenje je korištenje obrasca tvornice.



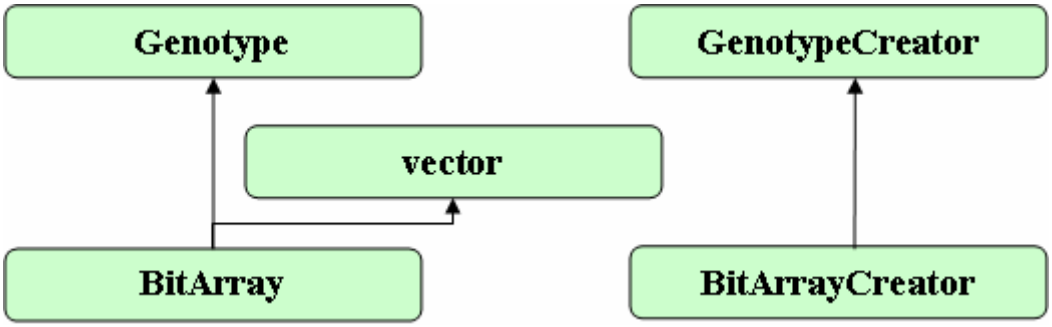
Slika 4.8 Stablo nasljeđivanja stvaratelja jedinke

Trenutno implementirani IndividualVCreator zna stvoriti jedinku koja svoje genotipe čuva u vektoru.

4.3. Ostale komponente

4.3.1. Genotip

Genotip je nositelj informacije i predstavlja dio jedinke. Postoji više vrsta genotipova: niz bitova, niz cijelih ili realnih brojeva, permutacije brojeva itd.

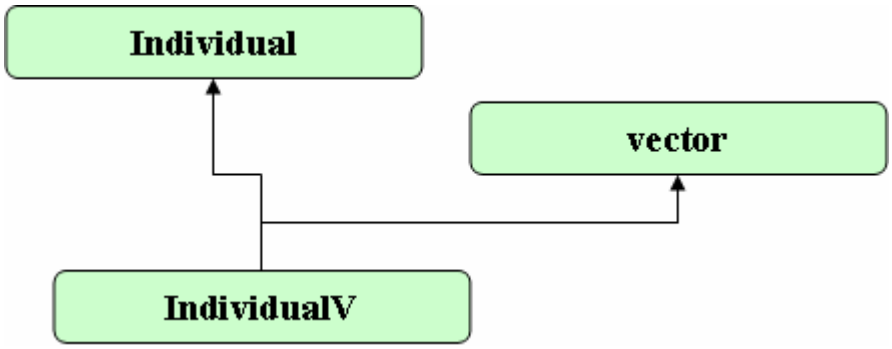


Slika 4.9 Stablo nasljeđivanja genotipa i stvaratelja genotipa

Trenutno je implementiran samo tip genotipa koji je predstavljen kao niz bitova. Također uz svaki genotip dolazi i njegova tvornica (objekt koji ga zna stvoriti). Te tvornice koristi Stvaratelj jedinke koji kao parametar prima vektor Stvaratelja genotipova.

4.3.2. Jedinka

Jedinka (Individual) je razred koji predstavlja određeno rješenje. Svaka jedinka se može sastojati od više različitih (ili istih) genotipova.

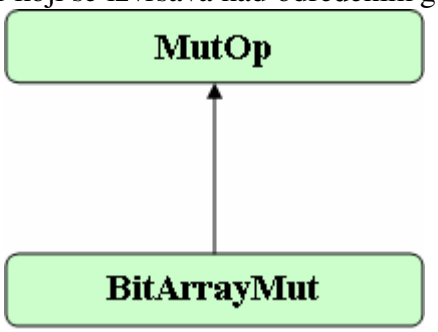


Slika 4.10 Stablo nasljeđivanja jedinke

Za sada je implementirana samo jedinka koja čuva svoje genotipove u vektoru. Ovoj jedinci odgovara njezina tvornica u obliku IndividualVCreator.

4.3.3. Operator mutacije

Operator mutacije je operator koji se izvršava nad određenim genotipom.



Slika 4.11 Stablo nasljeđivanja mutacije

4.3.4. Generator slučajnih brojeva

Generator slučajnih brojeva (Randomizer) omogućava generiranje slučajnih cijelih ili realnih brojeva u nekom intervalu ili generiranje *true* ili *false* vrijednosti s određenom

vjerojatnošću generiranja *true* vrijednosti. Razred je implementiran kao Jedinstveni objekt (obrazac Singleton) i kao takav je dostupan u svim razredima okruženja.

4.3.5. Stvaratelj populacije

Stvaratelj populacije[4] (PopulationCreator) je razred koji omogućava stvaranje vektora jedinki. Prilikom stvaranja vektora jedinki kao parametar mu se predaje Stvaratelj jedinke i željeni broj novih jedinki. Kao i Generator random brojeva implementiran je kao Jedinstveni objekt i kao takav je dostupan svim razredima u okruženju.

4.4. Usporedba radnog okruženja

Ovdje će biti uspoređeno radno okruženje s nekim poznatim radnim okruženjima iz domene evolucijskog računanja (Open Beagle , EO …).
Gagné i Parizeau su u svom radu [9] predložili šest kriterija koje bi trebalo zadovoljavati dobro razvijeno radno okruženje.
Generičko predstavljanje jedinke: U radno okruženje bi se moralo moći dodati razne tipove jedinki bez obzira pomoću kojih struktura podataka su te jedinke realizirane. Tako dodane jedinke ne smiju utjecati na prijašnji kod (načelo nadogradnje bez promjene) i prijašnji kod mora znati raditi s novim jedinkama.
Ovaj kriterij je zadovoljen u radnom okruženju jer svaka jedinka ovisi o apstraktnom razredu (sučelju) kojeg nasljeđuje. Budući da se s jedinkama upravlja preko apstraktnog sučelja i algoritmi ne znaju o kojem se tipu jedinke radi nadogradnja je vrlo jednostavna. Naravno, pritom je potrebno paziti na operator mutacije jer se ne može svaka mutacija upotrijebiti na svaku jedinku.
Generička dobrota: Dobrota jedinke mora biti neovisna od same jedinke i od bilo kakvih operatora. Također mora se moći specificirati je su li veće ili manje vrijednosti dobrote bolje. Korisnik bi trebao moći dodavati nove tipove dobrote, a da pritom ne utječe na ostatak okruženja.
Ovaj kriterij je djelomično zadovoljen u okruženju, jer se omogućuje korisniku da definira što se smatra boljom jedinkom, no sama dobrota je ugrađena u svaku jedinku i ima fiksni tip (*double*). Također nije moguće definirati proizvoljne tipove dobrote, nego je korisnik ograničen na korištenje tipa *double* što može predstavljati nedostatak.
Generički operatori: Radno okruženje bi trebalo omogućavati implementaciju i korištenje bilo kojeg tipa operatora. Također se zahtijeva da su implementirani operatori što nezavisniji od reprezentacije pojedine jedinke, što je ponekad nemoguće (npr. operator mutacije) ,ali je i nekad nužno (nitko ne želi imati deset implementacija turnirske selekcije za deset različiti tipova jedinki).
Budući da radno okruženje ne nudi veliku fleksibilnost korisniku što se tiče samih operatora može se smatrati da je ovaj kriterij samo djelomično zadovoljen. Kao što se može uočiti iz prethodna dva poglavlja radno okruženje od operatora nudi samo mutaciju. Tu je korisnik slobodan i može implementirati za svaku jedinku (genotip) razne tipove mutacija i koristiti ih. Budući da klonska selekcija ne poznaje operator križanja takav niti ne postoji u radnom okruženju. Operator selekcije je "hard kodiran" u same algoritme. Algoritmi koriste operacije koje im pruža sama populacija i tako implementiraju određenu selekciju. Ovo je ograničavajući faktor jer onemogućuje korisniku da ispita algoritme s raznim operatorima selekcije (što bi bilo svakako poželjno), no cilj je bio da se na temelju različitih implementacija populacije (temeljene na vektoru, stablu ili možda nekoj drugoj

[4] iako bi sam naziv mogao dovesti u zabludu, razred ne stvara određenu populaciju , nego vektor jedinki koje mi možemo kasnije ubaciti u populaciju

strukturi) omogući onome koji implementira populaciju da razvije efikasne metode za rad nad samom populacijom. Naime operacija selekcije je najskuplja od operacija ,jer najčešće uključuje sortiranje cijele populacije, pa se i pokušava omogućiti razvoj što efikasnijih populacija. Olakotnu okolnost može predstavljati činjenica da algoritmi klonske selekcije daju dobre rezultate na malim populacijama, no takve činjenice se ne mogu uzimati u obzir kod razvoja radnog okruženja.

Generički evolucijski model: Ovaj kriterij u idealnom slučaju zahtijeva da korisnik ima mogućnost kreiranja algoritma spajanjem operatora određenim redoslijedom.

Ovaj kriterij nije podržan u radnom okruženju jer korisnik ima mogućnost definiranja željenog algoritma kojemu kao parametar predaje operatore mutacije.

Rukovanje parametrima: Radno okruženje podržava neki mehanizam koji omogućuje dinamičko mijenjanje parametara bez ponovnog prevođenja programa.

Ovaj kriterij nije zadovoljen u radnom okruženju . Svi parametri algoritma se definiraju prilikom samog stvaranja određenog algoritma.

Konfigurabilan izlaz: Korisnik mora moći sam odrediti što želi kao ispis rezultata. Također mora moći dodavati određene statistike.

Kriterij je djelomično zadovoljen u radnom okruženju jer je moguće dodavati proizvoljne statistike te unutar tih statistika ispisivati potrebne rezultate, no nije moguće dobiti formatirani ispis željenih rezultata.

Tablica 4.1 Usporedbe raznih razvojnih okruženja

Kriterij	ECJ	OpenBeagle	EO	GAlib	radno okruženje
Generičko predstavljanje jedinke	2	2	2	2	2
Generička dobrota	2	2	2	0	1
Generički operatori	2	2	2	1	1
Generički evolucijski model	2	2	2	1	0
Rukovanje parametrima	2	2	2	2	0
Konfigurabilan izlaz	2	2	1	0	1

2 – potpuno implementirano, 1 – djelomično implementirano, 0 – nije implementirano

5. Eksperimentalni rezultati

5.1. Aproksimacijski problem

Ponekad su u stvarnom životu dostupne velike količine podataka. Najčešće se ti podaci dobivaju mjerenjem. Nakon mjerenja potrebno ih je obraditi i spremiti u računalo. Tu se pojavljuje problem jer obrada i spremanje velike količine takvih podataka može biti skoro pa i nemoguća. Srećom takav tip podataka se vrlo lako može predstaviti pomoću točaka u 2D prostoru. Intuitivno se nameće pitanje "Je li moguće pronaći funkciju koja prolazi kroz te točke i kasnije obrađivati i pamtiti pronađenu funkciju umjesto svih tih točaka?". Odgovor je "da" i postoji više načina kako se može doći do te funkcije. Kako će biti pokazano kasnije jedan od načina je i pretvaranje problema pronalaženja funkcije u optimizacijski problem te primjena neke optimizacijske tehnike (klonske selekcije) za rješavanje tog problema.

5.1.1. Definicija problema

Neka je zadan skup točaka $T = \{ (x_1,y_1), (x_2,y_2), \ldots,(x_n,y_n) \}$. Potrebno je pronaći funkciju f(x) koja će prolaziti tim točkama. Takvih funkcija ima beskonačno mnogo, no dovoljno je pronaći samo jednu. Također najčešće nije potrebno naći funkciju koja točno prolazi svim zadanim točkama ,nego je dovoljno naći funkciju koja aproksimira zadane točke tj. prolazi kroz većinu ,a približava se ostalima.
Upravo udaljenost vrijednosti funkcije u pojedinoj točki od stvarne vrijednosti može predstavljati optimizacijski problem. Budući da udaljenost može biti ili pozitivna ili negativna (u ovom slučaju), može se uzeti kvadrat udaljenosti.

$$\min \left\{ \sum_{i=1}^{n} \left(f(x_i) - y_i \right)^2 \right\} \qquad \textbf{(5.1)}$$

Jednadžba (5.1) predstavlja optimizacijski problem. Dakle potrebno je pronaći takvu funkciju f(x) za koju će izraz unutar vitičastih zagrada u jednadžbi (5.1) poprimiti minimum. Naravno minimum je 0 i to je slučaj kada funkcija prolazi kroz sve zadane točke.

5.1.2. Oblik rješenja

Oblik rješenja također nije jedinstven. Neki od mogućih oblika dani jednadžbama (5.2) i (5.3).

$$f(x) = a_0 x^{b_0} + a_1 x^{b_1} + \ldots + a_n x^{b_n} \qquad \textbf{(5.2)}$$

$$f(x) = a_0 + a_1 x + \sum_{i=1}^{n} A_i \sin(B_i x + C_i) \qquad \textbf{(5.3)}$$

Jednadžba (5.2) predstavlja polinomni oblik rješenja dok jednadžba (5.3) predstavlja oblik koji sadrži sinusne funkcije.
Prema [10] jednadžba (5.3) je pogodnija za aproksimacijski problem jer je puno manje osjetljiva na promjene.

Parametri jednadžbe su a_0,a_1,A_i,B_i,C_i. Oni direktno utječu na veličinu jedinke ovisno o rasponu i preciznosti koju želimo postići za pojedinu jedinku. Parametar *n* predstavlja broj sinus članova u funkciji. Što je taj broj veći to će funkcija lakše aproksimirati sve točke, ali također će rad samog algoritma biti usporen jer *n* također utječe na veličinu same jedinke.

Tablica 5.1 Parametri funkcije

parametar	donja granica	gornja granica	preciznost
$\mathbf{a_0}$	-10	70	2
$\mathbf{a_1}$	-1	10	3
$\mathbf{A_i}$	-70	70	2
$\mathbf{B_i}$	0	2	5
$\mathbf{C_i}$	-10	10	3

Za broj sinus članova funkcije se može uzeti n=9.Zadani parametri (Tablica 5.1) dat će veličinu jedinke od 450 bitova. Ispitivanje će se provoditi na 1000 generacija. Za aproksimaciju je korišteno 20 točaka: { (1,1), (4,50), (5,51), (7,52), (9,60), (11,71), (14,76), (15,73), (16,79), (17,68), (20,55), (22,57), (25,80), (27,82), (29,78), (31,100), (34,90), (36,85), (38,80), (40,78)}

5.2. CLONALG1 i CLONALG2

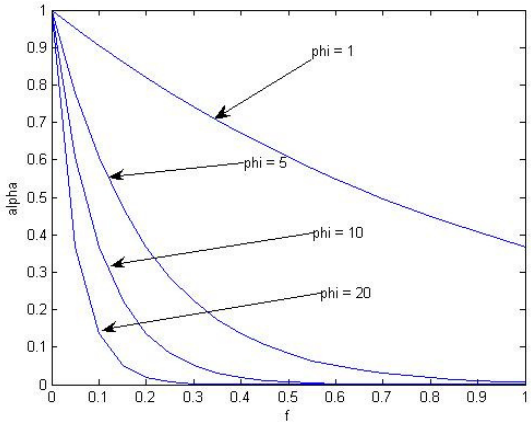
Vjerojatnost mutacije pojedine jedinke ovisi o vlastitoj dobroti prema jednadžbi (3.3). Ta jednadžba daje vjerojatnosti mutacije na intervalu [0 ,1]. Kada se bolje razmisli, taj interval nije zadovoljavajući jer najbolji klonovi dobivaju vjerojatnosti mutacije blizu 0 , a najgori blizu 1. To svakako nije dobro jer onemogućuje dobrim klonovima da se mutacijom poboljšaju , a loše klonove mutacija invertira i tako radi beskoristan posao. Zato je interval [0,1] dodatno skaliran na interval [0.01,0.1]. Velika vjerojatnost mutacije nema previše smisla jer će se algoritam pretvoriti u slučajno pretraživanje.
I jedan i drugi algoritam ovise o istim parametrima (Tablica 3.1). Parametri imaju vrlo veliku ulogu na učinkovitost algoritma i potrebno ih je precizno odrediti.
N – veličina populacije: direktno utječe na samo vrijeme izvođenja algoritma i stoga nije dobro uzimati preveliki iznos. Također valja napomenuti da i nije potrebno imati preveliku populaciju jer ionako postoji kloniranje koje će svaku dobru jedinku umnožiti nekoliko puta i tako omogućiti bolje istraživanje prostora.
n – broj najbolji jedinki za kloniranje: utječe na vrijeme izvođenja algoritma. Kod optimizacijskih problema može se koristiti n = N ([2],[3]).
d - broj novih jedinki koje se uvode u populaciju. Omogućuje unošenje novog korisnog materijala u populaciju.
β - faktor kloniranja: utječe na vrijeme izvođenja algoritma. Što više kloniramo jedinku to je veća vjerojatnost da ćemo mutacijom dobiti bolje rješenje, no također se usporava rad algoritma.
φ – faktor mutiranja : utječe na mutaciju pojedine jedinke. Što je iznos manji ravnomjernije je mutiranje jedinke, dok veći iznosi pridjeljuju lošim jedinkama veliku mutaciju, a dobrim vrlo malu(Slika 5.1).

Od svih parametara najveći utjecaj ima φ. Jasno je da povećanjem preostalih parametara, dolazi do poboljšanja, no također se usporava rad algoritma. Da bi se pokazao utjecaj parametra φ, provedeno je kratko ispitivanje na algoritmu CLONALG1.
Parametar φ je mijenjan od 2.0 do 6.0. Ostali parametri su konstantni i iznose N = 10, n = 10, d = 0, β = 0.3.

Tablica 5.2 rezultati mjerenja za različiti parametar φ

φ	najbolji	prosjek
2.0	823.079	5655.32
3.5	928.35	4192.06
4.0	737.80	3433.52
4.5	775.66	4853.54
5.0	838.85	5193.96
6.0	1568.73	5288.78

Dobiveni rezultati su daleko od zadovoljavajućih, no zadovoljavajuće rezultate nije niti bio cilj dobiti u ovom ispitivanju, nego otprilike saznati područje za parametar φ za koje se postižu najbolji rezultati.
Iz tablice 5.2 se uočava da se najbolji rezultati mogu očekivati za parametar φ na intervalu [4.0 ,4.5]. Iako je ispitivanje provedeno na algoritmu CLONALG1, za očekivati je sličan rezultat i kod algoritma CLONALG2 jer je operator mutacije kod oba algoritma isti.



Slika 5.1 Utjecaj parametra φ na mutaciju

Za parametre algoritma uzete su sljedeće vrijednosti. N = 30. Jedinka je duljine 450 bitova i prevelika populacija znatno utječe na vrijeme izvođenja algoritma. Također je za *n* uzeto *n* = N = 30. Na taj način se cijela populacija vodi prema optimumu. U takvim slučajevima (kada je *n* = N) za parametar *d* se uzima *d=0* jer nema smisla u svakoj generaciji uvoditi slučajno generirane jedinke i tako eliminirati moguća dobra rješenja. Postoji mogućnost modifikacije[5] algoritma na način da se uvede novi parametar koji govori nakon koliko generacija se uvode nove slučajno generirane jedinke. Tako nove jedinke

[5] takva modifikacija ovdje nije provedena

imaju mogućnost (uz vrlo veliku mutaciju) doći do zadovoljavajuće dobrote i "preživjeti" sljedeću slučajnu generaciju novih jedinki i izbacivanje *d* najlošijih.
Takva "politika" prema čitavoj populaciji ne predstavlja opasnost kod CLONALG1 verzije algoritma. Naime, CLONALG1 vodi svaku jedinku prema optimumu zasebno tj. nema interakcije između jedinki. Zbog toga ne postoji opasnost da cijelu populaciju preuzmu klonovi jedne (jako dobre) jedinke. To nije slučaj kod CLONALG2 verzije algoritma. Zbog elitističke selekcije (najboljih *n* klonova se vraća u populaciju) i ostavljanja od svake jedinke po jedan klon nemutiran postoji opasnost da cijela populacija bude preuzeta od jedne jedinke. To će se spriječiti na način da se promijeni mutacija tako da samo jedan klon najbolje jedinke ostavljamo nemutiran, a kod svih ostalih mutiramo sve klonove.
Za faktor φ je uzet iznos $\varphi = 4.0$. Za faktor β uzet je iznos $\beta = 0.2$ što rezultira s po 6 ($N = 30$) klonova po jedinci. Veći iznosi za faktor β uzrokuju znatnije usporenje algoritma, dok manji iznosi smanjuju učinkovitost samog algoritma. Broj generacija kroz koje prolazi algoritam je 1000.

Provedeno je po 10 mjerenja za obje varijante algoritma i rezultati su prikazani u tablici 5.3. Za svaku varijantu algoritma na kraju izvođenja zabilježena je vrijednost najbolje jedinke i prosječna vrijednost.

Tablica 5.3 Rezultati mjerenja za CLONALG1 i CLONALG2 algoritam

mjerenje	**CLONALG1**		**CLONALG2**	
	najbolji	**prosjek**	**najbolji**	**prosjek**
1.	298.757	3372.6	570.419	1434.54
2.	1129.76	4332.17	1466.13	3030.13
3.	792.087	4299.26	501.801	661.461
4.	459.555	4775.61	180.91	1000.06
5.	193.91	3400.05	816.589	1400.33
6.	265.004	4393.99	404.147	892.474
7.	646.13	4081.07	383.059	1229.59
8.	403.819	4859.78	189.897	733.058
9.	538.513	3849.85	442.782	782.667
10.	720.351	4070.32	204.515	422.973
najbolji	**193.91**	**3372.6**	**180.91**	**422.973**
prosjek	**550.78**	**4143.47**	**516.02**	**1158.73**

Iz tablice 5.3 se mogu uočiti zanimljivi (ali i očekivani) rezultati.
Može se uočiti da su i najbolje postignute vrijednosti kao i prosjek najboljih vrijednosti približno jednaki. Razlog tome leži u zajedničkom i potpuno istom operatoru mutacije. Kod prosječne vrijednosti cijele populacije su rezultati znatno drugačiji. Iznos prosjeka populacije kod CLONALG1 varijante algoritma je puno veći (preko 7 puta) od najbolje jedinke dok je ta razlika kod CLONALG2 varijante puno manja (oko 2 puta).
Razlog tome leži u konstantnoj raspodjeli mutacije. Određena jedinka može biti relativno dobra (npr. u slučaju aproksimacije 4000), no ako je ta jedinka najlošija u populaciji njezini klonovi će biti najjače mutirani što je vrlo loše jer više tako snažna mutacija nije potrebna. Kod CLONALG2 varijante ne javljaju se tako velike razlike između najbolje vrijednosti i prosjeka cijele populacije jer se tijekom selekcije uvijek uzimaju najbolji klonovi. Također predstavljena izmjena mutacije i selekcije sprečava potpuno preuzimanje populacije od strane jedne jedinke.

5.3. Algoritam zasnovan na B - stanicama

Iako Kelsey i Timmis u svom radu ([5]) za duljinu na kojoj se odvija mutacija koriste slučajno generirani broj tj. duljina ne ovisi o dobroti pojedine jedinke u ovom radu se koristi drugačije generiranje duljine. Ovdje je duljina obrnuto proporcionalna dobroti pojedine jedinke tj. što je jedinka bolja to je kraće područje na kojemu se odvija mutacija.
Tablica 3.2 prikazuje parametre algoritma. Algoritam također kao i CLONALG algoritam ovisi o veličini populacije i broju klonova. Što je veća populacija i što se veći broj klonova generira, algoritam ima veću vjerojatnost pronaći bolje rješenje. Zanimljivo je napomenuti da Kelsey i Timmis ([5]) navode da je za veličinu populacije dovoljno uzeti N u intervalu [3,5]. Očekivano najveći utjecaj na rezultat algoritma ima parametar φ. Taj parametar definira vjerojatnost mutacije pojedinog bita unutar dijela jedinke koji je odabran za mutiranje. Da bi se dobio barem približan dojam o utjecaju parametra φ na rezultat provedeno ispitivanje. Za veličinu populacije uzeto je N = 5. Za broj klonova po jedinci je uzeto $N_c = 5$ i slučajno generiranih klonova $N_r = 2$. Parametar φ je mijenjan od 0.1 do 0.9.

Tablica 5.4 rezultati mjerenja za različiti parametar φ

φ	najbolji	prosjek
0.1	1893.76	3317.58
0.2	979.7	1986.58
0.3	596.68	2472.09
0.4	586.68	3095.34
0.5	522.75	2188.11
0.6	146.121	3555.54
0.7	335.59	2456.50
0.8	284.294	2707.37
0.9	328.93	4963.60

Uočava se da algoritam za sve vrijednosti parametra φ postiže vrlo dobre rezultate. Prema rezultatima mjerenja najbolji rezultati se očekuju za vrijednost $\varphi = 0.6$.

Kao i kod CLONALG algoritma provedeno je deset mjerenja s konstantnim parametrima. Za veličinu populacije je uzeto N = 5 jedinki , broj klonova po jedinci $N_c = 5$, broj slučajno generiranih klonova $N_r = 2$ i $\varphi = 0.6$.

Tablica 5.5 Rezultati mjerenja za BCA algoritam

mjerenje	**BCA**	
	najbolji	**prosjek**
1.	336.823	3591.49
2.	134.879	2363.09
3.	61.3569	2276.87
4.	721.981	5239.1
5.	562.977	3423.35
6.	292.896	3923.75
7.	243.607	4020.69
8.	176.521	1489.72
9.	454.394	1835.25
10.	310.471	2854.38
najbolji	**61.3569**	**1489.72**
prosjek	**328.59**	**3101.77**

Kao što se vidi iz tablice 5.5 rezultati su nešto bolji od CLONALG algoritma. Najbolje rješenje je puno bolje od najboljeg rješenja kod CLONALG algoritma što pokazuje da je BCA algoritam sposoban pronaći bolje rješenje od CLONALG algoritma. Nažalost sam algoritam također ima problema s mutacijom. Nakon određenog broja generacija kada su već sve jedinke dobre , algoritam na onim najlošijima i dalje provodi veliku mutaciju. To se može uočiti iz rezultata jer je razlika između najbolje jedinke i prosjeka vrlo velika (oko 10 puta).

Budući da je ovaj algoritam pokazao vrlo dobre rezultate, postavlja se pitanje može li pokazati još i bolje. Naravno potrebno je promijeniti neke parametre. Povećan je parametar $N_c = 5$ na $N_c = 10$ i broj generacija sa 1000 na 2000. Promatraju se isti podaci (najbolji i prosjek) u 1000. i 2000. generaciji.

Tablica 5.6 Rezultati mjerenja za BCA algoritam (broj generacija = 2000)

mjerenje	BCA			
	1000. generacija		2000. generacija	
	najbolji	prosjek	najbolji	prosjek
1.	256.148	2132.98	209.341	1318.31
2.	277.381	2258.68	234.881	2149.89
3.	381.391	2134.99	253.897	1793.19
4.	74.2663	1877.54	72.1471	1515.88
5.	118.109	1996.89	98.8769	1803.65
6.	24.9558	1745.42	14.4935	1460.47
7.	291.483	1989.33	227.456	1671.7
8.	333.107	1684.11	273.96	1444.47
9.	52.9915	1269.03	46.3502	1109.23
10.	176.041	2080.03	138.608	1879.5
najbolji	**24.9558**	**1269.03**	**14.4935**	**1109.23**
prosjek	**198.59**	**1916.9**	**157.00**	**1614.629**

Iz dobivenih rezultat se uočava značajno poboljšanje rezultata algoritma. Ako se promotre iznosi dobrote najbolje jedinke u 1000. i 2000. generaciji ne uočava se prevelika razlika. Može se zaključiti da povećanje broja generacija na 2000 nije značajno doprinijelo poboljšanju rezultata. Ako se promotre rezultati iz tablice 5.5 (N_c = 5) i usporede s dobivenima u tablici 5.6 (N_c=10) može se uočiti značajnije poboljšanje rezultata. Zaključuje se da je povećanje parametra N_c značajno doprinijelo rezultatima. To je i razumljivo jer povećanjem broja klonova se omogućuje da pojedina jedinka puno bolje istraži prostor u kojemu se nalazi.

5.4. OptIA

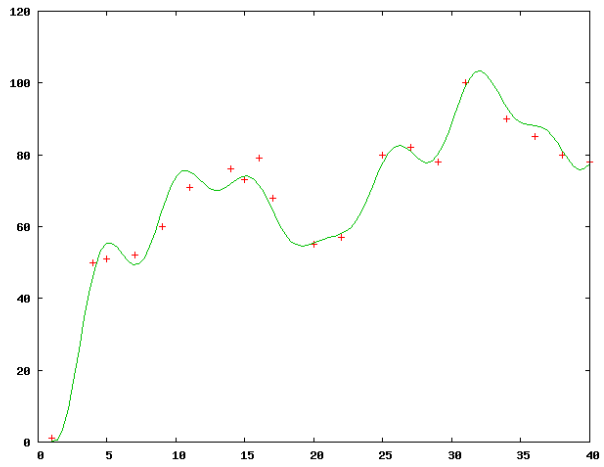
OptIA algoritam je, od dosada predstavljenih algoritama, najosjetljiviji na parametre. Budući da je duljina jedinke direktno utječe na broj mutacija koje se mogu izvršiti nad jedinkom, potrebno je taj broj regulirati parametrom *c*. Naime, jedinka je duljine 450 bitova i preveliki parametar *c* bi uzrokovao vrlo veliki broj mutacija. Takav vrlo veliki broj mutacija može značajno usporiti rad algoritma te ga tako pretvoriti u neupotrebivog. Da se to ne bi dogodilo parametar *c* je postavljen na $c = 0.01$.
Operator hipermakromutacije ne ovisi o parametru *c* i kod njega nije moguće regulirati broj mutacija parametrom pa je ograničen broj mutacija na 20.
Provedeno je ispitivanje uz sljedeće parametre . $N = 5$, $c = 0.01$, $\beta = 2$ (2 klona po jedinci) i $\tau_B = 20$.

Tablica 5.7 Rezultati ispitivanja algoritma OptIA

mjerenje	**OptIA**	
	najbolji	**prosjek**
1.	1020	1020.4
2.	2372.18	3372.27
3.	886.447	888.03
4.	200.07	202.258
5.	2784.68	2784.94
6.	107.045	107.067
7.	843.219	843.27
8.	1911.313	1912.26
9.	477.394	477.609
10.	1065.94	1066.64
najbolji	**107.045**	**107.067**
prosjek	**1157.82**	**1267.47**

Dobiveni rezultati su najlošiji od svih. Rezultati pokazuju da algoritam može pronaći vrlo dobro rješenje , ali također i da je algoritam jako ovisan o inicijalnoj populaciji. To se može uočiti iz velike razlike između najboljeg rješenja i prosjeka najboljih rješenja.
Prema selekciji algoritam je sličan CLONALG2 algoritmu. Također u algoritmu vjerojatnost mutacije pojedinog bita ne ovisi o dobroti jedinke pa se ne pojavljuje problem kao u prijašnjim algoritmima kada najlošije jedinke nisu mogle napredovat jer im je faktor mutacije bio prevelik.

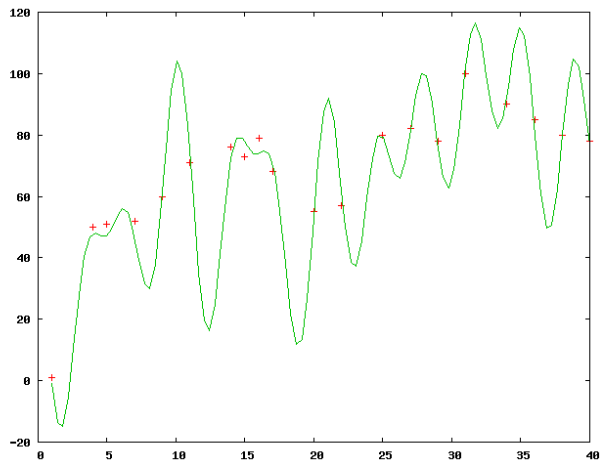
5.5. Grafički prikaz najboljih rješenja



f(x)= 20.3357 + 3.29714*x + 1.70482* sin(0.319101*x + 6.87551) + 26.2559* sin(0.964977*x + 4.68795) - 4.37099* sin(0.688304*x - 5.62609) - 8.76335* sin(0.312242*x -4.62386) + 5.26827* sin(1.13994*x -3.1254) + 26.2473* sin(0.95409*x -4.37483) + 8.75481* sin(0.303407*x - 8.43562) - 48.098* sin(0.0263215*x - 6.245) - 17.4284* sin(0.0896762*x - 8.74691)

dobrota = 193.91

Slika 5.2 Algoritam CLONALG1 – najbolje rješenje

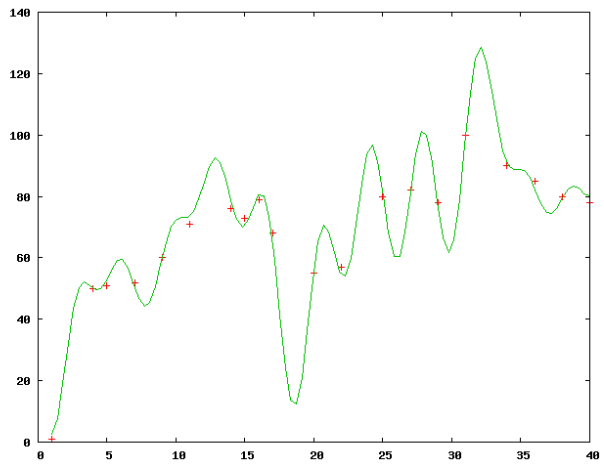


f(x) = 59.9792 - 0.995971*x - 13.5915* sin(0.281922*x + 1.98157) -42.2615* sin(1.24559*x + 5.25315) + 3.54209* sin(0.911327*x -5.7036) -9.84862* sin(1.77932*x + 4.58785) -5.77245* sin(0.750438*x -7.26676) -31.7335* sin(1.25365*x - 4.13556) + 12.1046* sin(1.75416*x - 9.53185) -63.4456* sin(0.0468752*x - 3.65276) + 18.8641* sin(1.10799*x - 9.83947)

dobrota = 180.91

Slika 5.3 Algoritam CLONALG2 – najbolje rješenje

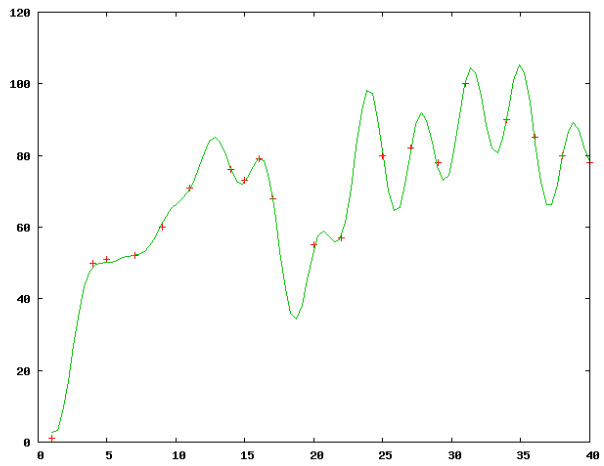
Sa slike 5.2 i slike 5.3 se uočavaju vrlo dobri rezultati i za CLONALG1 i CLONALG2. Niti za jednu točku se ne pojavljuje veliko odstupanje. Algoritam CLONALG2 je dao nešto kvalitetnije rješenje što se vidi i iz same slike 5.3. Ako se bolje promotri može se uočiti da je posljednjih 10 točaka točno aproksimirano.



f(x) = 54.1485 + 1.8247*x + 7.65244* sin(1.12496*x -3.81878) + 16.5397* sin(0.334184*x + 4.16059) - 11.7543* sin(0.680354*x - 4.84481) + 2.17482* sin(1.37296*x - 4.25459) + 10.4383* sin(1.50815*x - 3.59539) + 19.6673* sin(0.00772098*x + 4.60128) - 5.58445* sin(0.550791*x + 4.09711) - 12.643* sin(1.72508*x - 5.60717) - 8.49844* sin(0.113404*x - 8.25739)

dobrota = 61.3569

Slika 5.4 Algoritam BCA – najbolje rješenje (N_c = 5)

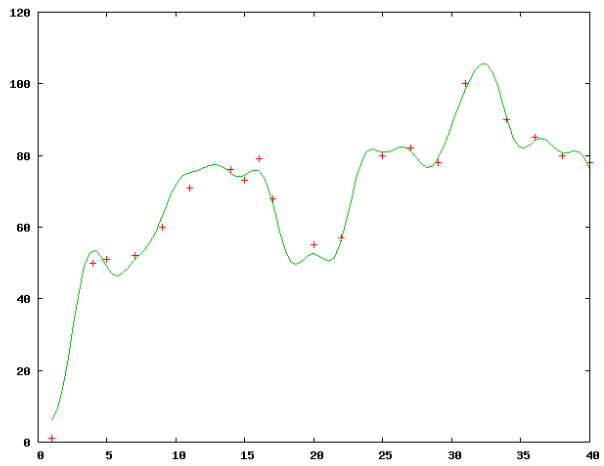


f(x) = 38.0039 + 1.06061*x - 4.90081* sin(1.57166*x + 4.31501) - 9.59226* sin(0.124962*x - 4.47859) + 7.30208* sin(0.0558169*x - 5.92029) - 12.8823* sin(0.249986*x - 3.9848) + 14.2153* sin(1.676*x - 5.33799) - 4.7299* sin(0.997257*x - 5.62487) - 8.74626* sin(0.410082*x - 6.2505) - 17.5053* sin(1.71654*x - 5.78112) - 10.9339* sin(0.671862*x + 1.40538)

dobrota = 14.4935

Slika 5.5 Algoritam BCA – najbolje rješenje (N_c = 10)

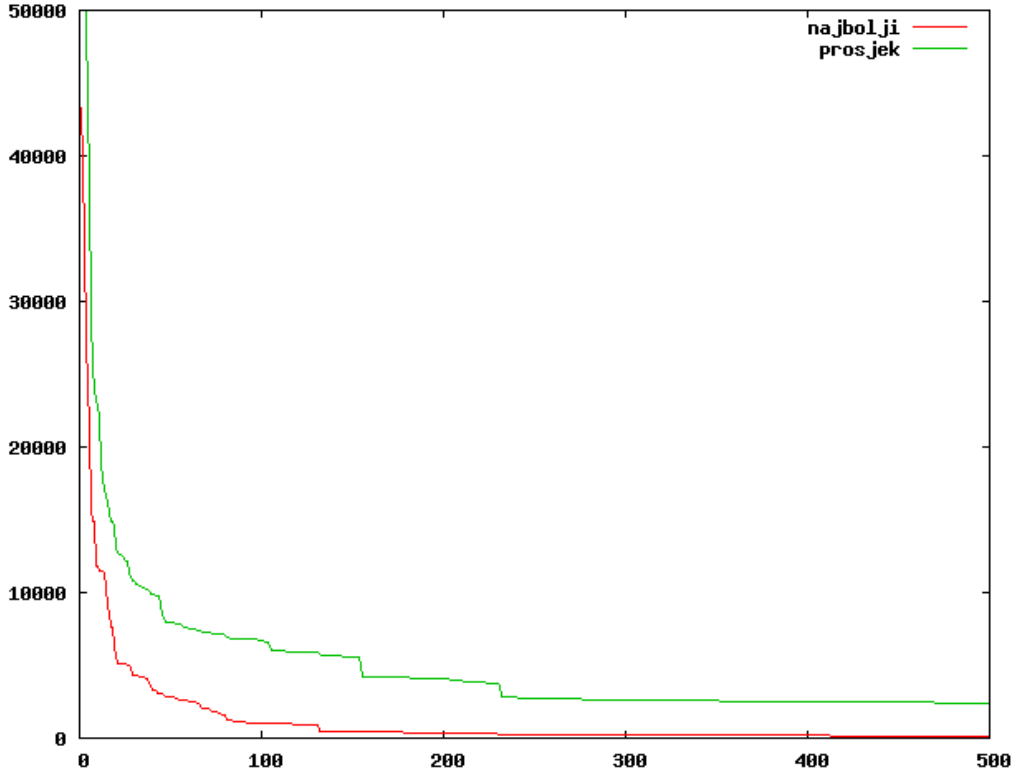
Iako su rezultati bolji za otprilike 120 do 150 sa slike 5.4 i slike 5.5 se jasno uočava da je to vrlo veliki napredak. Može se reći da je BCA algoritam uspio aproksimirati gotovo sve točke (19/20) što je odličan rezultat.



f(x) = 40.6312 + 1.40573*x + 4.36245* sin(0.187562*x + 5.54186) + 7.58408* sin(0.633311*x + 5.11216) + 1.56809* sin(0.828685*x - 4.2967) + 9.84862* sin(0.937519*x + 3.84075) - 16.2321* sin(0.312509*x - 4.84481) + 2.17482* sin(1.89683*x - 4.8442) + 2.18336* sin(1.57459*x - 4.57076) + 6.01172* sin(1.02056*x + 5.31236) + 4.37099* sin(0.180215*x - 8.51192)

dobrota = 107.045

Slika 5.6 Algoritam OptIA – najbolje rješenje



Slika 5.7 BCA algoritam N_c = 10 - Konvergencija najboljeg rješenja i prosjeka prema optimumu

Konvergencija svih algoritama prema optimumu je vrlo brza i algoritam već u prvih nekoliko generacija dolazi do dobrote koja je manja od 10000. Na slici 5.7 se može jasno vidjeti i loša strana mutacije. Vrlo brzo se pojavljuje velika razlika između prosjeka cijele populacije i vrijednosti najbolje jedinke.

6. Zaključak

U ovom radu su predstavljeni algoritmi zasnovani na klonskoj selekciji. Predstavljena su četiri algoritma koja imaju zajedničke korake, no svaki od njih sadrži i određene specifičnosti kojima nastoji poboljšati svoj rezultat.
Algoritmi su ispitani na aproksimacijskom problemu, problemu koji se smatra dovoljno složenim da pokaže realnu sliku o kvaliteti samog algoritma. Dobiveni podaci prilikom ispitivanja pokazuju da se algoritmi zasnovani na klonskoj selekciji mogu nositi i s vrlo teškim optimizacijskim problemima.
Algoritmi također predstavljaju novi oblik mutacije (mutacija ovisna o dobroti jedinke) koji do sada nije bio poznat u području evolucijskog računanja. Takav operator se pokazao kao vrlo moćan operator jer je kao jedini operator uspio dovesti populaciju do zadovoljavajućih rješenja. Ovdje valja posebno istaknuti mutaciju (mutacija na neprekinutom dijelu jedinke) koja je predstavljena u sklopu algoritma zasnovanog na B stanicama. Algoritam uspijeva na vrlo maloj populaciji (svega 5 jedinki) doći do izrazito dobrih rješenja. Nažalost sama mutacija ne zna kada je cijela populacija došla do dovoljno dobrih rješenja i uvijek one najlošije jedinke (iako su možda relativno dobre) najjače mutira. To se tijekom ispitivanja pokazalo u velikoj razlici između najbolje jedinke i prosjeka cijele populacije. Na tom području svakako postoji velika mogućnost za poboljšanje rezultata.

Predstavljeno radno okruženje je vrlo jednostavno, no dovoljno dobro da su se njime mogla izvršiti mjerenja. Budući da rad temelji na dinamičkom polimorfizmu, moguće su nadogradnje i proširenje same funkcionalnosti okruženja. Razvoj radnog okruženja nije samo zahtjevan programerski posao, nego i zahtijeva veliku ekspertnost samog programera u području kojem je radno okruženje namijenjeno.
Na predstavljeno radno okruženje veliki utjecaj ima veličina jedinke. Samo računanje dobrote jedinke je poprilično zahtjevno jer je jedinka velika 450 bitova. Budući da veličina jedinke može preći i preko 1000 bitova ovo može biti ograničavajući faktor.

Iako su relativno nova grana istraživanja unutar evolucijskog računanja, umjetni imunološki sustavi već sada pokazuju jako dobre rezultate. Područje primjene umjetnih imunoloških sustava je vrlo veliko ([11]) te se s pravom mogu očekivati novi algoritmi i još bolji rezultati.

7. Literatura

[1] J. Doyne Farmer, Norman H. Packard, Alan S. Perelson; The immune system, adaptation and machine learning; Physica 22D (1986) 187. - 204. str. North-Holland, Amsterdam

[2] Leonardo N. de Castro, Fernando J. Von Zuben ; Learning and Optimization Using the Clonal Selection Principle; IEEE Transactions on Evolutionary Computation Special Issue on Artificial Immune Systems, vol. 6, n. 3, pp. 239-251, 2002. god.

[3] Vincenzo Cutello, Giuseppe Narzisi, Giuseppe Nicosia, Mario Pavone, Clonal Selection Algorithms:A comparative Case Study Using Effective Mutation Potentials , 4th Int. Conference on Artificial Immune Systems, ICARIS 2005, August 14-17, 2005, Banff, Canada. Springer, LNCS 3627:13-28, 2005.

[4] Leonardo N. de Castro Jon I. Tinnis ; Artificial Immune Systems as a Novel Soft Computing Paradigm, Soft Computing Journal , 7. 7. 2003. god

[5] Johnny Kelsey , Jonathan Timmis ; Immune Inspired Somatic Contiguous Hypermutation for Function Optimisation; Genetic and Evolutionary Computation Conference - GECCO 2003, volume 2723 of Lecture Notes in Computer Science, Chicago. USA., July 2003. Springer-Verlag.

[6] Vincenzo Cutello , Giuseppe Nicosia, Mario Pavone, Jonathan Timmis An Immune Algorithm for Protein Structure Prediction on Lattice Models *IEEE Transactions on Evolutionary Computation*, 11(1):101-117, 2007

[7] Vincenzo Cutello , Giuseppe Nicosia, Mario Pavone , Real Coded Clonal Selection Algorithm for Unconstrained Global Optimization using a Hybrid Inversely Proportional Hypermutation Operator ; *The 21st Annual ACM Symposium on Applied Computing, SAC 2006*, April 23 -27, 2006, Dijon, France. ACM Press, vol. 2, pp. 950-954, 2006

[8] Vincenzo Cutello , Giuseppe Nicosia, Mario Pavone, Exploring the Capability of Immune Algorithms: A Characterization of Hypemutation Operators", Third Int. Conference on Artificial Immune Systems, ICARIS 2004, September 13-16, 2004, Catania, Italy. Springer, LNCS 3239:263-276, 2004.

[9] Christian Gagné , Marc Parizeau , Genericity in evolutionary computation software tools: principles and case-study , International Journal on Artificial Intelligence Tools , October 5. 2005.

[10] Marin Golub, Vrednovanje uporabe genetskih algoritama za aproksimaciju vremenskih nizova, magistarski rad, 5.11.1996

[11] Branko Mihaljević , Područja primjene umjetnih imunoloških sustava i teorije opasnosti, Kvalifikacijski doktorski ispit, Fakultet elektrotehnike i računarstva