SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1719

PRIMJENA EVOLUCIJSKIH ALGORITAMA ZA RJEŠAVANJE APROKSIMACIJSKOG PROBLEMA

Tonči Damjanić

Zagreb, ožujak 2008.

Sažetak

Ovaj diplomski rad opisuje rješavanje aproksimacijskog problema primjenom dva evolucijska algoritma, genetskog algoritma i jedne varijacije genetskog programiranja, programiranja genskih izraza. Ulazni podaci problema predstavljeni su kao niz točaka proizvoljnih vrijednosti, a rezultat optimiranja je njihova aproksimacijska funkcija. U analizama, najveći naglasak stavljen je na kvalitetu rada programiranja genskih izraza i osjetljivost njegovog algoritma na promjene ulaznih parametara. Također, postoji usporedba kvalitete tih rješenja s rješenjima genetskog algoritma.

Abstract

This diploma thesis describes solving of the approximation problem by two evolutionary algorithms, genetic algorithm and a variant of genetic programming, gene-expression programming. The problem's input data is represented by an array of arbitrary points, while the result of the optimization is an approximation function of those points. Most of the analyses are focused on the result quality of gene-expression programming and sensitivity analyses of its algorithm. Also, there's a comparison of result quality between this method and genetic algorithm.

Sadržaj

1. Uvod

Kada čovjek prikuplja informacije o svijetu oko sebe, on nesvjesno iz mnoštva informacija odabire sebi najzanimljivije te ih zadržava u sjećanju. Na taj si je način predočio svijet u jednostavnijem (aproksimiranom) obliku. Taj se proces neprestano i potpuno nesvjesno ponavlja kroz cijeli njegov život.

U matematici, aproksimacija funkcije je jako slična stvar. Za poznate vrijednosti neke potpuno nepoznate i složene funkcije, potrebno je pronaći neku koja daje slične vrijednosti, uz prihvatljivu pogrešku. Razlozi ovog procesa mogu biti mnogi, ali najčešći je ljudska potreba da si neki složen proces predoči i vizualizira jednostavnijim modelom. Analiza složenog modela možda i jest moguća, ali je u tom slučaju rijetko isplativa, bilo vremenski, bilo financijski. Kako bi se izbjeglo nepotrebno trošenje sredstava, analizom jednostavnijeg modela mogu se također dobiti prihvatljivi rezultati.

Najjednostavniji aproksimacijski problem jest spomenuto traženje jednodimenzijske funkcije za neki ulazni skup točaka. Upravo to je tema ovog diplomskog rada, s time da se ne koriste egzaktne metode, već evolucijski algoritmi. To su algoritmi koji simulacijom evolucije nad populacijom rješenja, uzrokuju njeno postupno napredovanje prema boljem rješenju, a u konačnici i najboljem. U prvi tren iznimno opskurna ideja, ali s nebrojeno pozitivnih primjena na raznolikim praktičnim problemima.

Ovaj diplomski rad sastavljen je od osam poglavlja. Nakon uvodnog, slijedi poglavlje o evolucijskim algoritmima. Ono započinje temom o biološkoj evoluciji, metodama napretka populacije kroz generacije te nastankom novih vrsta. Nastavlja se detaljnim opisom genetskog algoritma, metodologije za rješavanje raznih matematičkih problema pomoću najvažnijih evolucijskih mehanizama – operatora križanja i mutacije.

Treće poglavlje daje opis varijante genetskog algoritma, genetskog programiranja, metodologije za poboljšavanje računalnih programa. Računalni program je, u bilo kakvom obliku, jedinka populacije i na nju se mogu primijeniti genetski operatori. Ipak, zbog sporosti rada, napravljene su prilagodbe algoritma pa je nastala nova metodologija imena programiranje genskih izraza. Po prvi puta, jedinka ima razdvojen genotip i fenotip, što značajno olakšava i ubrzava genetske operacije.

Četvrto poglavlje ukratko definira aproksimacijski problem, njegove ulazne podatke i izlaznu funkciju. Sljedeće, peto poglavlje, daje opis implementacije dvaju programa za rješavanje aproksimacijskog problema pomoću genetskih algoritama i programiranja genskih izraza. Usto, opisane su njihove ulazne i izlazne datoteke te upute za ispravno pokretanje procesa traženja rješenja.

Šesto poglavlje stavlja naglasak na analizu rješenja dobivenih programiranjem genskih izraza, jer je to naslovna metodologija ovog rada. Opisano je jedno zadovoljavajuće

rješenje te kako se algoritam ponaša kad mu se mijenjaju neki od ulaznih parametara. Na kraju, obavljena je usporedba ovih rješenja i rješenja dobivenih genetskim algoritmom.

2. Evolucijski algoritmi

2.1. Evolucija u prirodi

2.1.1. Darwinova istraživanja

Organizmi na Zemlji razvijali su se kroz milijune godina evolucije. Kao početak evolucije najčešće se uzima trenutak prije otprilike četiri milijarde godina kada su se u praoceanu oblikovale prve jednostavne stanice (prokarioti). Iz tih malenih i jednostavnih organizama, djelovanjem evolucijskog procesa, s vremenom se razvio cijeli današnji živi svijet.

Put do današnjeg stadija nije bio nimalo lak. To je prvi ustvrdio Charles Darwin u svojoj knjizi „O porijeklu vrsta“, objavljenoj 1859. godine u Velikoj Britaniji. Darwin iznosi teoriju biološke evolucije po kojoj populacija neke vrste *evoluira* (napreduje) kroz generacije pomoću procesa prirodne selekcije. Također, Darwin je primijetio da veću šansu za preživljavanje imaju jedinke s boljim svojstvima, a da manju šansu imaju jedinke s gorim svojstvima pa one najčešće odumiru. Opisani proces jest proces prirodne selekcije. Zbog njega, populacija polagano napreduje po pitanju kvalitete njenih pripadnika (npr. sve je više jačih, bržih i otpornijih pripadnika).

Stoga, Darwin izvodi ključna svojstva evolucijskog procesa [1]:

1. Vrste reproduciraju više potomaka od trenutnog broja jedinki u populaciji.
2. Ipak, kroz vrijeme, broj jedinki u populaciji je stalan ili se jako sporo mijenja.
3. Količina dostupne hrane je ograničena, ali je većinu vremena konstantna.
4. Iz ovoga proizlazi da je potrebna (nemilosrdna) borba za opstanak.
5. Kod vrsta koje se spolno razmnožavaju, ne postoje dvije iste jedinke, tj. sve su međusobno različite.
6. Neke od ovih različitosti izravno utječu na sposobnost preživljavanja jedinke.
7. Većina različitosti je nasljedna te se prenosi s oba roditelja na djecu.
8. Slabije prilagođene jedinke će teže preživjeti i reproducirati se, dok će prilagođenije jedinke lakše preživjeti i zato se reproducirati.
9. Jedinke koje prežive će vrlo vjerojatno ostaviti svoje značajke budućim generacijama u naslijeđe.
10. Ovaj spori proces rezultira novom populacijom, bolje prilagođenom okruženju u kojem živi; akumuliranjem tih malenih promjena može doći do izdvajanja populacije u potpuno novu vrstu.

Sažimanjem gornjih tvrdnji i znanjem stečenim kroz vlastita istraživanja, Charles Darwin definira dva glavna pokretača evolucije [1]:

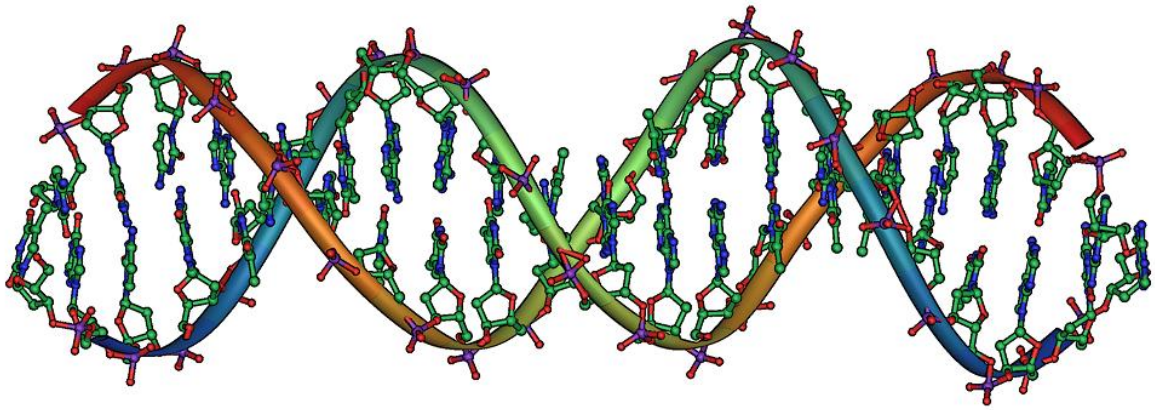
1. Prirodna selekcija – proces po kojem priroda „određuje“ koja jedinka nema dovoljno dobra svojstva za preživljavanje i kasniju reprodukciju.

2. Spolno razmnožavanje – različitost roditelja se prenosi na djecu, gdje ona dio svojstava primaju od jednog, a dio od drugog roditelja; spolnim se razmnožavanjem čuva raznolikost populacije i značajno ubrzava evolucijski proces.

2.1.2. Moderna genetika – od Mendela do molekule DNK

Ono što Darwin nije poznavao jest način na koji se svojstva roditelja prenose na djecu. Prva zapažena istraživanja ostvario je Gregor Mendel, moravski znanstvenik iz 19-og stoljeća. On je uzgajao i križao desetke tisuća stabljika graha te proučavao kako se značajke roditelja prenose na potomstvo. Otkrio je da nasljeđivanje značajki prati određena pravila te da se značajke dijele na dominantne i recesivne. Kako potomak sadrži značajke oba roditelja, ipak se na fenotipu ispoljava samo dominantna. Tek ukoliko su obje značajke recesivne, onda se ona ispoljava kao takva. Iz ovoga je Mendel zaključio da je svaka karakteristika fenotipa[1] svake jedinke nekako opisana diskretnim i neovisnim značajkama – kasnije nazvanih genima.

Daljnji napredak tehnologije omogućio je bolji pogled unutar stanice i njene jezgre, gdje su smješteni svi geni jedinke – njen genotip[2]. Geni su zapravo nizovi nukleotida međusobno povezani u dugačak lanac. Takav se lanac uparuje s drugim, komplementarnim lancem nukleotida gdje svaki nukleotid ima svoj komplementarni par. Na ovaj način se dobije dvostruka, ljestvičasta uzvojnica koja predstavlja jednu molekulu deoksiribonukleinske kiseline ili DNK, najvažnije molekule živog bića (Slika 2.1). Strukturu molekule DNK u obliku dvostruke, ljestvičaste uzvojnice opisala su 1953. godine dva engleska znanstvenika, James D. Watson i Francis Creek.



Slika 2.1. Djelić strukture molekule DNK[3]

[1] Vidljive značajke jedinke (npr. boja kose, visina, težina i sl).
[2] Genetski identitet jedinke; predstavlja cjelokupni genetski sastav jedinke iz kojeg se oblikuje fenotip.
[3] Slika preuzeta sa stranice: http://en.wikipedia.org/wiki/DNA , 12. siječnja 2008.

Razvučena molekula DNK nije praktična za korištenje zbog čega se kida na manje lance te višestrukim namatanjem nakuplja u kromosome. Svrha ovakvog nakupljanja je manje zauzimanje prostora unutar stranice, kao i lakša dioba iste kada dođe vrijeme za to.

Za vrijeme nespolne diobe stanice (mitoze) svaki se kromosom udvostruči i u stanice-kćeri se sele identične kopije kromosoma stanice-roditelja. U slučaju spolne diobe (mejoze), nema samo bezuvjetnog udvostručavanja kromosoma, već oni prolaze proces rekombinacije. Rekombinacijom se u paru kromosoma dio jednog kromosoma zamijeni s dijelom drugog kromosoma. Trenutak razmjene dijelova kromosoma naziva se križanje (engl. *crossing over*).

Uz rekombinaciju, druga moguća promjena genetske poruke naziva se mutacija. Mutacija je nepredvidljiva i relativno malo vjerojatna anomalija u procesu diobe stanice. Ipak, posljedica mutacije ne mora nužno biti loša, tj. može biti i dobra. Upravo je pozitivnim mutacijama dodatno ubrzano napredovanje populacije u evoluciji, jer bi se ta nova i dobra značajka jako brzo proširila kroz cijelu populaciju (npr. izrazito veća otpornost na neku bolest daje veću šansu za preživljavanjem te jedinke i njenih potomaka). U slučaju negativne mutacije, jedinka vrlo vjerojatno umire bez stvaranja potomstva.

2.2. Genetski algoritam kao imitacija evolucije

Veliki izazov u modernom računarstvu jest rješavanje jako teških problema neegzaktnim metodama, jer je računanje egzaktnim metodama vremenski prezahtjevno. Evolucija je jedna tipična neegzaktna metoda, jer ne postoji jedan, unaprijed određen put prema poboljšanju populacije, a i uvelike se oslanja na slučajnost. Važna karakteristika evolucije jest ta da njeno djelovanje „gura“ populaciju naprijed, pomiče prema boljem, tj. optimira je. Zbog toga, nema nikakvog razloga da se taj proces ne primijeni na razne probleme i pokuša ih se riješiti pomoću njega na računalu.

2.2.1. O evolucijskim algoritmima

Evolucijski algoritmi (engl. *evolution algorithms*) su podskup evolucijskog računanja (engl. *evolutionary computation*) i predstavljaju skup metoda za traženje optimalnog rješenja zadanog problema. Ove metode u svom radu koriste zakone i procese preuzete iz biološke evolucije. Međusobno su slične i među nekima je teško definirati granice. U evolucijske algoritme spada pet metoda optimiranja [2, 3]:

- evolucijska strategija (engl. *evolution strategy*)
- evolucijsko programiranje (engl. *evolutionary programming*)
- genetsko programiranje (engl. *genetic programming*)
- genetski algoritmi (engl. *genetic algorithm*)
- klasifikatorski sustav s mogućnošću učenja (engl. *learning classifier system*)

Evolucijska strategija (ES) je tehnika slična genetskom algoritmu, razvijena u Njemačkoj u 1960-ima. Oslanja se na generacijsko poboljšavanje populacije rješenja na koju se primjenjuju operatori, najčešće selekcije i mutacije [4].

Evolucijsko programiranje (EP) je danas metodologija širokog opsega, jer ne koristi fiksnu strukturu za reprezentaciju jedinki. Zbog toga jako podsjeća na evolucijske strategije i teško je povući granicu među njima. U prošlosti, koristili su se fiksni konačni automati i evoluirali su se njihovi numerički parametri [5].

Genetsko programiranje (GP) predstavlja metodu poboljšavanja računalnih programa za rješavanje zadanog problema. Na populaciju potencijalnih rješenja također se kroz generacije primjenjuju genetski operatori. Kako je genetsko programiranje jedna od glavnih tema ovog diplomskog rada, više detalja se nalazi u poglavlju 3.1.

Genetski algoritam (GA) je metoda optimiranja kojom se prati populaciju rješenja nekog problema i simuliranjem evolucijskog procesa nad njom traži što bolje (poželjno najbolje) rješenje tog problema.

Klasifikatorski sustav s mogućnošću učenja (LCS) je sustav za strojno učenje. Oslanja se na tehnike genetskog algoritma u optimiranju skupa pravila za klasifikaciju nepoznatog skupa uzoraka [6].

2.2.2. Genetski algoritam

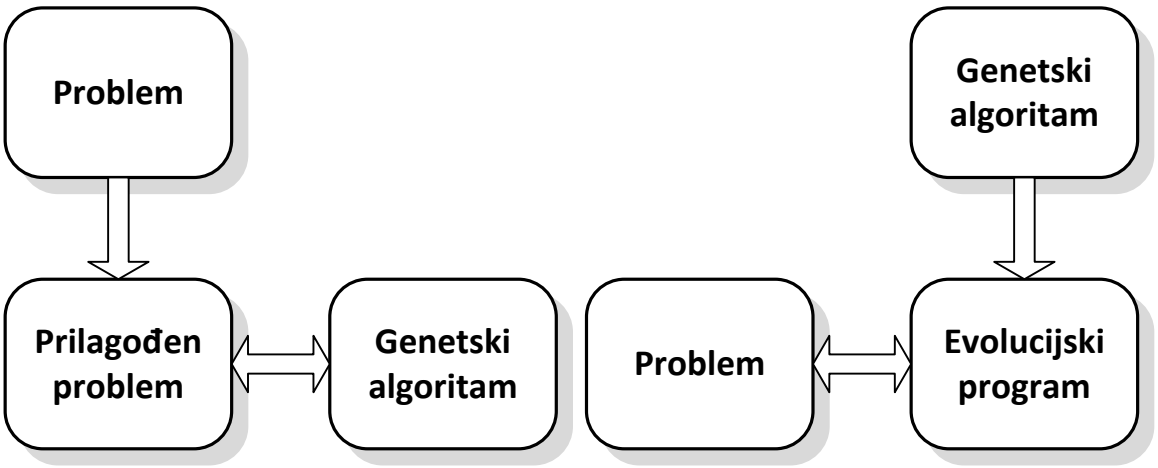
Ideju o genetskom algoritmu popularizirao je dr. John H. Holland u ranim 1970-ima. On se bavio njihovim teorijskim proučavanjem sve do sredine 1980-ih. Formalno je razradio pravila za određivanje kvalitete sljedeće generacije populacije, poznatija kao *Hollandov teorem sheme* (engl. *Holland's schema theorem*) [7]. Krajem 1980-ih, snaga računala je dovoljno narasla pa se teorijska podloga genetskih algoritama efikasno pretočila u praktičnu primjenu. Danas genetski algoritmi predstavljaju moćno oruđe za rješavanje raznolikih problema inženjerske prakse.

Naravno, genetski algoritam nije idealna preslika evolucije, već se često navodi kao „gruba aproksimacija“ evolucije. Ukoliko bi se htjela simulirati stvarna evolucija, onda bi svaka jedinka zauzimala poprilično količinu memorijskog prostora. Npr. za preslikavanje genotipa samo jedne jedinke žabe, bilo bi potrebno oko 1 GB memorijskog prostora [3]. Najčešće to nije slučaj pa genetski algoritam koristi manje memorijskog prostora.

Moguće je opisati dva, podjednako prihvaćena, pristupa rješavanju problema pomoću genetskog algoritma (Slika 2.2) [3]:

1. prilagodba genetskog algoritma specifičnostima problema
2. prilagodba problema općenitosti genetskog algoritma

U prvom slučaju, genetski algoritam rukuje veličinama svojstvenima određenom problemu. Najčešće ovo znači korištenje posebnih struktura podataka i korištenje prilagođenih genetskih operatora. Ovim prilagodbama dobiva se usko specijalizirani genetski algoritam, koji se tad obično naziva *evolucijskim programom*, izvrsnih performansi i komercijalne isplativosti. Negativna strana ovog pristupa je problemska ovisnost – primjena na drugi problem je nemoguća bez prilagodbe, a ona ne mora biti laka i jeftina. Kompromisno rješenje jest djelomična generalizacija evolucijskog programa što će mu omogućiti rješavanje čitave klase problema.



Slika 2.2. Dva pristupa rješavanju problema pomoću genetskog algoritma

Drugi pristup uvjetuje prilagodbu problema genetskom algoritmu na način da se općenite strukture podataka (npr. niz bitova) smatraju pojedinim potencijalnim rješenjem. Dakle, potrebno je definirati pravila preslikavanja iz općenitog zapisa u konkretno rješenje. Ovakav pristup ima dosta negativnih strana jer se može dogoditi da genetski algoritam za vrijeme rada generira nepostojeća rješenja. Ona se javljaju kad općeniti zapis može dati više vrijednosti od potrebnih pa se primjenom genetskih operatora dobivaju nedopuštene vrijednosti. Detekcija i uklanjanje takvih jedinki nepotrebno usporava algoritam. Također, postoje problemi čija se rješenja ne mogu modelirati na ovaj način (npr. problem rasporeda). Navedeni problemi uklanjaju se specijalizacijom algoritma: uvođenjem usko specijaliziranih zapisa jedinke i potrebnim prilagodbama genetskih operatora. Tim se potezom sprečava mogućnost nastajanja nedopuštenih vrijednosti, a genetski algoritam prelazi u evolucijski program.

John Holland je u svom radu predložio (jednostavni) genetski algoritam kao računarski proces koji imitira evolucijski proces u prirodi i primjenjuje ga na apstraktne jedinke [3]. Jednostavni genetski algoritam je često korišteni demonstracijski primjer u literaturi, jer istovremeno predstavlja i jednostavno i moćno oruđe za rješavanje problema.

2.2.3. Jednostavni genetski algoritam

Glavne značajke jednostavnog genetskog algoritma su:

- binarni prikaz jedinke
- jednostavna selekcija
- križanje s jednom točkom prekida
- jednostavna mutacija

Tablica 2.1. Pseudokod jednostavnog genetskog algoritma

```
Genetski_algoritam
{
    t = 0;

    generiraj početnu populaciju potencijalnih rješenja P(0);

    sve dok nije zadovoljen uvjet završetka evolucijskog procesa
    {
        t = t + 1;

        selektiraj P'(t) iz P(t-1);

        križaj jedinke iz P'(t) i djecu spremi u P(t);

        mutiraj jedinke iz P(t);
    }

    ispiši rješenje;
}
```

Tablica 2.1. prikazuje strukturu genetskog algoritma kroz pseudokod. Algoritam započinje *generiranjem* početne populacije potencijalnih rješenja. Nakon inicijalizacije populacije, algoritam ulazi u petlju koja će se izvršiti konačan broj puta i zatim će se ispisati najbolje rješenje u rezultirajućoj populaciji.

Unutar tijela petlje (tj. jedne *iteracije*), nad populacijom se obavljaju genetske operacije. Prvo se *selekcijom* uklanjaju najgore jedinke i na njihovo mjesto dolaze kopije boljih. Drugi korak je *križanje* parova jedinki, gdje djeca nastala križanjem zamjenjuju svoje roditelje. Posljednji korak jest eventualno *mutiranje* određenog broja jedinki u populaciji. Nakon ovog koraka, postupak se ponavlja. U terminologiji genetskog algoritma, iteracija se naziva *generacijom*.

Petlja se prekida kad se zadovolji *uvjet završetka evolucijskog procesa*. On je najčešće sastavljen od dva dijela i za zadovoljenje uvjeta dovoljno je ispunjenje jednog od njih (logička operacija *ILI*). Prvi dio je trivijalan, zajednički za sve genetske algoritme i predstavlja maksimalan broj generacija algoritma. Nakon određenog broja generacija, algoritam prestaje s radom. Drugi dio je specifičan za problem koji se želi riješiti i obično

predstavlja posebno konstruiran uvjet. Na primjer, to može biti uvjet da se 95 % jedinki nalazi unutar intervala ε ili da se trajanje projekta smanji na iznos ispod granične vrijednosti [3]. Važno je napomenuti da drugi dio često nije potrebno odrediti ili je to nemoguće pa se u tim slučajevima izostavlja.

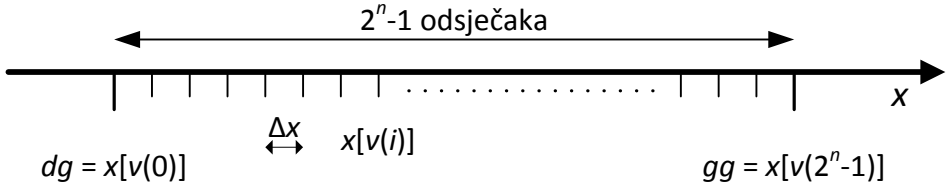
2.3. Objekti genetskog algoritma

Svaki genetski algoritam radi sa strukturama podataka koje najviše odgovaraju problemu koji se nastoji riješiti. U ovom su poglavlju opisani objekti jednostavnog genetskog algoritma, format njihovog zapisa te prednosti i mane takvog pristupa.

2.3.1. Jedinka

Računalo najbolje radi s brojevima. Ono je prije svega numeričko-aritmetički stroj i zbog toga se najbolje performanse u genetskog algoritma dobivaju kad se kao jedinke koriste brojevi. Naravno, to nije jedina reprezentacija jedinke. One se mogu reprezentirati i kompliciranijim strukturama podataka – poput niza, liste ili stabla. Jedinka populacije genetskog algoritma naziva se *kromosom*. Kod jednostavnog genetskog algoritma, kromosom je konačan niz bitova (engl. *bit array*), koji se zbog načina na koji računalo radi, tretira i kao pozitivan cijeli broj[4] (engl. *unsigned integer*). Ta dva zapisa su ekvivalentna; kad se koji koristi, ovisi o konkretnoj situaciji.

Cjelobrojna vrijednost kromosoma najčešće ne odgovara onoj koju zaista predstavlja, tj. potrebno je obaviti njeno preslikavanje u točku iz zadanog intervala na brojevnom pravcu. Kako je interval kontinuiran te ima beskonačno točaka, a skup vrijednosti kromosoma konačan i diskretan, kromosomima se mogu dodijeliti samo diskretne vrijednosti iz tog intervala (Slika 2.3). Interval je određen donjom (*dg*) i gornjom granicom (*gg*) te se naziva *područjem pretraživanja genetskog algoritma*.



Slika 2.3. Jednolika raspodjela kromosoma unutar intervala [*dg*, *gg*]

Funkcije za pretvaranje vrijednosti kromosoma u točku iz intervala su jednostavne. Kako je interval zatvoren, donja granica je predstavljena kromosomom s najmanjom vrijednošću (nula), a gornja granica kromosomom s najvećom vrijednošću ($2^n - 1$), gdje je n duljina binarnog zapisa kromosoma (broj bitova).

[4] Strogo matematički gledano nulu je potrebno posebno naglasiti obzirom da ona nije ni pozitivan ni negativan broj. Ipak, zbog jednostavnosti, ona će se podrazumijevati kao dio skupa pozitivnih cijelih brojeva.

$$b(B) = \sum_{i=0}^{n-1} B_i \cdot 2^i \tag{2.1}$$

$$x(B) = dg + \frac{b(B)}{2^n - 1} \cdot (gg - dg) \tag{2.2}$$

U navedenim funkcijama *B* predstavlja niz bitova (binarni vektor, engl. *bit vector*) i vrijedi da je $B = [B_{n-1}, B_{n-2}, \dots, B_1, B_0]$. Uz gore navedene dvije funkcije, postoji i koristi se još jedna – funkcija *v* pretvara cijeli broj *b* u binarni vektor *B*, zbog čega vrijedi: $B = v(b)$.

Odabrane točke intervala uvijek su jednako razmaknute, što znači da je interval podijeljen na $2^n - 1$ jednakih odsječaka. Duljina odsječka označava se kao Δx i iznosi:

$$\Delta x = x[v(i)] - x[v(i-1)] = \frac{gg - dg}{2^n - 1} \tag{2.3}$$

Kraći odsječak znači veću preciznost konačnog rješenja. Povećanje preciznosti rješenja postiže se produljenjem binarnog zapisa kromosoma na duljinu koja udovoljava zahtjevima. U takvim se situacijama zahtijeva preciznost rješenja *x* na najmanje *p* decimalnih mjesta, tj. x ne smije odstupati od točnog rješenja za više od 10^{-p}. Uvjet koji mora zadovoljiti *n* zadan je sljedećim izrazom:

$$(gg - dg) \cdot 10^{-p} < 2^n - 1 \tag{2.4}$$

iz kojeg proizlazi da je minimalna duljina kromosoma:

$$n \geq \frac{\log[(gg - dg) \cdot 10^p + 1]}{\log 2} \tag{2.5}$$

Dakle, bilo koji *n* koji zadovoljava gornju nejednadžbu, osigurava preciznost rješenja na barem *p* decimalnih mjesta. Veća preciznost usporava algoritam stoga je potrebno odvagnuti najbolji odnos brzina-preciznost za konkretni problem.

2.3.2. Populacija

Već je spomenuto da je veličina populacije neke vrste u prirodi konstantna ili se jako sporo mijenja. To se svojstvo prenosi i u područje genetskih algoritama. Stoga jednostavni genetski algoritam radi s populacijom uvijek iste veličine. Postoje neke implementacije s promjenjivom veličinom populacije, ali se pokazalo da to svojstvo ne poboljšava konačno rješenje, a donekle usporava izvođenje algoritma. Veličina populacije je još jedan ulazni parametar genetskog algoritma i u ovom radu označavat će se kao *VEL_POP*.

Dva su istaknuta načina stvaranja početne populacije. Najčešće se ona popuni slučajno generiranim jedinkama, što rezultira velikom genetskom raznolikošću i većom vjerojatnošću pronalaska najboljeg puta prema optimumu. U drugom slučaju, populacija

se radi kopiranjem jedne jedinke. Ovime je uklonjena genetska raznolikost i potrebno je neko vrijeme dok se ona dobije radom genetskih operatora. Ponekad se u početnu populaciju kao jedinka ubaci (među)rješenje dobiveno nekom drugom optimizacijskom tehnikom [3].

Ukupna dobrota populacije *D* i prosječna dobrota populacije $\overline{D}$ računaju se pomoću izraza:

$$D = \sum_{i=1}^{VEL\_POP} dobrota(v_i) \tag{2.6}$$

$$\overline{D} = \frac{D}{VEL\_POP} \tag{2.7}$$

2.4. Operatori genetskog algoritma

2.4.1. Evaluacija

Mjera kvalitete svake jedinke određuje se u ovoj fazi. Sâm genetski kod ništa ne govori o dobroti jedinke, već se ta vrijednost mora izračunati. Kako bi se dobrota mogla izračunati, prvo je potrebno odrediti *funkciju dobrote* (funkcija sposobnosti, funkcija cilja, engl. *fitness function*). U najjednostavnijem slučaju, funkcija dobrote je jednaka funkciji *f* koju se želi optimirati i tada vrijedi:

$$dobrota(v) = f(x) \tag{2.8}$$

gdje je *v* binarni vektor (kromosom), a on je predstavljen brojem *x* iz intervala [*dg*, *gg*]. Veća dobrota jedinke znači veću vjerojatnost preživljavanja i križanja, a time i mogućnost poboljšanja genetskog materijala. Ovakva jednostavna veza funkcije dobrote i optimirane funkcije često nije moguća, zbog ograničenja koja se postavljaju funkciji dobrote. Za optimiranu funkciju *f*(*x*), naprotiv, najbolje je da nema nikakvih ograničenja, tj. ona je potpuno proizvoljna. U procesu prilagodbe problema genetskom algoritmu, odabir funkcije dobrote pokazao se kao najveća poteškoća, jer njena pogrešna ocjena vodi populaciju u krivom smjeru. Funkcija dobrote, stoga, treba vjerno odražavati problem koji rješava [3].

2.4.2. Selekcija

Uloga selekcije je čuvanje i prenošenje dobrih svojstava u iduću populaciju, dok jedinke s lošim značajkama odumiru. Ukoliko se selekcija definira deterministički (na primjer, u svakoj generaciji ukloni *M* najgorih jedinki, gdje je *M* broj jedinki za eliminaciju), genetski algoritam jako brzo, već kroz nekoliko iteracija, konvergira u najbližem lokalnom optimumu i dalje se ne miče iz njega. Stoga, potrebno je na neki način omogućiti i najgorim jedinkama bar malenu vjerojatnost preživljavanja za vrijeme selekcije. Boljim

jedinkama je ta vjerojatnost mnogo veća, ali nikad nije jednaka jedinici. Ovakva politika predstavlja rizik gubitka dobrog genetskog materijala, ali genetska raznolikost populacije u svakom je trenutku važnija od egzistencije pojedinih jedinki.

Postoje mnogi selekcijski mehanizmi, a u ovom poglavlju obradit će se najpoznatiji od njih: jednostavna selekcija, turnirska selekcija, eliminacijska selekcija i elitizam.

Jednostavna selekcija

Genetski algoritam koji koristi *jednostavnu selekciju* (selekcija roditelja ruletom, engl. *roulette wheel parent selection*) naziva se *generacijski genetski algoritam*. Jednostavna selekcija, naime, u svakoj generaciji od jedinki stare populacije generira potpuno novu populaciju. Ovo znači da se nijedna jedinka ne prenosi izravno iz stare u novu populaciju. Postupak generiranja nove populacije je jednostavan [3]:

1. Dobrota jedinke mora biti pozitivan broj, tj. $dobrota(v) \geq 0, \forall v \in P$, gdje je *P* populacija jedinki.
2. Izračunaju se sve vrijednosti dobrote jedinki.
3. Izračuna se ukupna dobrota populacije *D* prema formuli (2.6).
4. Izračunaju se kumulativne dobrote q_k za svaki kromosom tako da vjerojatnost selekcije za svaki kromosom v_k iznosi p_k:

$$q_k = \sum_{i=1}^{k} dobrota(v_i) \ , k = 1, 2, \dots, VEL\_POP \quad \textbf{(2.9)}$$

$$p_k = \frac{dobrota(v_k)}{D} \quad \textbf{(2.10)}$$

   Ovim se dobiva da je vjerojatnost selekcije proporcionalna dobroti kromosoma:

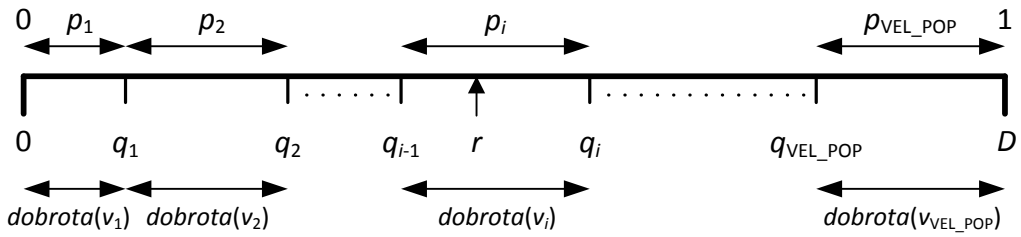
$$p_k \sim dobrota(v_k) \quad \textbf{(2.11)}$$

5. Generira se slučajan realan broj *r* u intervalu $[0, D\rangle$, potraži se *i*-ti kromosom za koji vrijedi da je $r \in [q_{i-1}, q_i\rangle$ te se prenosi u sljedeću populaciju.

Slika 2.4. grafički prikazuje mehanizam jednostavne selekcije. Slučajan broj *r* „pogodio“ je jedinku *i* zbog čega se ona prenosi u novu populaciju. Postupak se ponavlja *VEL_POP* puta. Upravo zbog sličnosti s pravim ruletom, gdje loptica na slučajan način pogađa numerirane utore, jednostavna selekcija je dobila i drugo ime.

Ovakvim načinom selekcije osigurana je veća vjerojatnost preživljavanja boljih jedinki, a opet nije u potpunosti isključeno preživljavanje najgorih. Ipak, jednostavna selekcija ima i velike nedostatke. U prvom redu, nije predviđen mehanizam provjere postoji li odabrana jedinka već u populaciji, zbog čega se može javiti mnogo duplikata (čak do 50% dupliciranih kromosoma [3]). Duplikati usporavaju napredovanje algoritma zbog smanjene genetske raznolikosti. Drugi problem može nastati ukoliko se loše definira funkcija dobrote, što može uzrokovati podjednaku vjerojatnost preživljavanja svih

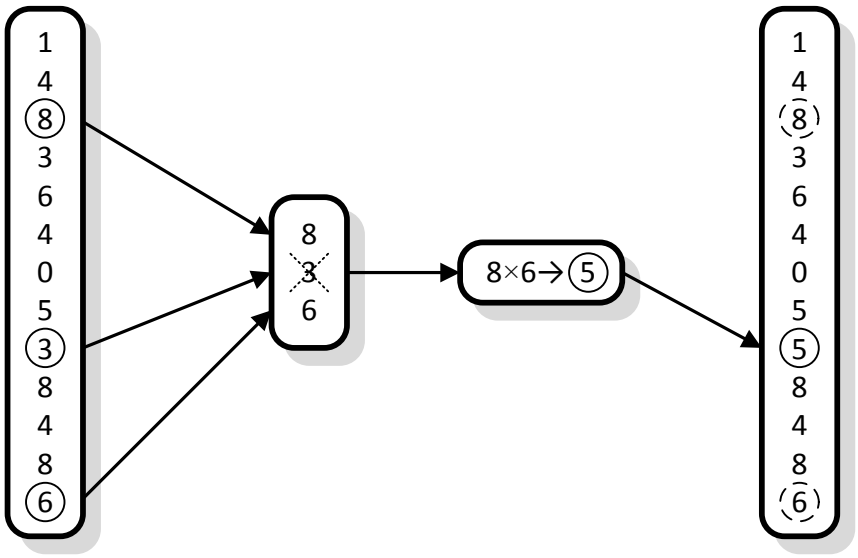
kromosoma. Ugradnjom jednostavnih dodatnih mehanizmima, oba problema se lako rješavaju.



Slika 2.4. Kumulativna dobrota q_i i vjerojatnost selekcije p_i

Turnirska selekcija

U slučajevima kada je kromosom predstavljen kompliciranijom strukturom podataka (npr. vezanom listom ili binarnim stablom) od običnog cijelog broja, obavljanje jednostavne selekcije postaje računski najzahtjevnija operacija. Obavlja se mnogo kopiranja, slijednog pregledavanja liste i međusobnog uspoređivanja elemenata, što sve zajedno jako usporava algoritam. U takvim se slučajevima mora nekako smanjiti broj kromosoma uključenih u selekciju po generaciji. Od takve ideje, nastala je *k-turnirska selekcija* (engl. *tournament selection*), gdje je *k* prirodni broj veći od jedinice i oznaka veličine turnira.



Slika 2.5. Primjer obavljanja jednog koraka eliminacijske 3-turnirske selekcije

Postoje dvije vrste turnirske selekcije: generacijska i eliminacijska. Generacijska turnirska selekcija izdvaja *k* slučajno odabranih jedinki i najbolju od njih kopira u novu populaciju. Kako bi se formirala nova populacija, potrebno je postupak ponoviti *VEL_POP* puta. Eliminacijska turnirska selekcija ne stvara novu populaciju, već mijenja postojeću. Prvo slučajnim odabirom stvara grupu od *k* jedinki i uklanja najgoru iz populacije. Križanjem preostalih jedinki stvara novu jedinku koja će popuniti nastalu prazninu u populaciji. Ovaj proces se ponavlja *M* puta, gdje je *M* mortalitet – broj jedinki za eliminaciju [8].

Slika 2.5. prikazuje izvođenje 3-turnirske selekcije s reprodukcijom. Iz početne populacije se izdvaja tri, slučajno odabrane jedinke. Uklanja se ona s najmanjom dobrotom (jedinka broj 3) i na njeno se mjesto u populaciji ubacuje nova jedinka (broj 5), ona dobivena križanjem. Na slici, znak „×“ predstavlja operator križanja.

U usporedbi s jednostavnom selekcijom, turnirska se mnogo brže izvodi i jednostavnija je za implementaciju. Relativno velika mogućnost duplikacije kromosoma i dosta kopiranja istih, problemi su generacijske turnirske selekcije, dok su isti ti problemi znatno umanjeni ukoliko se odabere eliminacijska turnirska selekcija.

Eliminacijska selekcija

Još jedno poboljšanje jednostavne selekcije predstavlja njena varijanta, zvana *eliminacijska selekcija* (engl. *steady-state selection*). Za razliku od jednostavne koja preferira *dobre* kromosome za prelazak u novu populaciju, eliminacijska selekcija odabire *loše* i uklanja ih iz populacije. Zatim se genetskim operatorima od preostalih kromosoma populacija dopunjava do svoje normalne veličine.

Kako bi se mogla provesti eliminacijska selekcija, potrebno je umjesto funkcije dobrote, definirati *funkciju kazne* tako da vrijedi:

$$p_k \sim kazna(v_k), \qquad k = 1, 2, \dots, VEL\_POP \tag{2.12}$$

$$kazna(v_k) = \max_{\mathrm{i}}\{dobrota(v_i)\} - dobrota(v_k) \tag{2.13}$$

Vjerojatnost eliminacije kromosoma proporcionalna je iznosu funkcije kazne tog kromosoma. Obzirom da je vjerojatnost eliminacije najbolje jedinke jednaka nuli, ona se neće izgubiti u selekcijskom procesu. Ovo je tipičan primjer *elitizma* po kojem se iz generacije u generaciju čuva najbolja jedinka populacije.

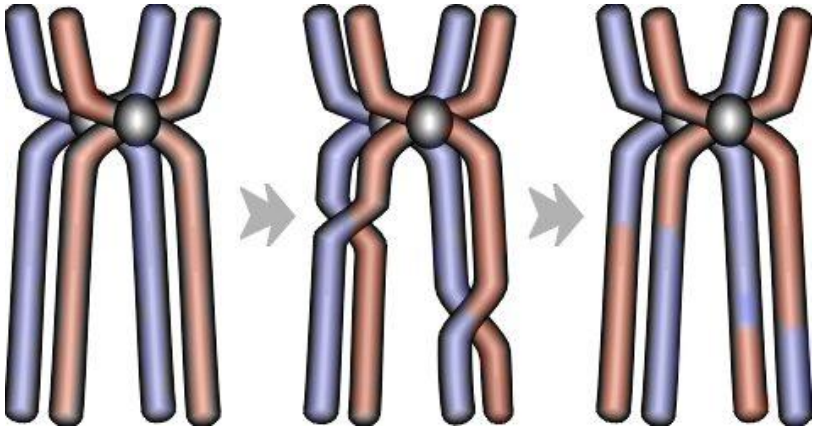
Elitizam

Kako genetski algoritam pripada razredu stohastičkih algoritama, ne postoji garancija da će najbolja jedinka svaki put preživjeti selekcijski „odstrel“. Dakle, postoji mogućnost gubitka najboljeg kromosoma i time nazadovanje populacije na nekoliko generacija. Rješenje problema je *elitizam*, a predstavlja mehanizam zaštite najbolje jedinke od eliminacije. Genetski algoritam s ugrađenim elitizmom kroz generacije asimptotski napreduje prema globalnom optimumu.

Negativna strana elitizma je usporenje algoritma zbog traženja najbolje jedinke prije početka selekcije. Povećanjem složenosti strukture kromosoma i pripadajuće funkcije dobrote, traženje najboljeg kromosoma može utrošiti značajnu količinu procesorskog vremena [3, 8].

2.4.3. Križanje

Kod živih organizama, *križanje* (engl. *crossing over*, *crossover*) se obavlja za vrijeme mejoze, kada se formiraju spolne stanice jedinke. Tada se upareni homologni kromosomi, bivalenti, prekidaju na slučajnoj poziciji i zatim razmjenjuju odsječke genetskog materijala (Slika 2.6).



Slika 2.6. Križanje kromosoma unutar stanice[5]

Nakon uspješne razmjene gena (*rekombinacije*), mejoza se nastavlja. Parovi kromosoma se razdvajaju te od početne (diploidne) stanice nastaju dvije stanice s polovičnim (haploidnim) brojem kromosoma. Ovakvo dijeljenje je potrebno kako bi potomci jedinke opet imali diploidan broj kromosoma. Naime, polovicu dobivaju od jednog, a polovicu od drugog roditelja u trenutku začeća novog organizma.

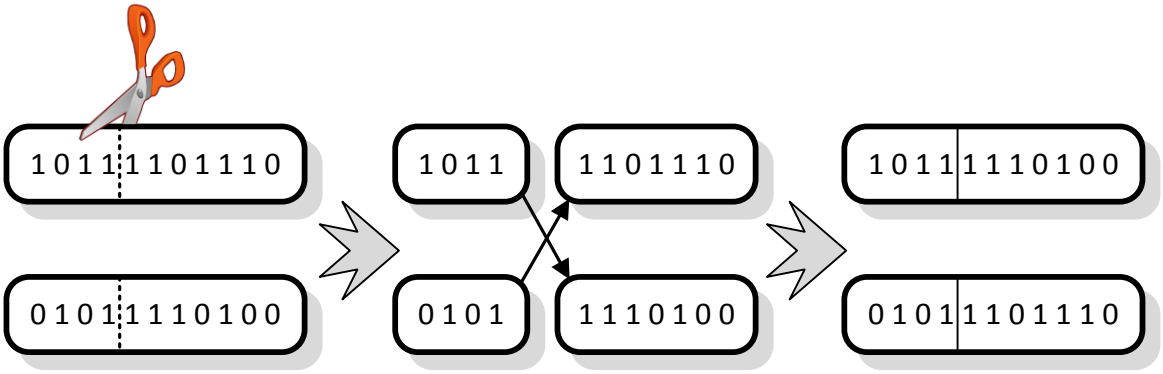
Križanje genetskog algoritma je poprilično jednostavnije od prirodnog, ali ipak zadržava osnovni smisao – nasljeđivanje gena i popravljanje kvalitete populacije. Kreće se od pretpostavke da ukoliko su roditelji *dobri* (prošli su selekciju), onda bi i djeca trebala biti *dobra*, a poželjno i bolja od roditelja [3]. Kod jednostavnog genetskog algoritma, kromosomi su nizovi binarnih znamenki i kao takvi lagani za križanje. Binarna znamenka u kromosomu predstavlja djelić genetskog materijala i zato se naziva *gen*.

Jedna od metoda križanja se naziva *križanje s* m *točaka prekida*. „Točka“ je u ovom slučaju mjesto između dva gena kromosoma, dok je $m \in \{1, 2, \dots, n-1\}$. Slika 2.7. zorno prikazuje križanje dva kromosoma gdje je točka prekida između 4. i 5. gena. Kromosomi se razrežu na toj poziciji, zatim se zamijene odrezani dijelovi te ponovo spoje u dvije kompaktne cjeline.

Druga izvedba križanja binarnih kromosoma zove se *uniformno križanje*. Kod takvog križanja svakom se genu roditelja pridružuje neka vjerojatnost njegovog nasljeđivanja na djecu. Naravno, zbroj vjerojatnosti prenošenja gena na istoj poziciji mora biti 1 (npr. vjerojatnost da će gen *x* biti naslijeđen od prvog roditelja iznosi 0,6, a od drugog 0,4). „Najpošteniji“ oblik uniformnog križanja je slučaj kad svaki gen ima vjerojatnost

[5] Slika preuzeta sa stranice: http://www.stanford.edu/group/Urchin/meiosis.htm , 29. siječnja 2008.

nasljeđivanja 0,5. U tom slučaju, operator križanja je najlakše izvesti u obliku logičke operacije nad bitovima, jer su one ugrađene u strojni jezik računala i najbrže se izvode.



Slika 2.7. Križanje kromosoma s jednom točkom prekida u genetskom algoritmu

Prvi korak jest prenošenje gena koji su jednaki, a ostale postaviti na nulu. To se radi logičkom operacijom *I*: *A I B*, gdje su *A* i *B* roditelji. Nadalje, neka je *R* slučajno generiran kromosom. Tada operacija: *R I* (*A XILI B*) daje masku slučajnih bitova na mjestima gdje se roditelji razlikuju. Konačno, logičkom operacijom *ILI* stapaju se dva izračunata međurješenja i dobiva se kromosom-dijete. Izraz:

$$dijete = A \cdot B + R \cdot (A \oplus B) \qquad \textbf{(2.14)}$$

predstavlja matematički zapisan način dobivanja kromosoma djeteta. Ova formula, naravno, vrijedi samo za kromosome u obliku binarnih brojeva. Ukoliko je gen predstavljen nekom složenijom strukturom, uniformno križanje se izvodi na drukčiji način.

Upravo operator križanja razlikuje genetski algoritam od ostalih evolucijskih algoritama, jer nijednom od njih nije sastavni dio.

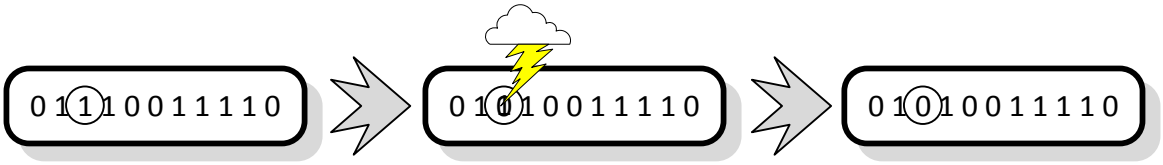
2.4.4. Mutacija

Kada se dogodi mutacija za vrijeme života stanice nekog organizma, dolazi do nepredvidljive promjene u genetskoj poruci. Najčešće se ta poruka detektira unutar same stanice pa se popravi na vrijeme, ne ostavljajući nikakav trag iza sebe. No, neke mutacije ostaju neprimijećene i često se njene manifestacije mogu vidjeti na fenotipu jedinke. Na primjer, mutacija kod leptira može uzrokovati promjenu uzorka na krilima što će mu omogućiti kvalitetniju mimikriju i veću sposobnost preživljavanja. Takvo, poboljšano svojstvo, vjerojatno će kroz nekoliko generacija prevladati u populaciji. Naprotiv, veliki broj mutacija ima negativne posljedice zbog kojih mutirane jedinke odumiru prirodnom selekcijom.

Genetski algoritam, kao i svi evolucijski algoritmi, oslanja se na mutaciju u svom radu. Ona je glavna pokretačka snaga algoritma, jer njena nepredvidljivost može jedinku baciti u nepoznato područje, područje gdje se nalazi globalni optimum. Naime, genetski algoritam

ima tendenciju zaglaviti u najbližem lokalnom optimumu (već nakon malenog broja generacija) te se kasnije teško ili nikako ne miče iz njega. Mutacija izbacuje jedinku iz tog napučenog područja. Ako se pomak pokazao pozitivnim, cijela populacija će se jako brzo preseliti na to novo, neistraženo područje. Ukoliko mutacija da loše rezultate, jedinka ubrzo odumire selekcijom.

Kao i sve kod jednostavnog genetskog algoritma, mutacija je također jednostavna. Ulazni parametar u algoritam jest vjerojatnost mutacije gena p_m. Ukoliko je ona bliska jedinici, algoritam se pretvara u algoritam slučajne pretrage prostora rješenja, a ukoliko je bliska nuli, populacija će zaglaviti u najbližem lokalnom optimumu iz kojeg će teško izaći. Jednostavna mutacija svaki bit mijenja s istom vjerojatnošću p_m [3]. Slika 2.8. prikazuje mutaciju trećeg gena kromosoma iz jedinice u nulu.



Slika 2.8. Prikaz mutiranja slučajno odabranog bita kromosoma

Uz jednostavnu, postoji još nekoliko vrsta mutacija nad binarnim kromosomima, koje će se samo ukratko spomenuti. One se ne moraju primijeniti na cijeli kromosom, već se mutirani dio određuje preko slučajno generirane maske, niza bitova dugog koliko i kromosom. Jedinice u maski određuju na koje će se bitove kromosoma primijeniti operator mutacije.

Dakle, preostale vrste mutacije su [3]:

- miješajuća mutacija – broj jedinica i nula u mutiranom dijelu ostaje isti
- potpuna miješajuća mutacija – generira slučajne bitove u mutiranom dijelu
- invertirajuća miješajuća mutacija – radi logičku operaciju *NE* nad mutiranim dijelom

Ukoliko je kromosom građen od gena složenije strukture, onda vjerojatnost mutacije gena možda nije odgovarajuća niti primjenjiva. U takvim slučajevima treba poznavati vjerojatnost mutacije kromosoma p_M. Ukoliko se ona želi odrediti u slučaju kad je poznat parametar p_m, tada vrijedi izraz:

$$p_M = 1 - (1 - p_m)^n \tag{2.15}$$

Uz ulogu izbacivanja populacije iz lokalnog optimuma, mutacija također omogućava povratak izgubljenog genetskog materijala, onog kojeg nije moguće vratiti samo križanjem jedinki [3].

3. Programiranje genskih izraza

Rješavanje određenih problema obavlja se traženjem optimalnog računalnog programa. U takvim situacijama najbolja se pokazala varijanta genetskih algoritama nazvana genetsko programiranje, iz kojeg je proizašla metoda programiranja genskih izraza.

3.1. Genetsko programiranje

Jednu od primjena genetskog programiranja ostvario je dr. John R. Koza u raznolikim optimizacijskim problemima i problemima traženja već u 1980-ima. Kako je genetsko programiranje „gladno“ računalne snage, tek se danas ono primjenjuje u mnogo većem spektru područja poput kvantnog računarstva, dizajna sklopovlja, igranja igara, sortiranja, traženja i sl. Često se ističe podatak o nekoliko patenata zasnovanih na rješenjima dobivenim radom genetskog programiranja [9].

Osnovna zamisao metode jest optimiranje računalnih programa na računalu. Računalo prati populaciju računalnih programa i kroz određeni broj generacija poboljšava tu populaciju primjenom poznatih genetskih operatora – selekcije, križanja i mutacije. Programi se prema nekom kriteriju vrednuju (preciznost njihovih rezultata, brzina izvođenja, minimalni broj instrukcija, itd) i u konačnici se najbolji predstavlja kao konačno rješenje.

Uz ovakav opis, na prvi pogled se zaključuje da je genetsko programiranje samo poseban slučaj genetskog algoritma. Ipak, zbog činjenice da se radi o optimizaciji računalnih programa, što je jako specifično područje, te zbog stalnog napretka u istraživanju tog područja, genetsko programiranje se navodi kao poseban evolucijski algoritam.

3.1.1. Računalni program – jedinka populacije

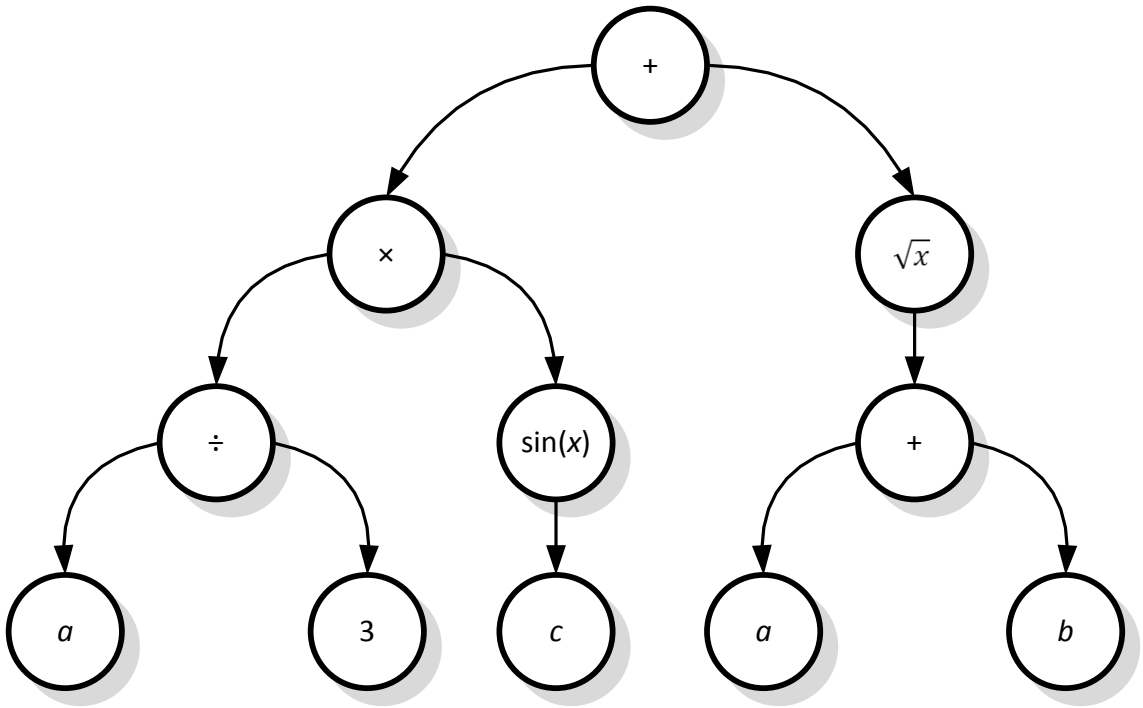
Kada se priča o računalnim programima, najčešće se misli na strukturiran slijed naredbi u nekom programskom jeziku (Tablica 3.1). Jezični procesori današnjice za vrijeme prevođenja programa primjenjuju mnoge transformacije napisanih naredbi. Rezultat obrade jest program koji radi brže i pritom troši manje spremničkog prostora, ali bez promjene funkcionalnosti. Takve transformacije gotovo sigurno ne dovode do optimalnog programa, ali svakako popravljaju kritične dijelove u iznimno kratkom vremenu.

Tablica 3.1. Primjer jednostavnog programa u programskom jeziku C

```
int main()
{
  printf("Hello world!");
  return 0;
}
```

Danas se takvi rezultati optimiranja smatraju zadovoljavajućima, ali već sutra mogu postati nedovoljno dobri. Kada se pojave takvi zahtjevi, genetsko programiranje bi moglo postati odgovor na njih. Trenutno, ono nije doraslo rješavanju takvih problema, već se problemi pojednostavljuju na programe građene od jednostavnih matematičkih operatora i funkcija – na matematičke izraze.

Jedinka populacije genetskog programiranja je funkcija predstavljena stablastom strukturom, nazvanom stablo izraza (engl. *expression tree*). Ono je građeno od nezavršnih i završnih simbola. Prvi se pojavljuju samo kao unutarnji čvorovi, dok se potonji javljaju kao vršni čvorovi (listovi) stabla. Nezavršni simboli su matematički operatori i funkcije, svaki s poznatim brojem argumenata i jednom vrijednošću kao rezultatom. Svaki izlaz čvora predstavlja jedan od ulaza njegovog roditelja. Izlaz iz korijena stabla daje vrijednost funkcije za njene ulazne argumente. Ulazni argumenti funkcije određeni su preko završnih simbola, razmještenih u listovima stabla. Završni simboli su ili varijable ili konstante. Ukoliko se u listove smjeste samo konstante, onda i funkcija postaje konstantna.



Slika 3.1. Primjer funkcije zadane u obliku stabla izraza

Slika 3.1. prikazuje jedno jednostavno stablo izraza. Vidljivo je kako listove stabla čine samo ulazne varijable funkcije (*a*, *b* i *c*) i konstanta 3. Unutarnji čvorovi su aritmetički operatori i poznate matematičke funkcije. Pretvorbom gornjeg stabla izraza u standardni matematički zapis dobiva se sljedeći izraz:

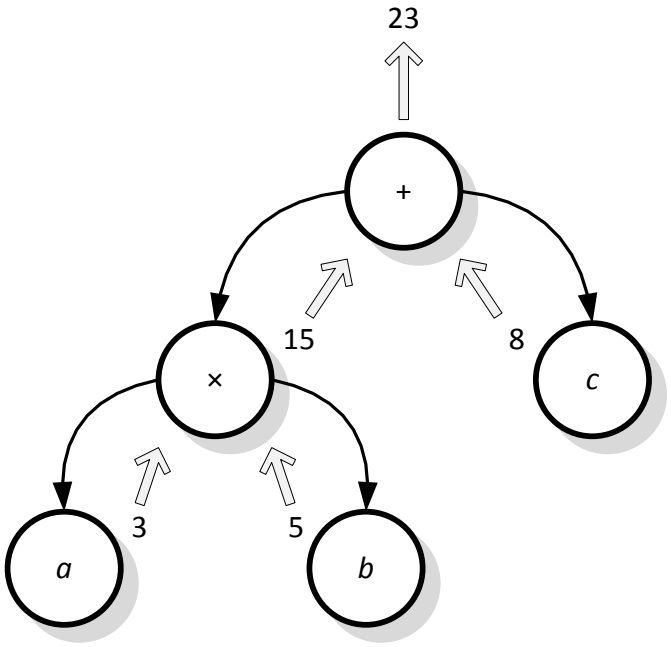
$$f(a,b,c) = \left[\left(\frac{a}{3}\right) \cdot \sin(c)\right] + \left[\sqrt{(a+b)}\right] = \frac{a}{3} \cdot \sin(c) + \sqrt{a+b} \qquad \textbf{(3.1)}$$

Kao što se vidi, zagrade osiguravaju jednoznačnu izgradnju stabla izraza, jer bi se bez njih moglo izgraditi više različitih stabala od iste početne funkcije.

Stablo izraza nije jedina korištena reprezentacija jedinke. Danas se istražuju i koriste razne strukture za spremanje i optimiranje programa sličnijih imperativnim jezicima. Podvrsta genetskog programiranja koja koristi takve strukture se zove *linearno genetsko programiranje* [9].

3.1.2. Evaluacija i dobrota računalnog programa

Kako bi se računalnom programu mogla odrediti dobrota, prvo ga je potrebno evaluirati. Pošto je riječ o jednostavnoj strukturi, evaluacija iste je također jednostavna. Prvo se listovima pridruže ulazne vrijednosti, koje se zatim prenose roditeljima dok se ne dobije vrijednost funkcije kao vrijednost izlaza iz korijena stabla (Slika 3.2).



Slika 3.2. Primjer evaluacije stabla izraza za (vrlo) jednostavnu funkciju

Ukoliko se genetskim programiranjem traži funkcija $g(x)$ koja najbolje opisuje zadani skup točaka $X = \{(x_1, y_1), (x_2, y_2), \dots, (x_{k-1}, y_{k-1}), (x_k, y_k)\}$, kao mjeru kvalitete jedinke bolje je uzeti funkciju kazne:

$$kazna(g) = \sum_{i=1}^{k} [g(x_i) - y_i]^2 \qquad \textbf{(3.2)}$$

Ona predstavlja sumu kvadrata odstupanja trenutne funkcije od očekivanih iznosa (uzoraka). Razlika se kvadrira kako bi se dodatno kaznile jedinke koje više odstupaju. Ovo kao posljedicu može imati brzu homogenizaciju populacije, zbog čega je nužno dobro

odrediti ostale parametre genetskog programiranja (vjerojatnosti križanja i mutacije, veličinu populacije, itd).

3.2. Proširenje genetskog programiranja genskim izrazima

Povećanje dimenzije problema ili samog opsega pretraživanja, rezultirat će značajnim usporenjem rada genetskog programiranja. Uzrok tolikom usporenju leži u činjenici da se u svakoj novoj jedinki, bilo slučajno generirana, bilo dobivena nekim genetskim operatorom, mora provjeriti odgovaraju li njeni unutarnji čvorovi funkcijskim zahtjevima. Na primjer, križanjem dvije jedinke dobio se slučaj da čvor označen operacijom sinus ima dva djeteta, a očekivalo se samo jedno. Takvu nevaljalu jedinku treba ukloniti i na njenom mjestu stvoriti novu.

Tablica 3.2. Pseudokod programiranja genskih izraza

```
Programiranje_genskih_izraza
{
    t = 0;

    generiraj početnu populaciju genskih izraza P(0);
    generiraj početno prihvatljivo rješenje;

    sve dok nije zadovoljen uvjet završetka evolucijskog procesa
    {
        t = t + 1;

        izgradi sva stabla izraza za P(t-1);
        izvrši i evaluiraj svaki program u P(t-1);
        spremi najbolji program u P(t);

        selektiraj jedinke iz P(t-1) u P'(t);

        unutar P'(t) radi
        {
            mutiraj jedinke;
            premjesti odsječke gena jedinki;
            premjesti gene jedinki;
            rekombiniraj jedinke s jednom i dvije točke prekida;
            rekombiniraj gene;
        }

        spremi P'(t) u P(t);
    }

    ispiši rješenje;
}
```

Usto, genetsko programiranje radi sa stablima, a operacije vezane za rad sa stablima su sporije od rada s linearnim strukturama (nizom ili listom). Povećavanjem veličine jedinki, u želji za dobivanjem boljeg rješenja, algoritam se dodatno usporava zbog tog negativnog

svojstva funkcija za rad sa stablima. Zbog toga, javila se potreba za vraćanjem genetske poruke u linearni oblik, a da se opet iz nje može rekonstruirati računalni program. Nova metoda optimiranja koja donosi rješenje zove se programiranje genskih izraza (engl. *gene expression programming*, GEP) [10].

Programiranje genskih izraza predstavila je dr. Cândida Ferreira 1999. godine. Metoda je revolucionarna zbog uvođenja novog, dvojakog gledanja na jedinku: njenu linearnu genetsku poruku (genotip) i stablo izraza izgrađeno iz genetske poruke (fenotip). Usko povezani par genotip-fenotip analogan je živim bićima u prirodi, gdje se iz linearne strukture molekule DNK gradi cijeli organizam jedinke. Kako se jedinke u prirodi međusobno uparuju i reproduciraju na temelju tjelesnih svojstava jedinki, a zapravo se prenosi njihova genetska poruka, tako se kod programiranja genskih izraza evaluira fenotip, a njegov genotip provodi kroz žrvanj genetskih operatora.

Tablica 3.2. prikazuje pseudokod algoritma programiranja genskih izraza. Većina navedenih operatora je zasnovana na onima od genetskog algoritma, naravno, uz potrebne prilagodbe specifičnostima algoritma. Detaljnije o karakteristikama operatora programiranja genskih izraza nalazi se u poglavlju 3.2.4.

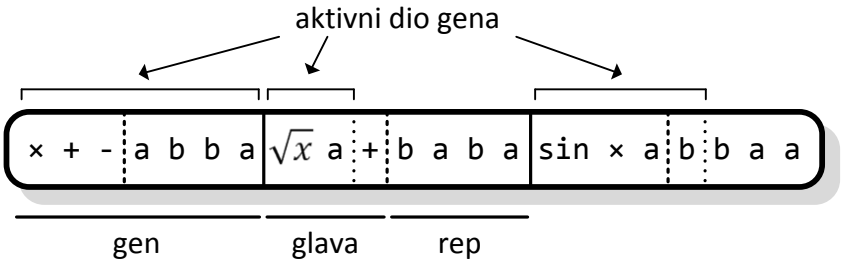
3.2.1. Genski izraz – jedinka populacije

Život jedinke počinje stvaranjem njenog genotipa, koji se još naziva i kromosomom. Struktura kromosoma je jednostavna. Svaki kromosom je građen od jednog ili više gena, gdje je svaki gen jednake duljine. Gen se dalje dijeli na glavu i rep, oboje građenih od simbola koji predstavljaju elemente funkcije. U glavi se mogu naći i nezavršni i završni simboli, dok se u repu pojavljuju isključivo završni simboli. Kao i kod genetskog programiranja, nezavršni simboli su matematičke funkcije i operatori, upareni s potrebnom brojem ulaznih argumenata (npr. funkcija tangens ima jedan argument, a operator zbrajanja dva); završne simbole čine varijable i konstante.

Kako je rečeno, pretpostavka je da su geni jednake duljine. Kako bi se ona mogla izračunati, potrebno je doznati dva podatka: duljinu glave i najveći od brojeva argumenata nezavršnih simbola. Prvi podatak je ulazni parametar u algoritam, dok se drugi određuje na početku rada, prolazom kroz listu mogućih nezavršnih simbola. Jednadžba po kojoj se od duljine glave h i najvećeg broja argumenata n_{arg} računa duljina repa t, glasi:

$$t = h \cdot (n_{arg} - 1) + 1 \qquad \textbf{(3.3)}$$

Slika 3.3. prikazuje kromosom od tri gena. Duljina glave gena je 3 simbola, najveći broj argumenata imaju aritmetički operatori i on iznosi 2, zbog čega je duljina repa 4 simbola. Jednadžba za računanje duljine repa daje dovoljno dug rep kako bi svaka funkcija ili operator u glavi „dobila“ dovoljan broj ulaznih vrijednosti.



Slika 3.3. Genski izraz (kromosom) sastavljen od tri gena

Često se ne koriste svi elementi repa u izgradnji fenotipa. Aktivni genetski materijal (onaj koji se koristi u fenotipu), označen je oznakama intervala iznad svakog gena kromosoma. Neaktivni genetski materijal (intron[6]) ne igra ulogu u tom fenotipu, ali se može aktivirati nekom od genetskih promjena na kromosomu. Vrijedi i obrat, tj. aktivni simboli se mogu deaktivirati. Na primjer, ako mutacija promijeni prvi simbol bilo kojeg gena u varijablu *a*, onda ostatak gena postaje neaktivnim. O toj temi više u poglavlju 3.2.2.

3.2.2. Izgradnja stabla izraza iz genskog izraza

Izgradnja stabla izraza za kromosome s jednim genom

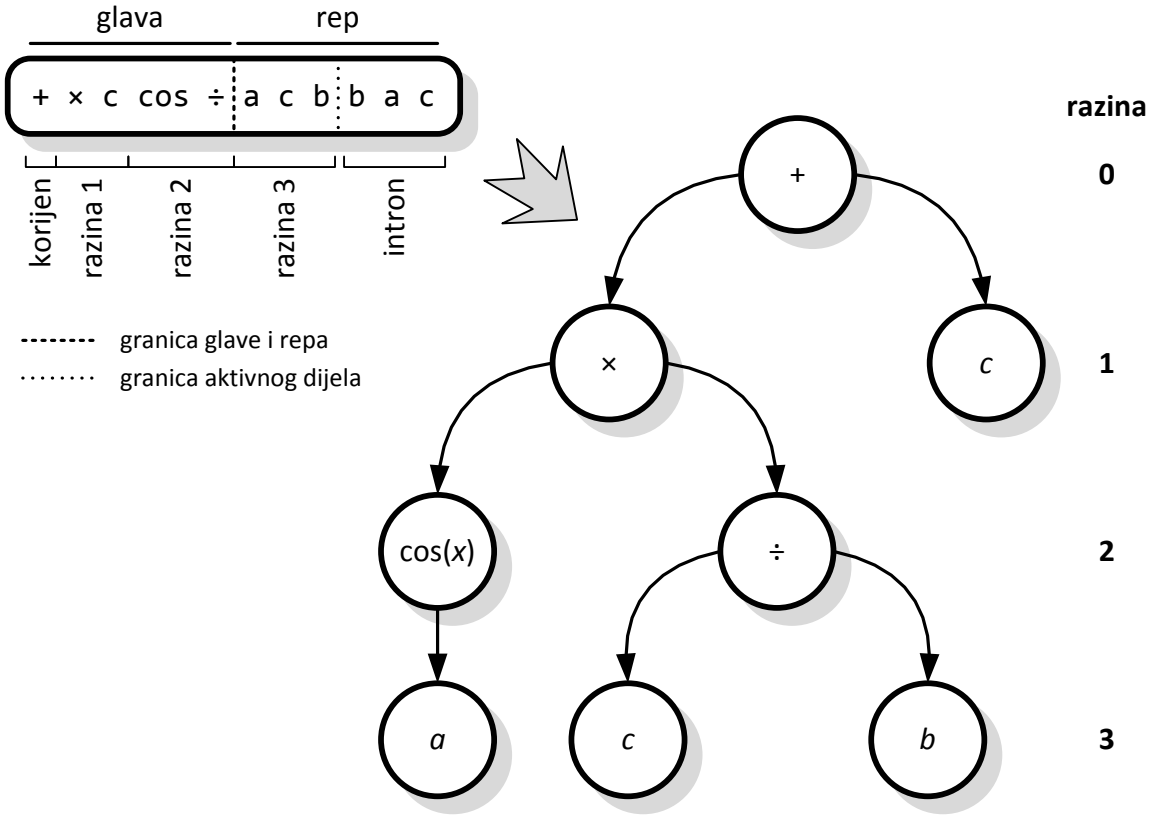
Struktura kromosoma opisana u prethodnom poglavlju zapravo predstavlja stablo izraza izravnato u niz. Ipak, kod programiranja genskih izraza stablo izraza se uvijek gradi iz genetske poruke, ali se ono nikad ne izravnava u kromosom. Kako se sve genetske operacije vrše nad kromosomima, a ne nad stablima izraza, obrnuti postupak jednostavno nije potreban. Izgradnja stabla izraza vrši se na sljedeći način [10]:

1. Prvi element gena postavlja se kao korijen stabla, po definiciji je to nulta razina.
2. Spušta se na novu razinu i redom uzimaju sljedeći simboli iz gena. Uvijek se u istom smjeru (npr. slijeva nadesno) popunjavaju mjesta za djecu razine iznad.
3. Postupak popunjavanja se ponavlja sve dok se ne dođe do razine u kojoj su postavljeni samo završni simboli.

Slika 3.4. prikazuje stablo izraza izgrađeno od priloženog gena, 11 simbola dugog (pet čine glavu, a šest rep). Simboli su redom obuhvaćeni intervalima, gdje svaki interval predstavlja simbole na istoj razini. Ovaj postupak je jednoznačan u oba smjera, dakle postoji bijekcija između dva zapisa jedinke.

Vidljivo je da se tri simbola desno od točkaste linije (intron) ne koriste u izgrađenom stablu, ali se mogu u budućnosti aktivirati radom genetskih operatora. O potencijalu genetske poruke zapisane u intronima bit će više riječi u idućim poglavljima. Također, važno je navesti poseban slučaj kada je korijen stabla završni simbol, a što nije rijetka pojava. Tada on sâm predstavlja cijelo stablo izraza, jer završni simboli nemaju djecu.

[6] Introni su neaktivni isječci DNK, zaostali kroz evoluciju unutar molekule. Oni danas nemaju poznatu funkciju, a pretpostavka je da su nekad u prošlosti bili aktivni dio DNK (egzoni).



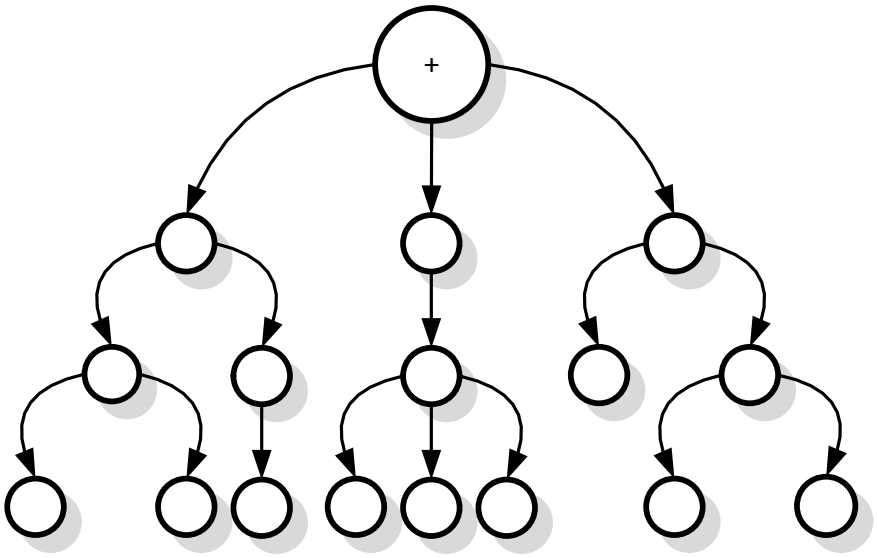
Slika 3.4. Jednoznačna izgradnja stabla izraza iz kromosoma od jednog gena

Izgradnja stabla izraza za kromosome s više gena

U praksi se češće koriste kromosomi sastavljeni od više gena. Kako je stablo izraza samo jedno za svaki kromosom, očito se na neki način moraju povezati izgrađena stabla izraza gena u jednu cjelinu. Za spajanje stabala u takvim slučajevima koristi se povezujuća funkcija (engl. *linking function*). Povezujuća funkcija je ulazni parametar u algoritam i zajednička je za sve kromosome. Za nju se najčešće uzima neka jednostavna aritmetička operacija, ona koja nema ograničenje na broju argumenata, jer kromosom može imati proizvoljan broj gena.

Povezujuća funkcija se koristi nakon što se izgrade stabla izraza svih gena kromosoma. Zatim se korijeni tih stabala postavljaju kao djeca čvora povezujuće funkcije. Nakon ovog koraka, formiralo se jedno veliko stablo izraza, spremno za evaluaciju (Slika 3.5).

Dodatna istraživanja na polju programiranja genskih izraza pokazala su da povezujuća funkcija ne mora uvijek biti implicitno određena na početku rada algoritma. Naime, moguće je proširiti zapis kromosoma podatkom o povezujućoj funkciji npr. simbol na početku kromosoma. Na taj način bi i povezujuća funkcija bila podložna djelovanju genetskih operatora. Analize pokazuju da se i takvim zapisom postižu dobri rezultati optimiranja [10].



Slika 3.5. Primjer zbrajanja kao povezujuće funkcije tri podstabla izraza

3.2.3. Evaluacija i dobrota stabla izraza

Budući da je programiranje genskih izraza proširenje genetskog programiranja, evaluacija je praktički ista. Razlika je jedino u tome što se kod programiranja genskih izraza stablo prvo mora izgraditi, jer se ono ne čuva kao jedinka populacije niti se na njemu obavljaju genetske operacije. Naime, nakon svake genetske promjene kromosoma, staro stablo se briše te se ispočetka gradi novo.

3.2.4. Prilagodbe osnovnih i specifični genetski operatori

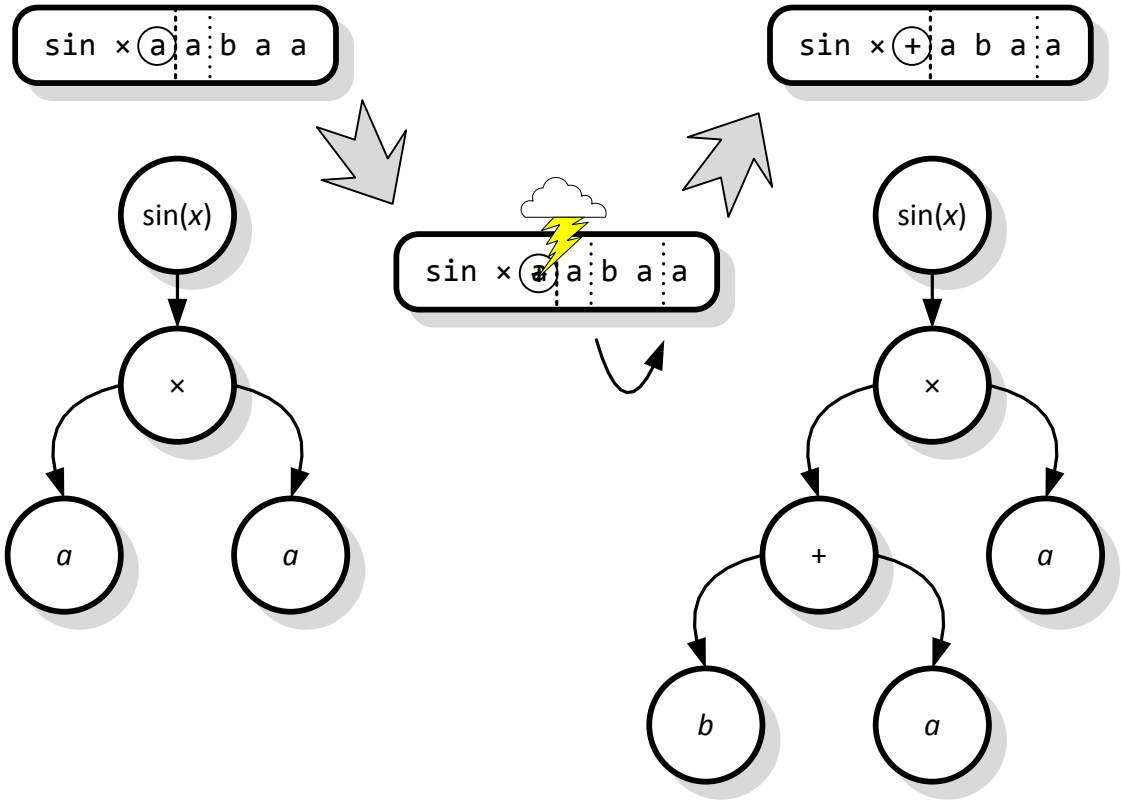
Lineariziranjem genetske poruke značajno se ubrzava izvođenje genetskih operatora te se olakšava njihovo programiranje. Također, mnogim programskim jezicima je inherentan rad s listama (poput Lispa i funkcijskih jezika), dok je takva podrška za stabla dosta rjeđa pojava. Većina operatora programiranja genskih izraza je slična operatorima genetskog algoritma, zbog čega će se naglasiti samo specifičnosti za prvu metodologiju.

Mutacija

Mutacija se može dogoditi bilo gdje u kromosomu, s jednakom vjerojatnošću za svaki gen i za svaki simbol u genu. U implementaciji, najčešće se koristi *točkasta mutacija* (engl. *point mutation*). Kod nje se prvo slučajno odabere jedan simbol unutar kromosoma. Ako je odabrani simbol smješten u glavi gena, onda on može mutirati u bilo koji iz skupa nezavršnih i završnih simbola. Naprotiv, ako je smješten u repu gena, onda on može mutirati isključivo u neki završni simbol.

Slika 3.6. oslikava izvođenje točkaste mutacije na priloženom genu duljine sedam simbola (tri čine glavu, a četiri rep). Uz početni i mutirani gen prikazano je odgovarajuće stablo izraza. Odabran je treći simbol gena, koji je iz završnog simbola *a* mutirao u nezavršni simbol operatora zbrajanja. Ovakva promjena je moguća jer se odabrani simbol nalazi u

glavi gena (lijevo od crtkane linije). Kao posljedica mutacije, aktivni genetski materijal se proširio za dodatna dva mjesta (pomak točkaste linije udesno), čime je u konačnici intron spao na duljinu od samo jednog simbola. Rezultat mutacije je novo stablo izraza, s jednom granom dodatno produženom i povećanim ukupnim brojem čvorova.



Slika 3.6. Prikaz slijeda mutacije gena i promjene na stablu izraza

Ono što se može naslutiti, već iz ovako jednostavnog primjera, jest velika moć operatora mutacije. On je u stanju iz temelja promijeniti fenotip jedinke i time u potpunosti okrenuti smjer optimizacije. U slučaju da je mutacija nastupila na prvom elementu gena, mutirani fenotip više ne bi niti podsjećao na početni. Suprotno ovome, mutacija unutar introna se ne bi odrazila na izgledu fenotipa (stabla izraza), ali ostaje upamćena za ubuduće.

Operatori za premještanje podniza unutar jedinke

Druga skupina operatora programiranja genskih izraza čine operatori premještanja podniza gena (engl. *sequence transposition*). Ovi operatori uzimaju podniz kromosoma i prebacuju ga na drugu poziciju. Odabrani podniz se naziva premjestivi element (engl. *transposable element*), a postoje tri njegove vrste [10]:

1. Kratak dio gena s nezavršnim ili završnim simbolom na prvom mjestu, koji se prepisuje u glavu gena, osim na njeno prvo mjesto (korijen) – niz za umetanje (engl. *insertion sequence*, IS).
2. Kratak dio gena s nezavršnim simbolom na prvom mjestu, koji se prepisuje na prvo mjesto glave – niz za umetanje korijena (engl. *root insertion sequence*, RIS).

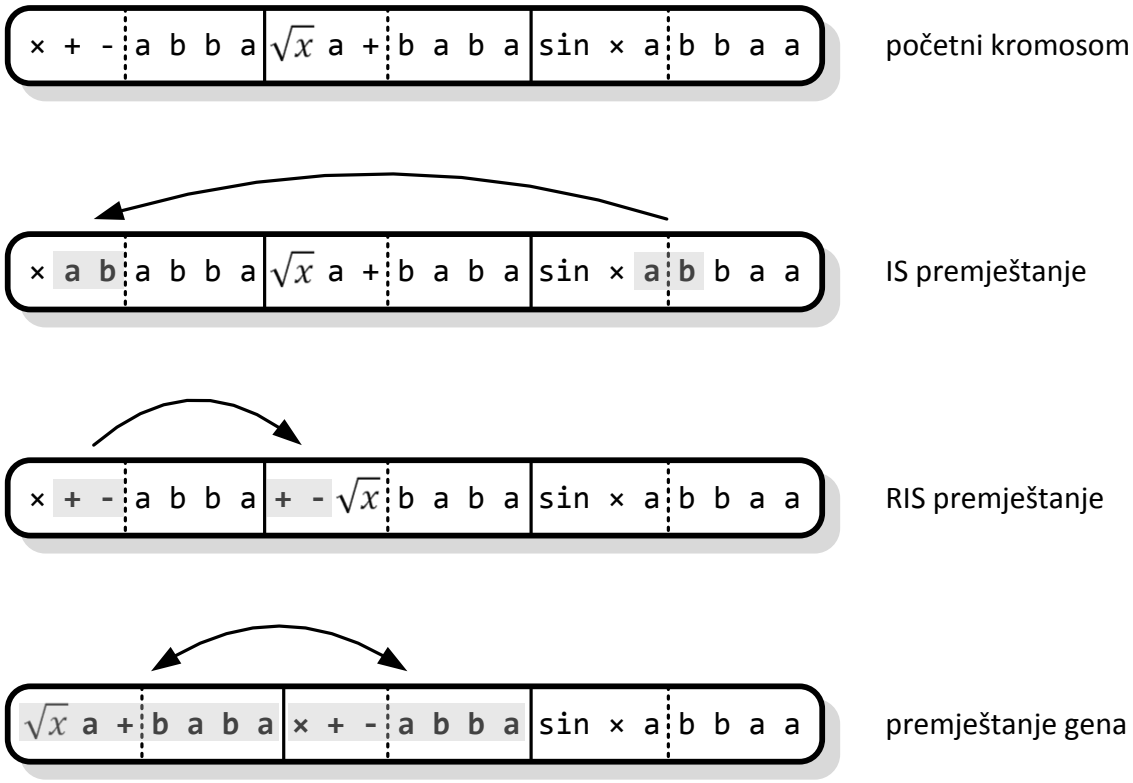
3. Kompletan gen koji se premješta na početak kromosoma.

Prvu vrstu koristi operator *premještanja niza za umetanje* (engl. *IS transposition*). Počinje se odabirom mjesta u glavi gena gdje će se ubaciti kopija premjestivog elementa. Naravno, i gen i mjesto ubačaja se biraju slučajnim putem. Drugi korak jest odabir niza za umetanje. Kako on cijeli mora „stati“ u glavu, potrebno mu je duljinu ograničiti razlikom ukupne duljine glave i mjesta ubačaja niza. Na ovaj način se ne prepisuje više elemenata od mogućeg. Dakle, slučajno se odabiru gen, početak niza i njegova duljina (uz ograničenje). Konačno se odabrani niz kopira u glavu. Svi simboli prije pozicije ubačaja ostaju na svome mjestu, a oni nakon nje se guraju prema kraju glave. Između se umeće premjestivi element. Kako je glava fiksne veličine, svi simboli koji bi trebali preći u rep jednostavno se brišu pa rep ostaje netaknut ovim operatorom. Struktura kromosoma ostaje očuvana, što znači da je i program ostao sintaktički ispravan.

Druga vrsta premjestivog elementa koristi se u operatoru *premještanja niza za umetanje korijena* (engl. *RIS transposition*). On zahtijeva da premjestivi element počinje funkcijom (nezavršnim simbolom), zbog čega se niz za umetanje traži samo u glavi. Slučajno se bira gen i pozicija u njegovoj glavi. Ako je na pogođenoj poziciji završni simbol, kreće se prema kraju glave i traži prvi nezavršni. Kad ga se nađe, uzima se slučajan podniz od njega do kraja glave, a ukoliko se ne pronađe završni simbol, operator ništa ne mijenja u kromosomu. Kopija podniza se zatim ubacuje kao početak glave slučajno odabranog gena, odbacujući jednako toliko simbola s kraja glave. I u ovom slučaju je rep odredišnog gena očuvan, a s njime i struktura kromosoma. Program izgrađen iz njega će se normalno izvršiti jer mu sintaktička ispravnost nije narušena.

Posljednju varijantu premjestivog elementa koristi operator *premještanja gena* (engl. *gene transposition*). Ovo je najjednostavniji operator za izvedbu. Slučajno se odabere kromosom i gen koji će se premjestiti. On se zatim premješta na prvo mjesto u tom kromosomu, pomičući sve gene prije njega jedno mjesto unatrag. Za razliku od prethodna dva operatora, ovdje je zaista riječ o premještanju, tako da je očuvana duljina kromosoma. Operator premještanja gena je utoliko besmislen ako je povezujuća funkcija asocijativna (poput zbrajanja ili množenja). Ipak, postoje problemi koji ne koriste asocijativne funkcije za povezivanje gena i tada ovaj operator može značajno utjecati na kvalitetu jedinke.

Slika 3.7. prikazuje moguća djelovanja sva tri operatora premještanja na početni kromosom sastavljen od tri gena. Operatori premještanja su ostatak razvojnog procesa metodologije programiranja genskih izraza, jer je u početku kromosome činio samo jedan gen. U takvom je okruženju kromosom sa završnim simbolom na početku rijetko bio koristan [10].



Slika 3.7. Prikaz rada svih operatora premještanja

Operatori rekombinacije (križanja)

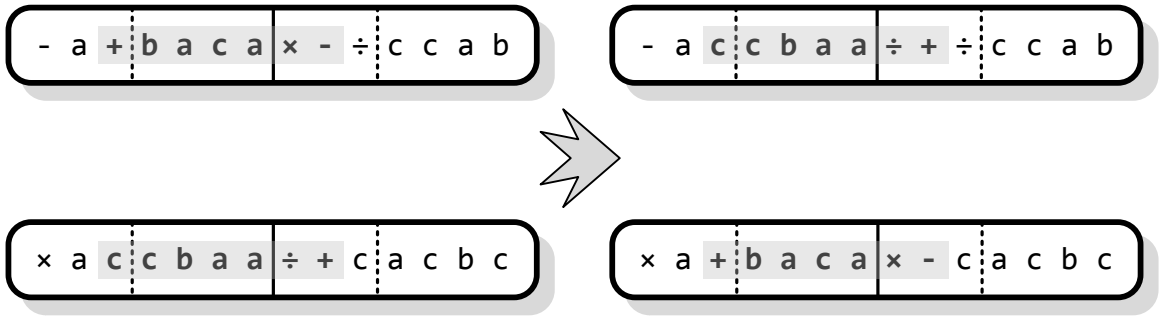
U radu programiranja genskih izraza koriste se četiri operatora rekombinacije ili križanja (engl. *recombination*). Oni su već detaljno opisani u poglavlju o križanju kod genetskog algoritma (poglavlje 2.4.3), jer se zapravo radi o varijantama križanja s jednom ili dvije točke prekida. Ipak, neki detalji su specifični za programiranje genskih izraza pa će se u ovom poglavlju staviti naglasak na njih. Uniformno križanje je također prilagođeno novom zapisu kromosoma.

Rekombinacija je operator koji u svom radu slučajnim odabirom prvo izdvaja dvije jedinke (roditelja) iz populacije. Zatim roditelji međusobno razmjenjuju svoj genetski materijal da bi se u konačnici dobile jedinke-djecu. One se vraćaju u populaciju i dalje sudjeluju u optimizacijskom procesu. Implementacija algoritma najčešće koristi četiri vrste operatora rekombinacije [10]:

1. križanje s jednom ili dvije točke prekida
2. rekombinacija gena
3. uniformno križanje

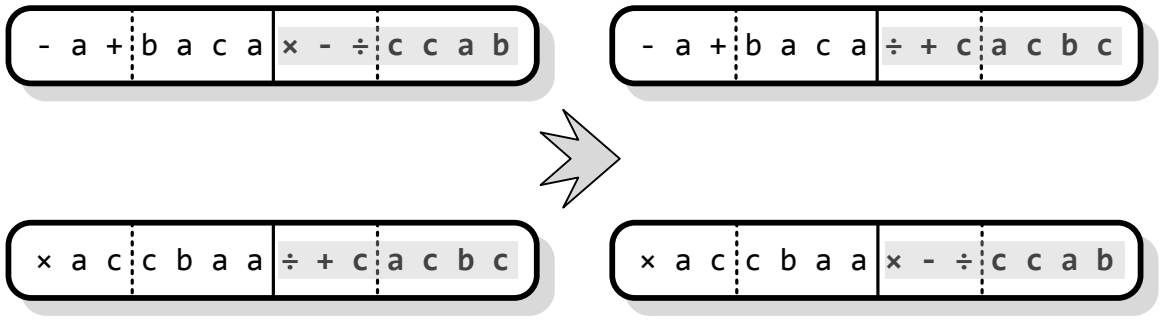
Operatori *križanja s jednom ili dvije točke prekida* (engl. *one/two-point recombination*) jako su slični. Slučajno se odabere jedna ili dvije točke na kromosomima roditelja gdje će se oni razrezati. Zatim se naizmjenično uzimaju elementi jednog i drugog kromosoma, dok se ne dobiju dva kromosoma sastavljena od dijelova roditelja (Slika 3.8). Na slici je prikazano križanje s dvije točke prekida, gdje su točke na rubovima označenog podniza

kromosoma. Kao što je vidljivo sa slike, podniz se može protezati i kroz više od jednog gena (u primjeru su kromosomi sastavljeni od dva gena).



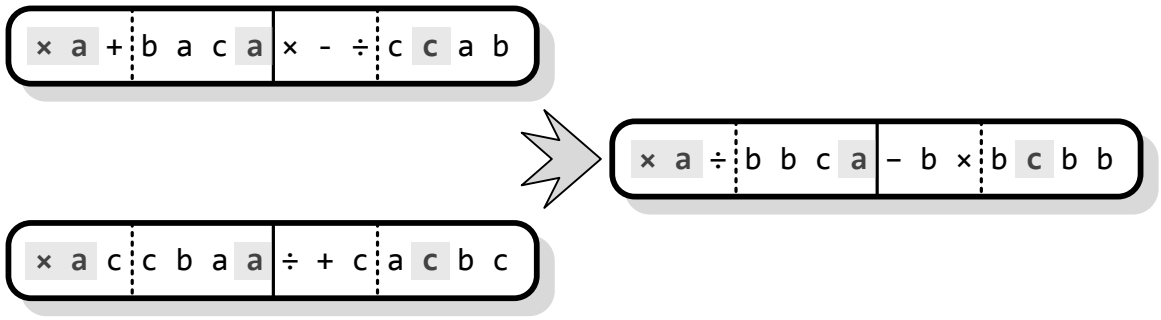
Slika 3.8. Križanje kromosoma s dvije točke prekida

Slika 3.9. prikazuje djelovanje operatora *rekombinacije gena* (engl. *gene recombination*). Slučajno se odabere gen za razmjenu (u primjeru je riječ o drugom genu) i zatim se cijeli njihov sadržaj razmijeni među roditeljima.



Slika 3.9. Rekombinacija gena kromosoma

Operator *uniformnog križanja* se razlikuje od prethodnih jer se njegovim djelovanjem dobije samo jedno dijete koje se kasnije vraća u populaciju. Slično uniformnom križanju na binarnim kromosomima, u dijete se prvo prepisuju geni roditelja koji su jednaki na nekoj poziciji. U preostala prazna mjesta se upisuju slučajno odabrani simboli, pazeći pritom nalazi li se to mjesto u glavi ili repu, kako se ne bi narušila sintaktička ispravnost programa (Slika 3.10).



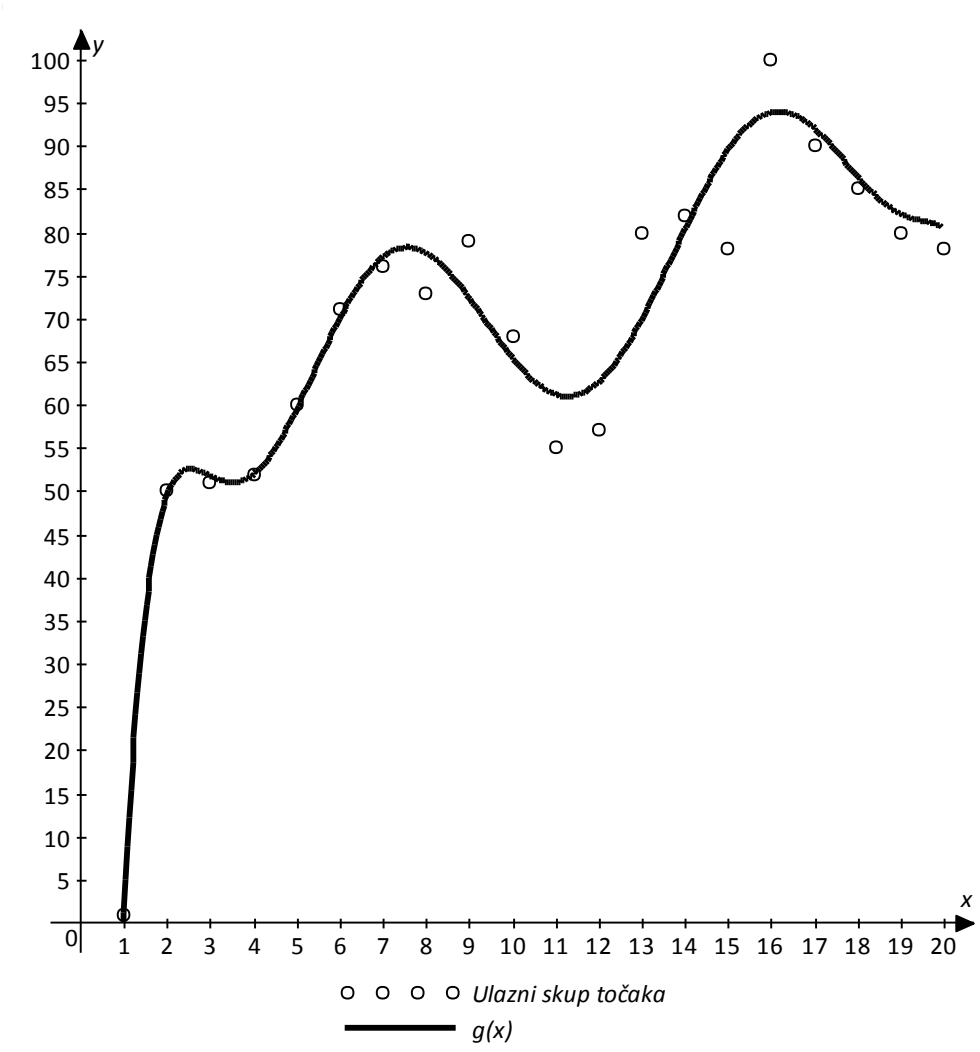
Slika 3.10. Uniformno križanje dva kromosoma

4. Aproksimacijski problem

U životu je čest slučaj kad je potrebno problem riješiti dovoljno dobrom aproksimacijom idealnog rješenja. Većina je slučajeva takvih da nije potrebna ili nije dosežna apsolutna preciznost konačnog rješenja. Kvaliteta konačnog rješenja određena je kroz dva zahtjeva: preko najveće dozvoljene pogreške ili preko raspoložive količine resursa dodijeljene za njegovo traženje (vrijeme, novac, itd).

4.1. Definicija problema

Aproksimacijski problem, kao tema ovog diplomskog rada, odnosi se na problem određivanja funkcije koja najbliže prolazi ulaznom skupu točaka. Metoda kojom se rješava opisani problem zove se *nelinearna regresija*, jer je rješenje redovito nelinearna funkcija.



Slika 4.1. Primjer rezultata nelinearne regresije na ulaznom skupu točaka

Funkcija $f(x)$ je uvijek nepoznata pa se traži nova funkcija $g(x)$ koja prolazi dovoljno blizu točaka ulaznog skupa (Slika 4.1).

4.2. Ulazni podaci

Ulazni skup točaka predstavljen je diskretnim točka. Svaka je točka određena uređenim parom (x_i, y_i), gdje je $y_i = f(x_i)$. Kako je funkcija f nepoznata, y_i se ne dobiva njenom evaluacijom, nego je otpočetka poznata samo njena vrijednost.

Na primjer, ulazni podaci predstavljaju cijene sirove nafte na tržištu u proteklih 365 dana. Očito ne postoji jasan matematički izraz kojim se može odrediti točna cijena goriva sljedećeg dana. Cijena je u skladu s uvjetima na tržištu, a izrazito je osjetljiva na razne lokalne krize. Ipak, njena vrijednost se određuje svakog dana i te se vrijednosti uzimaju kao ulazne točke za rješavanje pomoću nelinearne regresije.

4.3. Izlazna funkcija

Rezultat linearne regresije je matematička funkcija najčešće sastavljena od konačnog broja jednostavnih članova. Članovi su ili polinomni ili trigonometrijski, ovisno o pristupu rješavanja problema. Prva vrsta članova daje funkciju polinoma maksimalnog reda *r* i nije periodična. Ovo je čini pogodnom za aproksimiranje nepredvidljivih uzoraka, koji nemaju istaknut period. Naprotiv, druga vrsta članova daje funkciju s periodom, čime je ona pogodna za aproksimaciju ulaznih podataka s uočljivim periodom.

5. Praktični rad

5.1. Rješavanje jednostavnim genetskim algoritmom

Program za rješavanje zadanog aproksimacijskog problema korištenjem genetskog algoritma napisao M. Golub i objavio ga na svojim stranicama [11].

5.1.1. Programsko ostvarenje

Izvorni kod programa napisan je u jeziku C++ i predstavlja implementaciju jednostavnog genetskog algoritma (turnirska selekcija, uniformno križanje i jednostavna mutacija). Kromosomi su predstavljeni nizom bitova i predstavljaju koeficijente uz sinusne članove aproksimacijske funkcije. Funkcija cilja definirana je kao suma kvadrata odstupanja u zadanim diskretnim točkama, zbog čega se minimizira.

Aproksimacijska funkcija $g(x)$ se zadaje do k-tog sinusnog člana u sljedećem obliku:

$$g(x) = a_0 + a_1 \cdot x + \sum_{i=1}^{k} [a_{3i-1} \cdot \sin(a_{3i} \cdot x + a_{3i+1})] \quad \textbf{(5.1)}$$

Jedinka se stvara na način da se svi koeficijenti a_n redom poslažu u niz te se kasnije nad njom i ostalim jedinkama obavljaju razne genetske operacije. Rezultat je oscilirajuća funkcija koja prolazi blizu ulaznih točaka.

Ulazne točke se definiraju u posebnoj datoteci u kojoj se nalaze svi podaci nužni za pokretanje algoritma. Pokretanjem programa, ova se datoteka čita i iz nje se izvlače korisni podaci. Nakon pripreme ulaznih podataka, algoritam počinje s radom. Za vrijeme rada, svaki željeni broj iteracija se ispisuje najbolja jedinka populacije korisniku na ekran. Kako je riječ o iznimno brzom algoritmu, taj željeni broj bi trebao biti reda veličine tisuće, a ukupan broj iteracija se može postaviti na nekoliko desetaka tisuća. Završetkom optimiranja, program ispisuje najbolje rješenje na ekranu i prestaje s radom.

5.1.2. Pokretanje procesa optimiranja

Program je prilagođen radu na operacijskim sustavima Unix ili Linux, zbog čega ga je potrebno prevesti pomoću prevoditelja na toj platformi (gcc ili g++). Nakon uspješnog prevođenja, obavezno se definiraju parametri problema u posebnoj datoteci imena „22“. U njoj su popisani parametri zajedno s objašnjenjima što koji predstavlja pa se ovdje neće navoditi niti opisivati. Ukoliko se za ime izvršne datoteke odabralo ime „GA“, program se pokreće jednostavnom naredbom u komandnoj liniji:

```
$ ./GA 22
```

Program u radu ispisuje najbolje međurezultate na ekranu, a kad se obave sve iteracije algoritma, najbolje pronađeno rješenje (Tablica 5.1). Važno je napomenuti da se konačno rješenje ispisuje na ekranu, ali ne i u nekoj datoteci, pa je potrebno dobiveno rješenje ručno sačuvati nakon završetka rada programa.

Tablica 5.1. Primjer ispisa stanja iteracije

```
…
 plot "22", 5.668812+3.770904*x+3.717413*sin(1.123434*x+9.955155)+11.
697889*sin(0.689043*x+3.122941)+-2.108214*sin(1.626793*x+-5.656446)+2
.028031*sin(1.807326*x+-8.189608)+14.352562*sin(0.185584*x+3.752040)+
5.751334*sin(1.494461*x+-3.763075)+-10.421261*sin(1.702628*x+-0.00404
7)+1.944271*sin(1.999225*x+-7.500324)+-8.549360*sin(1.803168*x+-5.004
815)+4.356705*sin(1.249228*x+3.747235)+-1.474038*sin(0.542869*x+-4.32
4821)+1.582881*sin(0.964173*x+-9.024443)
pause -1 "# 61999 # f(x)=-274.076265"
…
```

5.2. Rješavanje programiranjem genskih izraza

5.2.1. Programsko ostvarenje

Programiranje genskih izraza je metodologija koja se uvelike oslanja na rad s listama i izvršavanje programskih struktura u obliku stabla. Zbog takvih zahtjeva, neki od popularnijih jezika, poput jezika C++, nisu dolazili u obzir. Stoga je odabran Common Lisp, programski jezik kojem su tražene funkcionalnosti ugrađene u samom jeziku.

Lisp je specifičan programski jezik u kojem su i podaci i naredbe prikazani istom, specifičnom strukturom, nazvanom *S-izraz* (engl. *S-expression*). „S“ dolazi od „simbol“, jer su elementi S-izraza simboli, a cijeli izraz je uokviren okruglim zagradama. Ukoliko je prvi element S-izraza simbol vezan za neku funkciju, npr. `(zbroji 1 2 3)`, onda se toj funkciji šalju ostali elementi izraza kao ulazni argumenti. Primjer se transformira u poziv `zbroji(1, 2, 3)` što daje broj 6 kao rezultat poziva. Također, moguće je posebnom sintaksom onemogućiti pozivanje funkcije, što će S-izraz ostaviti nepromijenjenim te će se njegovo izvođenje (evaluacija) odgoditi za prikladniji trenutak.

Tablica 5.2. Primjer S-izraza za kromosom od dva gena

```
((+ sin log x x x x)
 (÷ exp cos x x x x))
```

U implementaciji diplomskog zadatka, kromosomi su predstavljeni listom gena, gdje je svaki gen jedinstvena lista simbola – nisu podijeljeni na liste glave i repa. Tablica 5.2. prikazuje primjer jednostavnog kromosoma od dva gena u obliku S-izraza. Važno je uočiti da se kromosom u ovakvom obliku drži u memoriji, bez obzira što postoje funkcije

operatora „+“ i „÷“. Ovakvo ponašanje je potrebno, jer se ne evaluira kromosom u linearnom obliku, nego njegov stablasti ekvivalent.

Kako je sve u Lispu S-izraz, isto je i sa stablom izraza (Tablica 5.3). Ono je građeno od manjih stabala, konstruiranih iz pojedinih gena, a povezanih zajedničkom povezujućom funkcijom. Za primjer u tablicama odabrano je zbrajanje, jer je ono najčešći odabir.

Tablica 5.3. Izgrađeno stablo izraza iz prethodnog primjera u obliku S-izraza

```
(+
   (+
      (sin x)
      (log x))
   (÷
      (exp x)
      (cos x)))
```

U trenutku izvršavanja S-izrazi se gledaju kao matematički izrazi u prefiksnom zapisu. Ovaj način zapisa nešto je teži ljudima za čitanje od uobičajenog, infiksnog zapisa. Naprotiv, računalu ono predstavlja mnogo jednostavniju strukturu, lakšu za obradu i evaluaciju, jer nema potencijalnih dvosmislenosti kao kod infiksnog zapisa.

Kako je riječ o jednodimenzijskom aproksimacijskom problemu, postoji samo jedan ulazni argument x, kojem se pri evaluaciji pridružuju vrijednosti x_i iz ulaznog skupa točaka. Vrijednost $g(x_i)$ koju izračuna stablo izraza, oduzima se od očekivane vrijednosti y_i, razlika se kvadrira i dodaje u sumu dotad izračunatih razlika (funkcija kazne iz poglavlja 3.1.2). Nakon zbrajanja svih kvadrata odstupanja, računa se dobrota jedinke prema formuli:

$$dobrota(g) = -kazna(g) \tag{5.2}$$

Ovakva funkcija dobrote osigurava put prema optimumu, prema nuli, jer je jedino tada dosegnuto idealno poklapanje funkcije i njene aproksimacije.

Tablica 5.4. Struktura jedinke korištena u implementaciji algoritma

```
(kromosom
 stablo-izraza-kao-S-izraz
 stablo-izraza-kao-strojni-jezik
 dobrota)
```

Struktura podataka jedinke u memoriji je zapravo lista od četiri elementa: kromosom, stablo izraza u obliku S-izraza, stablo izraza prevedeno u strojni jezik i vrijednost funkcije dobrote jedinke (Tablica 5.4). Stablo izraza u obliku S-izraza se čuva zbog eventualnog ispisa na ekran. Kako se ista funkcija poziva za mnogo ulaznih točaka, veliko se ubrzanje računanja dobije prevođenjem S-izraza u strojni jezik. Naime, ukoliko se spomenuto

prevođenje ne obavi, ugrađena evaluacijska funkcija `eval` to radi svaki put kad se pozove. Očito je da se zbog brzine izvođenja isplati prevesti S-izraz u strojni jezik.

Ulazne datoteke

Program koristi tri ulazne datoteke. Kako one sadrže strukture zapisane u standardnom obliku jezika Lisp, na prvi tren su malo zbunjujuće. Za vrijeme razvoja aplikacije ovog diplomskog rada, koristile su se sljedeće datoteke:

- *data.txt* – u ovoj datoteci popisane su ulazne točke kojima se traži aproksimacijska funkcija. Elementi liste predstavljaju vrijednost y_i, dok njihova pozicija i u listi predstavlja vrijednost x_i, tj. $x_i \equiv i$, za $i \in \{1, 2, \dots, n\}$.
- *functions.txt* – sadrži tablicu raspršivanja (engl. *hash table*) u kojoj se nalaze parovi ključa i vrijednosti. Ključ je simbol funkcije, a njegova vrijednost jest broj argumenata te funkcije. Na primjer, sinus je predstavljen parom `(sin . 1.)`, a zbrajanje `(+ . 2.)`. Važno je pripaziti da se prilikom dodavanja nove funkcije u tablicu, postavi ispravan broj argumenata, kako njeno pozivanje ne bi završilo u neočekivanoj iznimci (engl. *exception*) i time prekinulo proces optimiranja.
- *settings.txt* – u ovoj tablici raspršivanja se nalaze svi parametri algoritma: sve vjerojatnosti izvođenja genetskih operacija, veličina populacije, duljina glave gena, broj gena po kromosomu, povezujuća funkcija i broj iteracija. Ključevi su predstavljeni nizom znakova (engl. *string*) što olakšava traženje odgovarajućeg polja ukoliko se mijenja njegova vrijednost.

Izlazne datoteke

Uz ispisivanje najnužnijih informacija o iteraciji na ekranu, program stvara tri izlazne datoteke s rezultatima optimiranja:

- *iteration-data.txt* – datoteka s popisom vrijednosti funkcije dobrote najbolje jedinke od prve do posljednje iteracije; prikladno za prikazivanje pomoću tabličnog kalkulatora u obliku grafikona.
- *result-function.txt* – zapis najbolje pronađene funkcije *g(x)* u prefiksnom obliku; može biti jako nečitak ukoliko se koristila veća duljina kromosoma.
- *results.txt* – popis vrijednosti $g(x_i)$ za svaki ulazni x_i. Također pogodno za prikazivanje u obliku grafikona, kao usporedbu s ulaznim nizom točaka.

5.2.2. Pokretanje procesa optimiranja

Programski kod je organiziran tako da se pozivanjem jedne funkcije pokrene optimizacijski algoritam, bez naknadne korisničke interakcije. U ovom slučaju, ime glavne funkcije je `main`, a ona se poziva jednostavnom naredbom:

```
CL-USER> (main)
```

Program zatim čita podatke iz ulaznih datoteka, priprema početnu populaciju te pokreće proces optimizacije. Za vrijeme trajanja optimizacije, periodički se ispisuje napredovanje najbolje jedinke kroz iteracije (Tablica 5.5). Povećavanje dobrote najbolje jedinke znak je dobrog rada algoritma i postupno približavanje optimumu. Kad se odrade sve iteracije, ispisuje se znak „T“ (oznaka istinitosti u Lispu), koji označava uspješan završetak rada.

Tablica 5.5. Primjer ispisa dobrote najbolje jedinke svakih 100 iteracija

```
Optimal fitness: 0
i =    0: Best fitness = -1324105023.3028002000
i =  100: Best fitness = -441129105.3803288000
i =  200: Best fitness = -174028712.4330331000
i =  300: Best fitness = -120330342.2389534900
i =  400: Best fitness = -76211144.9210937300
…
i = 4000: Best fitness = -5512158.7813292510
T
```

Programi napisani u programskom jeziku Lisp izvode se unutar posebnog okruženja nazvanog *slika* (engl. *image*). Ono predstavlja apstrakciju računala prema naredbama i objektima jezika, tako da je olakšano prenošenje koda s jedne platforme na drugu. Također, *slika* ima ugrađene mehanizme za čišćenje memorije od *smeća* (engl. *garbage collection*), tj. objekata koji se više ne koriste u programu i nepotrebno zauzimaju memoriju. Takva funkcionalnost postoji u Lispu još od 1959. godine, a danas je ima gotovo svaki viši programski jezik [12].

Već je rečeno da je za implementaciju diplomskog zadatka korišten Common Lisp, moderni dijalekt originalnog Lispa[7]. Pošto je riječ o otvorenom standardu jezika, postoji više implementacija standarda, a za ovaj rad je odabrana implementacija imena CLISP. Programski kod je napisan u tekst procesoru Emacs, proširen dodatkom SLIME (*Superior Lisp Interaction Mode for Emacs*) za lakši razvoj Common Lisp aplikacija. Navedeni softver složen je u besplatni paket pod imenom *„Lisp in a Box“* [13].

[7] Svi materijali se nalaze na CD-u priloženom uz ovaj diplomski rad.

6. Analiza i usporedba rezultata optimiranja

U ovom poglavlju prikazani su rezultati traženja najbolje aproksimacijske funkcije za dva skupa ulaznih točaka – manjeg i većeg.

6.1. Aproksimiranje manjeg skupa točaka

Manji skup sastoji se od 20 proizvoljno odabranih točaka, gdje je vrijednost y_i svake točke iz intervala $[0, 100]$. Zbog brže optimizacije nad malim skupom, može se napraviti više analiza osjetljivosti na promjene ulaznih parametara algoritma. Stoga su odabrani najzanimljiviji parametri – duljina glave gena, broj gena u kromosomu, broj iteracija algoritma i broj jedinki u populaciji. Testno računalno je Intel Core2 Duo procesor na 2,13 GHz, 2 GB radne memorije i operacijski sustav Microsoft Windows XP.

6.1.1. Početno dobro rješenje aproksimacijskog problema

Tijekom razvoja i testiranja programa, postavljen je skup parametara koji osigurava konvergenciju ka zadovoljavajućem rješenju. Taj će se skup parametara, uparen s dobivenim rješenjem, uzeti kao čvrsta točka u daljnjim analizama. Tablica 6.1. popisuje ulazne parametre i njihove vrijednosti.

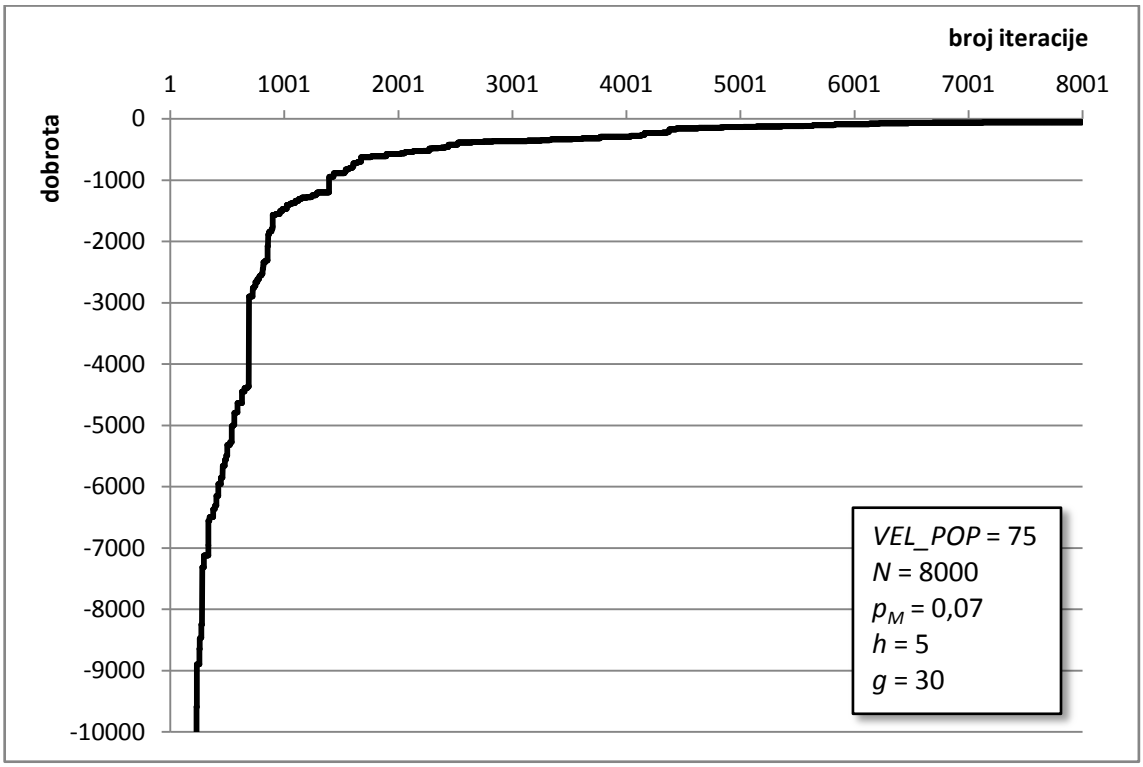
Tablica 6.1. Polazne vrijednosti ulaznih parametara algoritma

Parametar	Oznaka	Vrijednost
Broj iteracija	*N*	8000
Duljina glave gena	*h*	5
Broj gena	*g*	30
Povezujuća funkcija		+
Veličina populacije	*VEL_POP*	75
Vjerojatnost:		
- **mutacije kromosoma**	p_M	0,07
- **premještanja niza za umetanje**		0,1
- **premještanja niza za umetanje korijena**		0,1
- **premještanja gena**		0,1
- **križanja s jednom točkom prekida**		0,4
- **križanja s dvije točke prekida**		0,5
- **rekombinacije gena**		0,3

Skup raspoloživih funkcija činile su sljedeće funkcije: sinus, kosinus, tangens, e^x, kvadratni korijen i četiri aritmetičke operacije.

Algoritam optimiranja programiranjem genskih izraza pronašao je jako dobro rješenje za zadane parametre. Zahvaljujući ugrađenom elitizmu, dobrota najbolje jedinke očuvana je kroz sve iteracije te se lagano približavala globalnom optimumu (Slika 6.1). Na slici se vidi

karakterističan graf koji u početnim iteracijama jako brzo raste, da bi mu se nakon „koljena“, malo iza tisućite iteracije, značajno smanjio nagib. Takav nagib je znak približavanja nekom optimumu te da ga algoritam traži u njegovoj blizini. Algoritam asimptotski teži ka globalnom optimumu. Optimizacijski proces je trajao gotovo 45 minuta, a dobrota konačnog rješenja iznosi -58,47 jedinica dobrote.

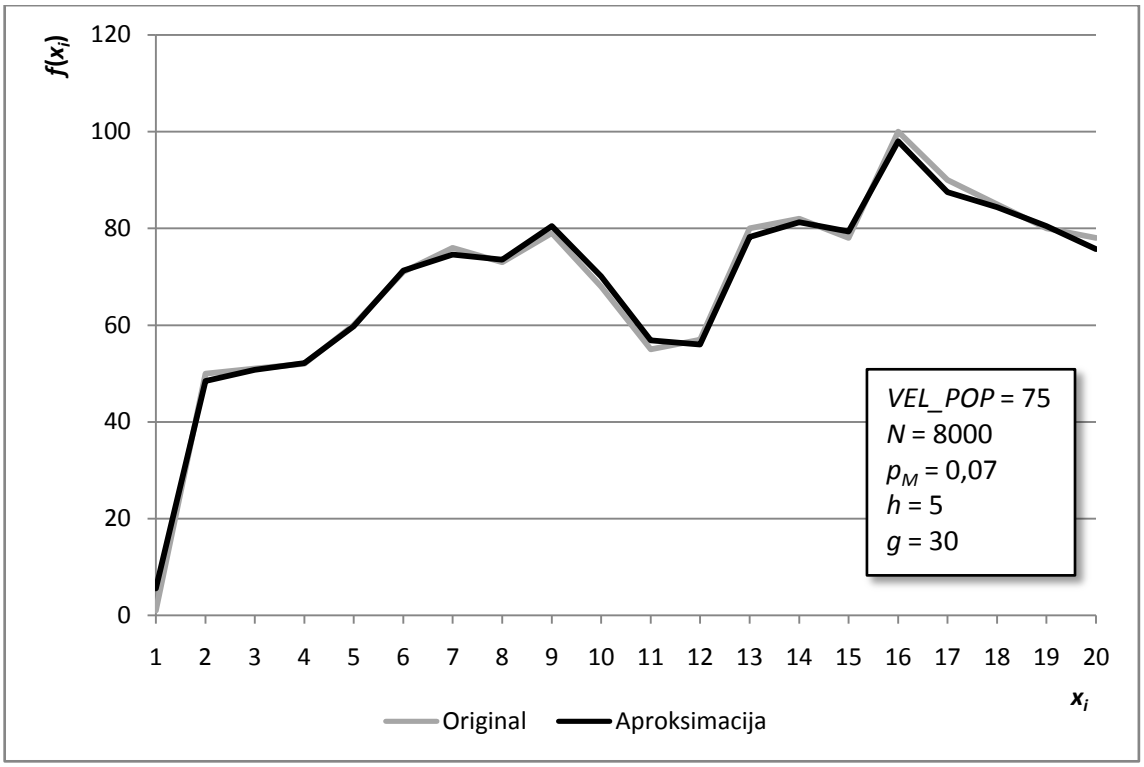


Slika 6.1. Približavanje najbolje jedinke globalnom optimumu

Tablica 6.2. daje statističku analizu kvalitete najbolje jedinke. Već je rečeno da je njena dobrota jednostavno dobivena, negiranjem *sume kvadrata odstupanja*. Dijeljenjem te sume s brojem uzoraka dobiva se *srednje kvadratno odstupanje* (pogreška, engl. *mean squared error*). Kvadratni korijen te vrijednosti daje *srednje odstupanje* aproksimacije u odnosu na originalnu funkciju. Kako ta vrijednost rijetko nešto znači, slijedi određivanje iznosa *relativnog srednjeg odstupanja* (pogreške), dijeljenjem srednjeg odstupanja i prosječne vrijednosti ulaznog skupa. Stoga, relativno srednje odstupanje najbolje jedinke iznosi jako dobrih 2,5 %.

Tablica 6.2. Kvaliteta najbolje aproksimacije manjeg skupa točaka

Podatak	Vrijednost
suma kvadrata odstupanja	58,47
broj uzoraka	20
srednje kvadratno odstupanje	2,924
srednje odstupanje	±1,71
prosječna vrijednost uzorka	68,3
relativno srednje odstupanje	≈ ±2,5 %



Slika 6.2. Poklapanje originalne funkcije i njezine aproksimacije za manji skup točaka

Slika 6.2. prikazuje poklapanje originalne funkcije i njene aproksimacije dobivene radom programiranja genskih izraza. Kako je aproksimacijska funkcija $g(x)$ iznimno složena (Tablica 6.3), njen graf je prikazan u znatno pojednostavljenom obliku. Za svaki x_i određene su odgovarajuće vrijednosti $g(x_i)$ te naknadno povezane linearnim odsječcima. Pravi izgled grafa nije pregledan, jer funkcija ima mnogo prekida i/ili skokova u beskonačnost. Razlog ovakvom izgledu grafa funkcije $g(x)$ jest nepostojanje uvjeta njene neprekinutosti. Ukoliko je postavljen, može se ispuniti korištenjem samo funkcije sinus te operacija zbrajanja i množenja, tj. upotrebom funkcija iz Fourierovog reda.

Tablica 6.3. S-izraz konačne aproksimacijske funkcije $g(x)$

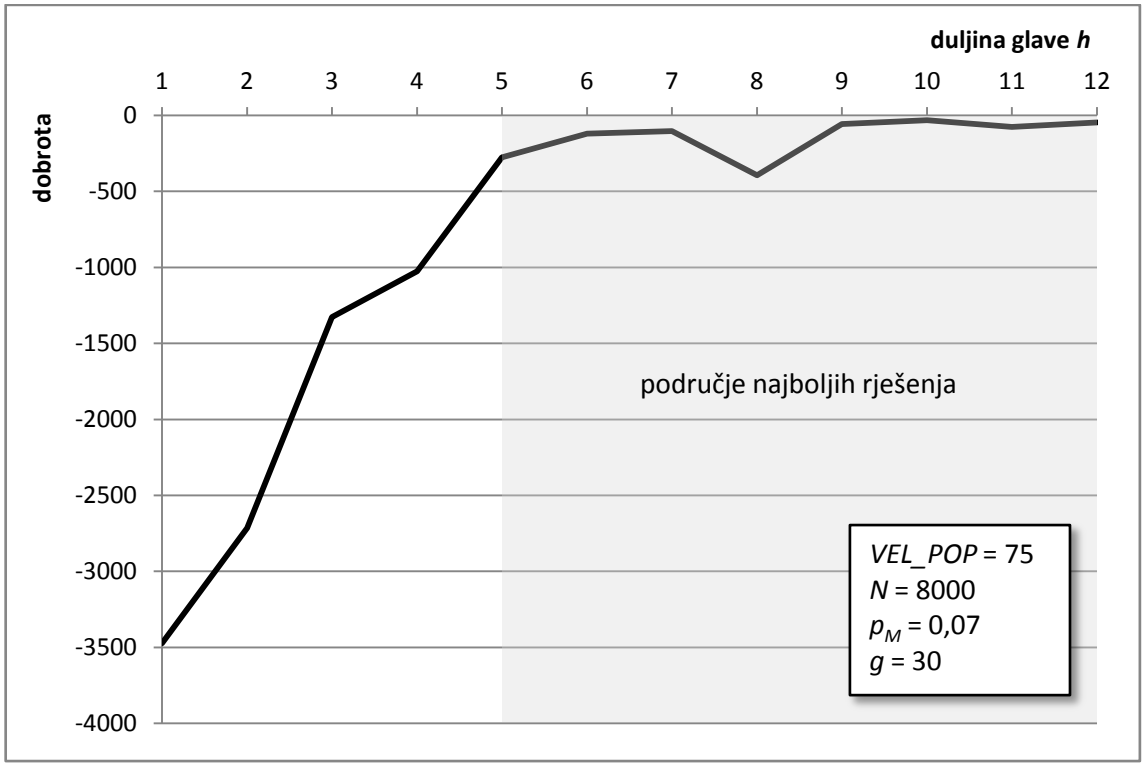
```
(+ (+ X (COS (+ X (- (COS (EXP X)) (* (TAN X) X))))) (+ (- (TAN (SIN X)) (+
(* X X) (/ X X))) (COS (- (SIN X) X))) (+ (SIN (* X (SQRT (SQRT (- (* X X)
X))))) X) (COS (+ (SIN (/ X (EXP (TAN (- (* X X) X))))) X)) (+ X (COS (+ X
(- (COS (EXP X)) (* (TAN X) X))))) (TAN (TAN (- (* (SIN (EXP X)) (SIN (SQRT
X))) (TAN (EXP X))))) (SIN (* (- X (SQRT (- (COS X) X))) (SIN (COS (/ X
X))))) (- (SIN (* (* (SIN X) X) (SIN X))) (EXP (/ (TAN X) X))) (SQRT (SQRT
(- (+ (- (- X X) (TAN X)) (EXP X)) (TAN X)))) (EXP (- (SQRT X) (EXP (SQRT
(+ (- (- X X) X) X))))) (+ X (SIN (+ (SIN (+ (TAN X) X)) (- (* X X) (EXP
X))))) (TAN (- (COS (+ (TAN X) (TAN X))) (+ (/ (/ X X) X) (/ X X)))) (+
(SIN (EXP X)) X) (+ X X) X (* (* (COS X) (SIN (/ X X))) (+ (- (SQRT X) X)
(COS X))) X (/ (TAN (EXP (/ (+ X X) (SIN X)))) (COS (+ (* X X) (SIN X)))) X
X X (- (TAN (* (SQRT (EXP X)) (SIN (/ X X)))) (TAN (SQRT (TAN X)))) X (SQRT
(SIN (* (- (- (+ X X) (* X X)) (/ X X)) (COS X)))) (SQRT (/ (COS (/ (TAN X)
(* X X))) (+ (COS (+ X X)) (SIN X)))) (SIN (TAN (EXP (SQRT (/ (* X (TAN X))
(- (SIN X) X)))))) (+ X (SIN (+ (/ (+ (SQRT X) X) (EXP X)) (TAN (+ X X))))
(+ X (SIN (SQRT (EXP (SQRT X))))) (+ X (SIN (SQRT (EXP (SQRT X))))) (TAN
(EXP (* (SIN (/ (EXP X) X)) (COS (COS (COS X)))))))
```

Nakon analize referentnog rezultata optimiranja, u sljedećim poglavljima dokumentirane su analize osjetljivosti na promjene nekih ulaznih parametara.

6.1.2. Osjetljivost kvalitete rješenja o duljini glave gena

Duljina glave gena prvi je od dva parametra koji određuju ukupnu duljinu kromosoma. Veća duljina glave omogućuje veću dubinu stabla izraza. Duboka stabla izraza daju složenije članove povezane odabranom povezujućom funkcijom. U ovoj analizi, duljina glave kreće se u rasponu od jednog do 12 elemenata. Svi su ostali parametri isti kao i u prethodnom poglavlju.

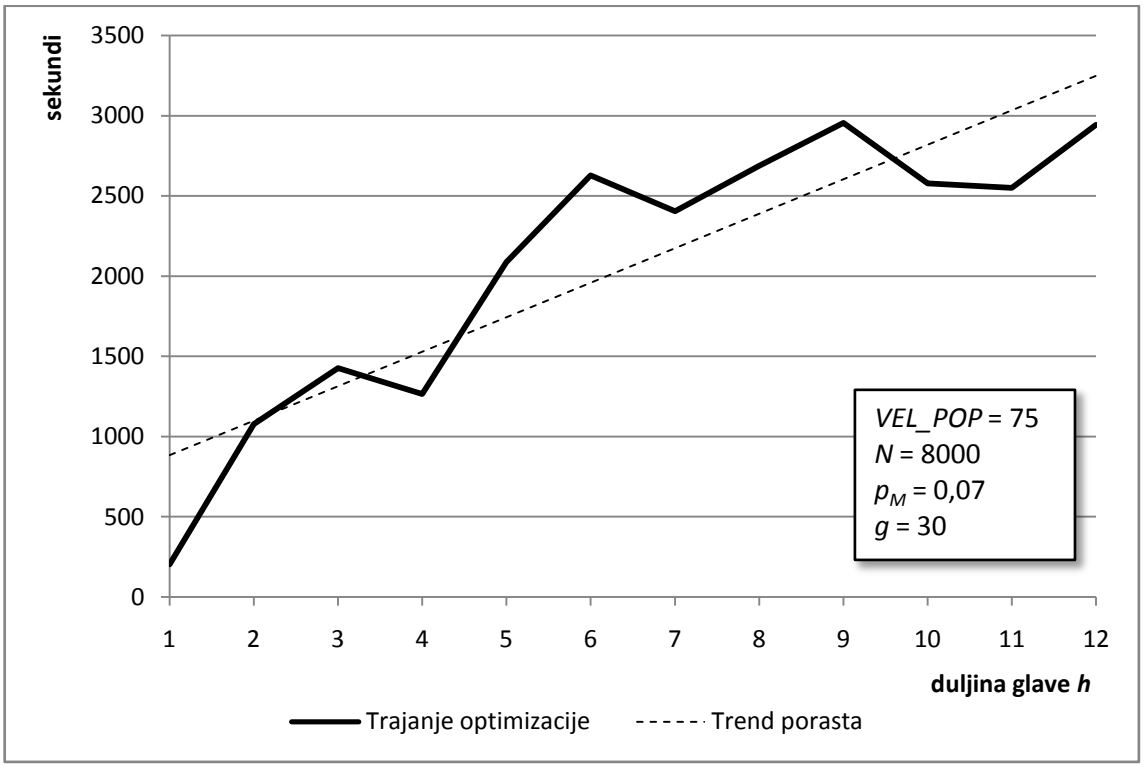
Porastom duljine glave konačno rješenje postaje sve bolje i bolje (Slika 6.3). Kada duljina glave dosegne šest elemenata, konačno rješenje postaje blisko optimumu pa dodatno produljenje glave primjetno ne popravlja rješenje. Kako se metoda optimiranja zasniva na slučajnim procesima, veća duljina glave ne osigurava nužno i bolje konačno rješenje. Na primjer, za glavu dugu 8 elemenata, konačno rješenje je nekoliko puta gore i od onog s kraćom glavom i od onog s duljom glavom. Stoga je nužno nekoliko puta ponoviti optimizaciju i od dobivenih rješenja odabrati najbolje, radi izbjegavanja ovakvih situacija.



Slika 6.3. Najbolja dobivena rješenja za različite duljine glave gena

Drugi graf prikazuje trajanje procesa optimizacije u ovisnosti o duljini glave (Slika 6.4). Uz stvarne podatke, ubačena je funkcija trenda porasta trajanja. Pokazalo se da je trajanje optimizacije linearno proporcionalno s duljinom glave. Odstupanja stvarnih vrijednosti od

linearne funkcije najvjerojatnije su uzrokovana složenošću cijele populacije. Ukoliko prevladavaju jednostavnije jedinke, evaluacija se brže izvodi pa je i cijeli algoritam brži.



Slika 6.4. Trajanje optimizacijskog procesa za različite duljine glave i trend porasta

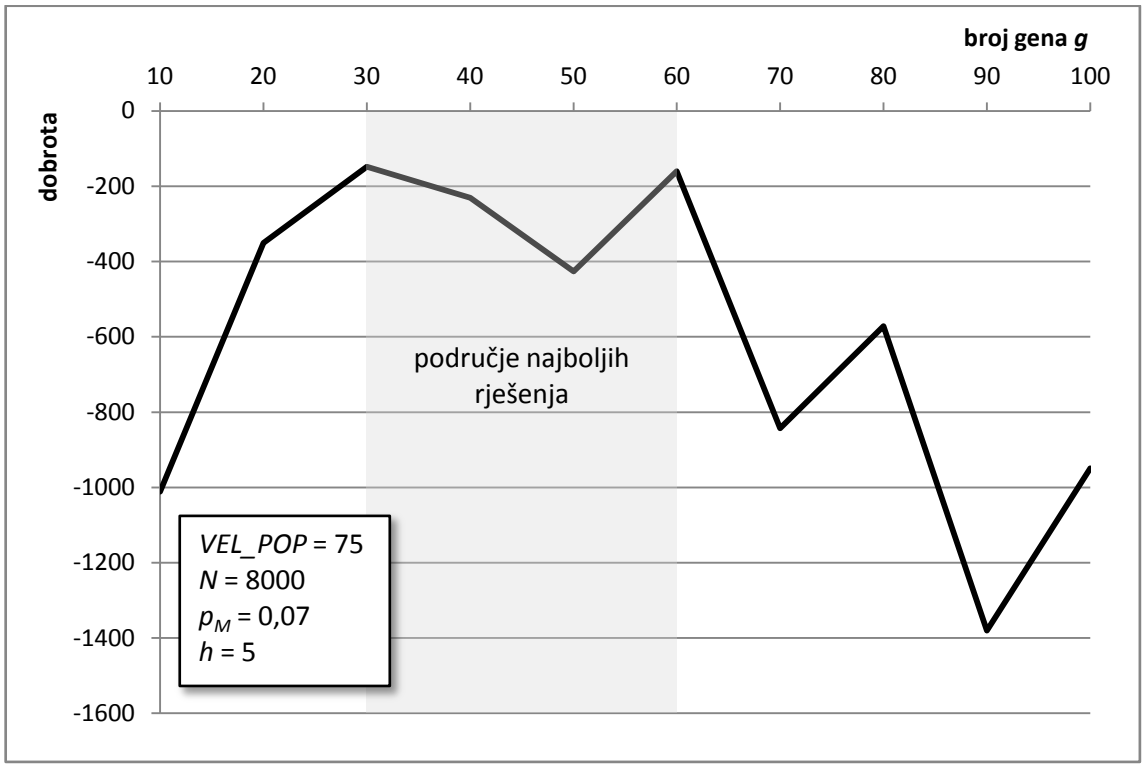
6.1.3. Osjetljivost kvalitete rješenja o broju gena

Drugi parametar koji određuje duljinu kromosoma jest broj njegovih gena. Broj gena određuje širinu konačnog stabla izraza cijelog kromosoma. Ovo se svojstvo može iskoristiti za traženje aproksimacijske funkcije građene od mnogo jednostavnih članova (uz korištenje gena s kraćom glavom), što je također jedan mogući smjer traženja globalnog optimuma. U ovoj analizi broj gena u kromosomu rastao je od 10 do 100 gena, u skokovima od po 10 gena. Ostali parametri su isti kao i u poglavlju 6.1.1.

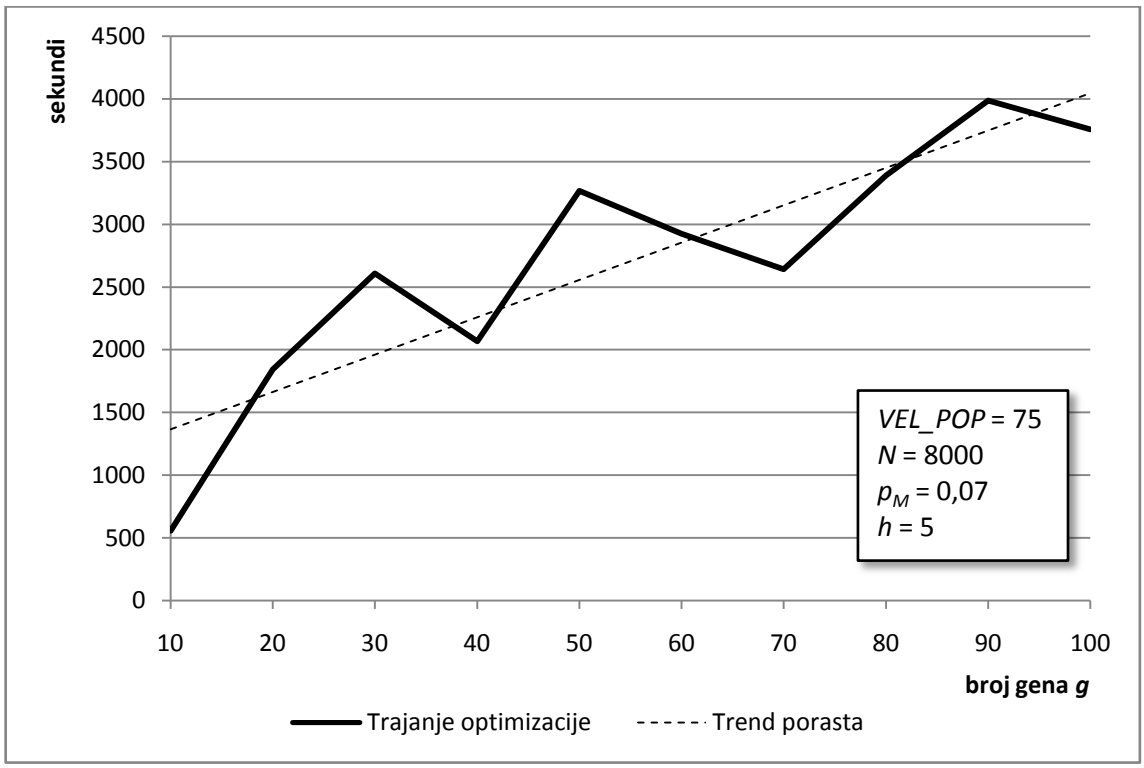
Slika 6.5. prikazuje graf kvalitete najbolje pronađene jedinke. Najbolji rezultati se postižu za broj gena od 30 do 60. Kao i u prethodnom poglavlju, algoritam zbog svoje stohastičke prirode nije došao do boljeg rješenja za broj gena 40 i 50, što ne znači da ne može.

Za kromosome s brojem gena manjim od 30 nema puno pomoći, jer ne daju dovoljno članova u konačnoj funkciji, a rješenje može biti produljenje glave gena. Slično, porastom broja gena preko 60 drastično opada kvaliteta najbolje jedinke. To znači da je kromosom presložen za efikasno rješavanje u okviru ulaznih parametara, tj. vjerojatno je dovoljno samo povećati broj iteracija i algoritam će doći bliže optimumu.

Kao i u prošlom poglavlju, trajanje izvršavanja algoritma je proporcionalno broju gena u kromosomu, a oscilacije oko linearne funkcije javljaju se iz istih razloga (Slika 6.6).



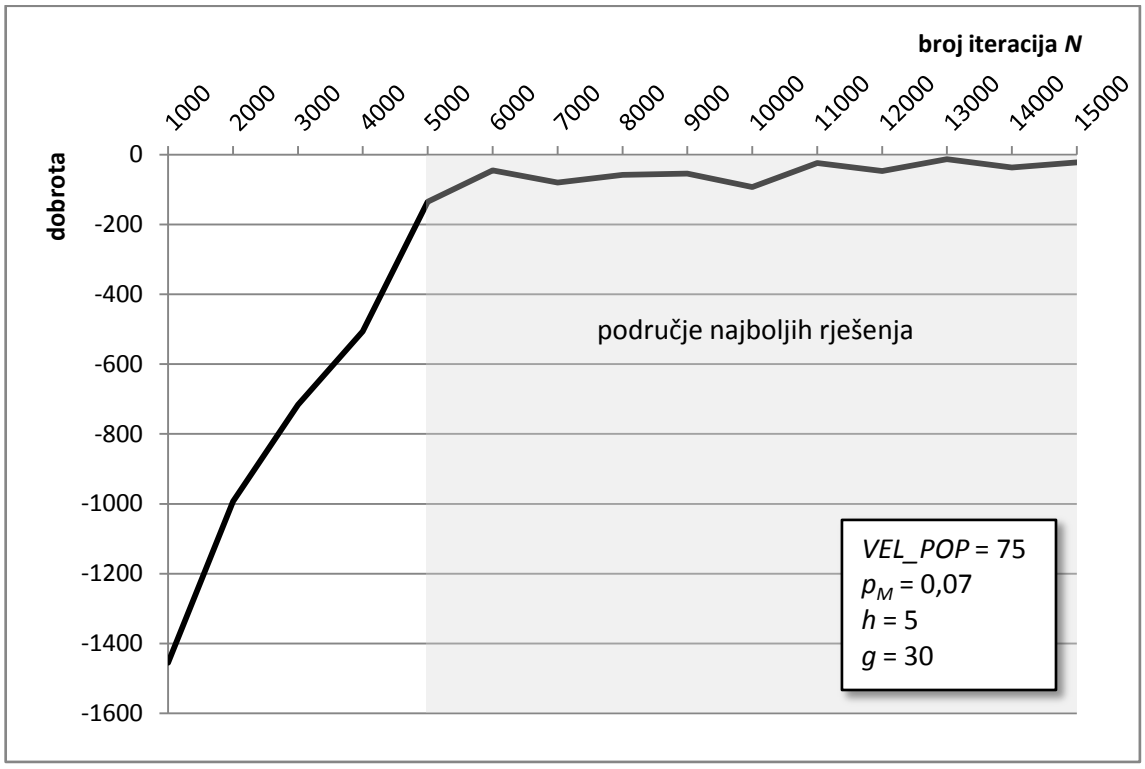
Slika 6.5. Najbolja dobivena rješenja za različit broj gena u kromosomu



Slika 6.6. Trajanje optimizacijskog procesa za različit broj gena i trend porasta

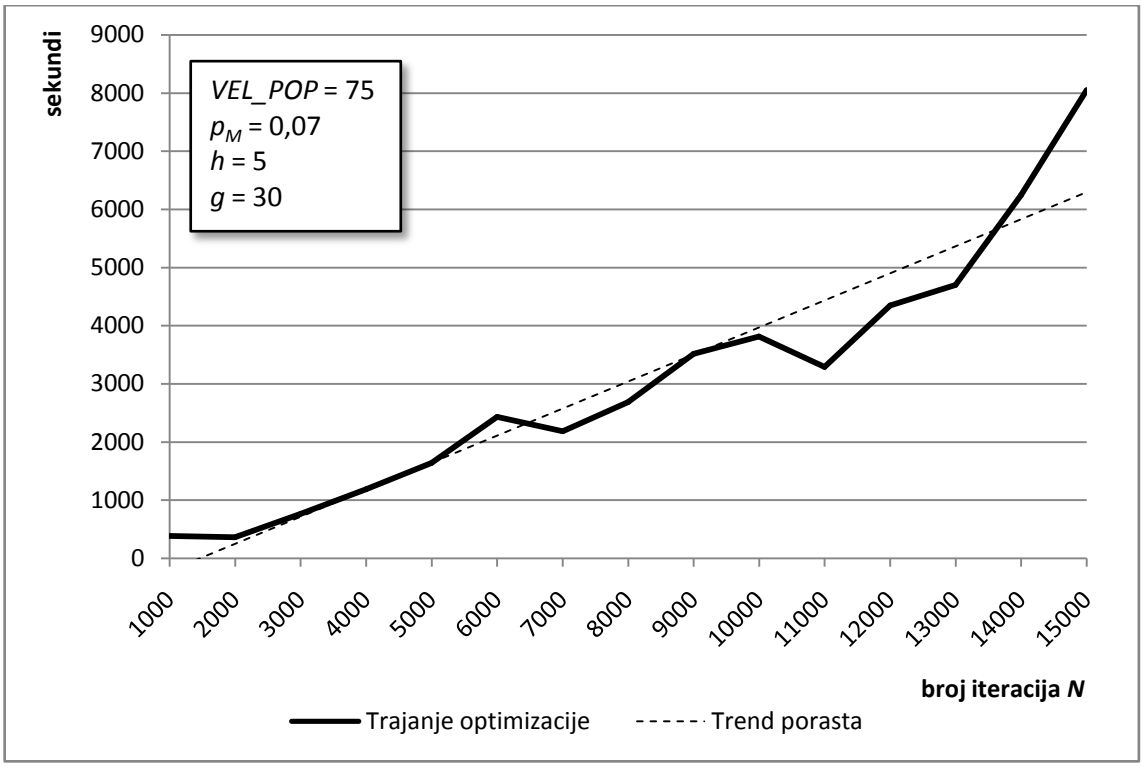
6.1.4. Osjetljivost kvalitete rješenja o broju iteracija

Treća analiza ukazuje na popravljanje kvalitete rješenja porastom broja iteracija. Broj iteracija rastao je od jedne do 15 tisuća, u skokovima od po tisuću iteracija. Prilikom izvođenja ove analize, nije se odradio samo jedan optimizacijski proces od 15 tisuća iteracija te usput bilježilo stanje u populaciji, već je program pokrenut 15 puta svaki put s uvećanim brojem iteracija. Slika 6.7, zbog već opisanih svojstava algoritma, ne prikazuje idealno poboljšavanje rješenja porastom broja iteracija. Ipak, vidi se da je za dostizanje kvalitetnog rješenja potrebno oko 5000 iteracija. Ostali parametri se ne mijenjaju i istovjetni su onima iz poglavlja 6.1.1.



Slika 6.7. Najbolja dobivena rješenja za različit broj iteracija algoritma

Kao i u prethodnim poglavljima, druga slika (Slika 6.8), prikazuje porast trajanja izvođenja optimizacijskog algoritma. I u ovom slučaju je očita linearna ovisnost o mijenjanom parametru. Oscilacije oko linearne funkcije naročito su došle do izražaja kod velikog broja iteracija.

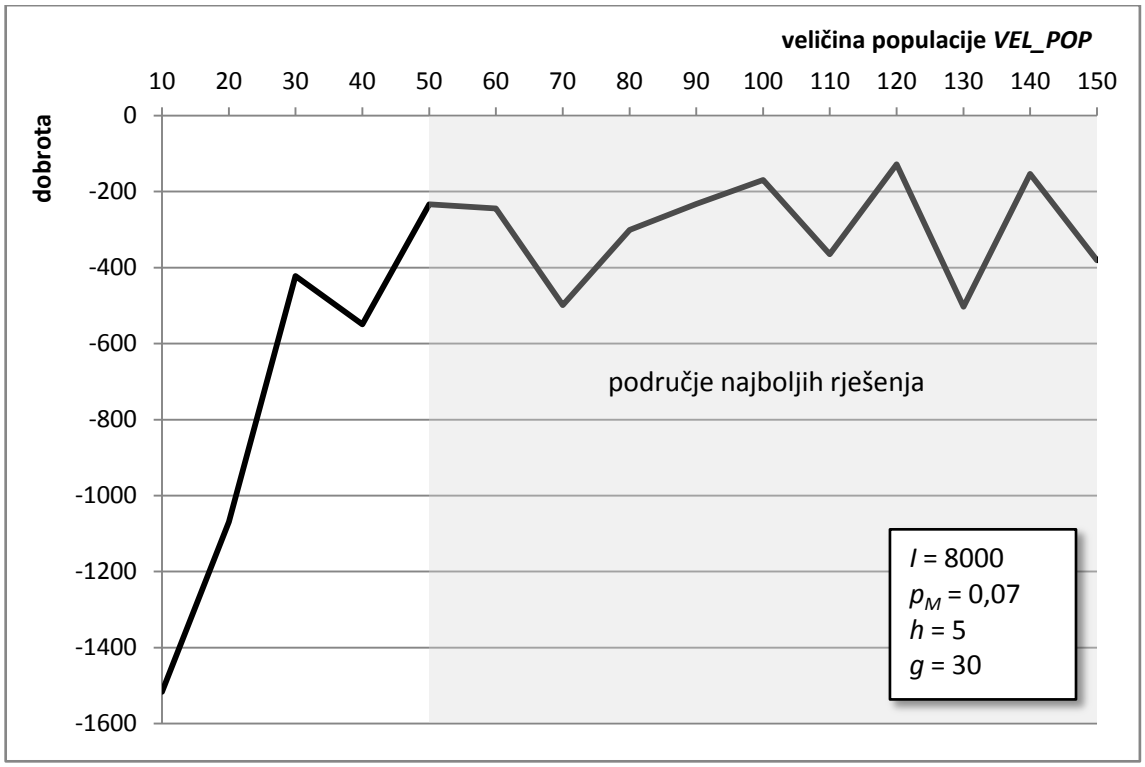


Slika 6.8. Trajanje optimizacijskog procesa za različit broj iteracija i trend porasta

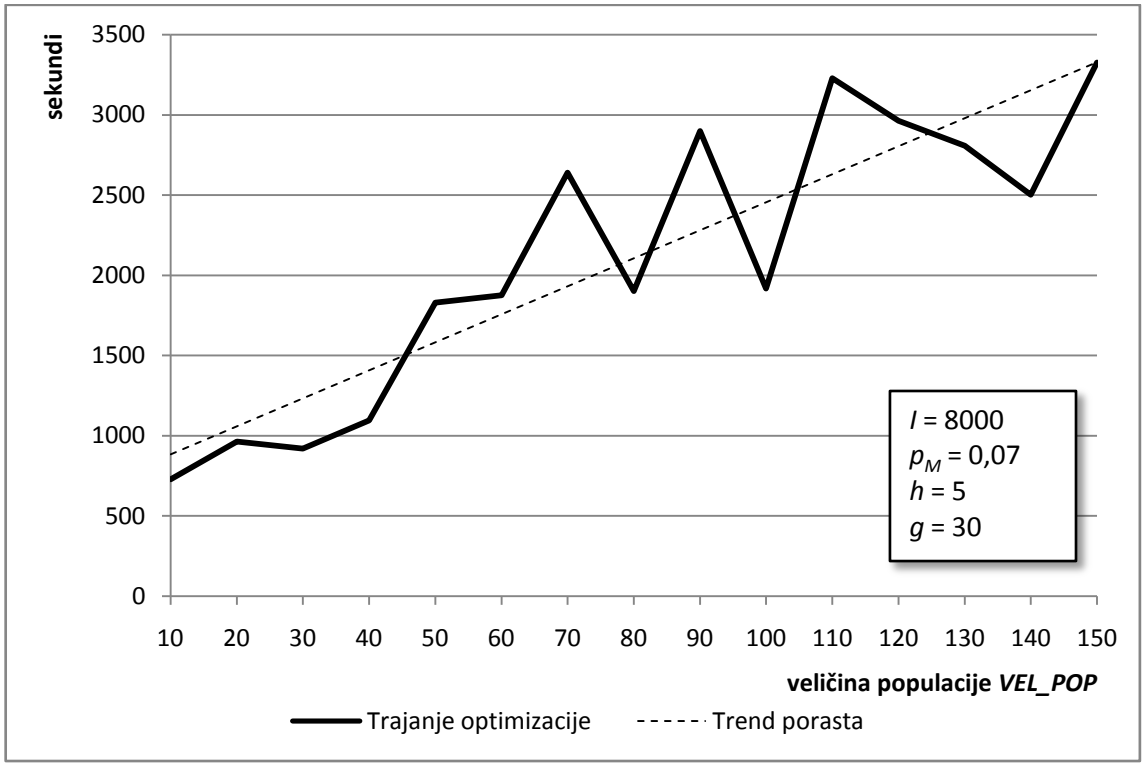
6.1.5. Osjetljivost kvalitete rješenja o veličini populacije

Sljedeća analiza demonstrira ovisnost kvalitete konačnog rješenja o veličini populacije. Slika 6.9. daje prikaz tog odnosa u obliku grafa. Kao što je i očekivano, za manje populacije (do 50 pripadnika), algoritam ne dolazi do najboljeg mogućeg rješenja. Razlog tome je njena nedovoljna genetska raznolikost, zbog čega populacija zastrani i zapne u nekom od lokalnih optimuma. Povećanjem populacije osigurava se dovoljna raznolikost pa algoritam dolazi do dobrog rješenja. Daljnjim porastom populacije (preko 150 članova) algoritam bi postupno dolazio do sve gorih rješenja, jer jako velike populacije značajno usporavaju konvergenciju ka optimumu.

Na prvi pogled, neočekivana je linearna ovisnost brzine rada algoritma o veličini populacije (Slika 6.10). Razlog ovome je stvar implementacije populacije u programskom jeziku. Naime, populacija se sprema u obliku vezane liste zbog čega je gornja granica trajanja pristupa nekom njenom elementu određena s $O(n)$. Ovaj problem se rješava korištenjem običnog niza ili tablice raspršivanja, gdje je trajanje pristupa elementu određeno kao $O(1)$.



Slika 6.9. Najbolja dobivena rješenja za različite veličine populacije

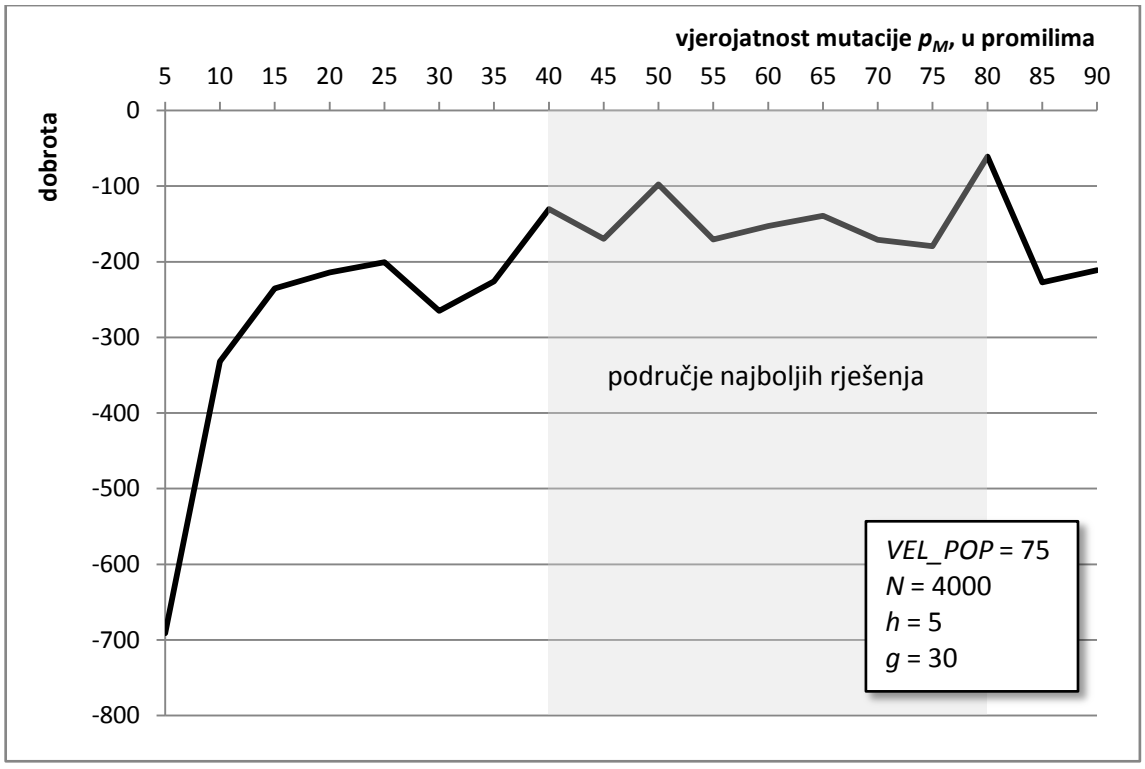


Slika 6.10. Trajanje optimizacijskog procesa za različite veličine populacije i trend porasta

6.1.6. Osjetljivost kvalitete rješenja o vjerojatnosti mutacije

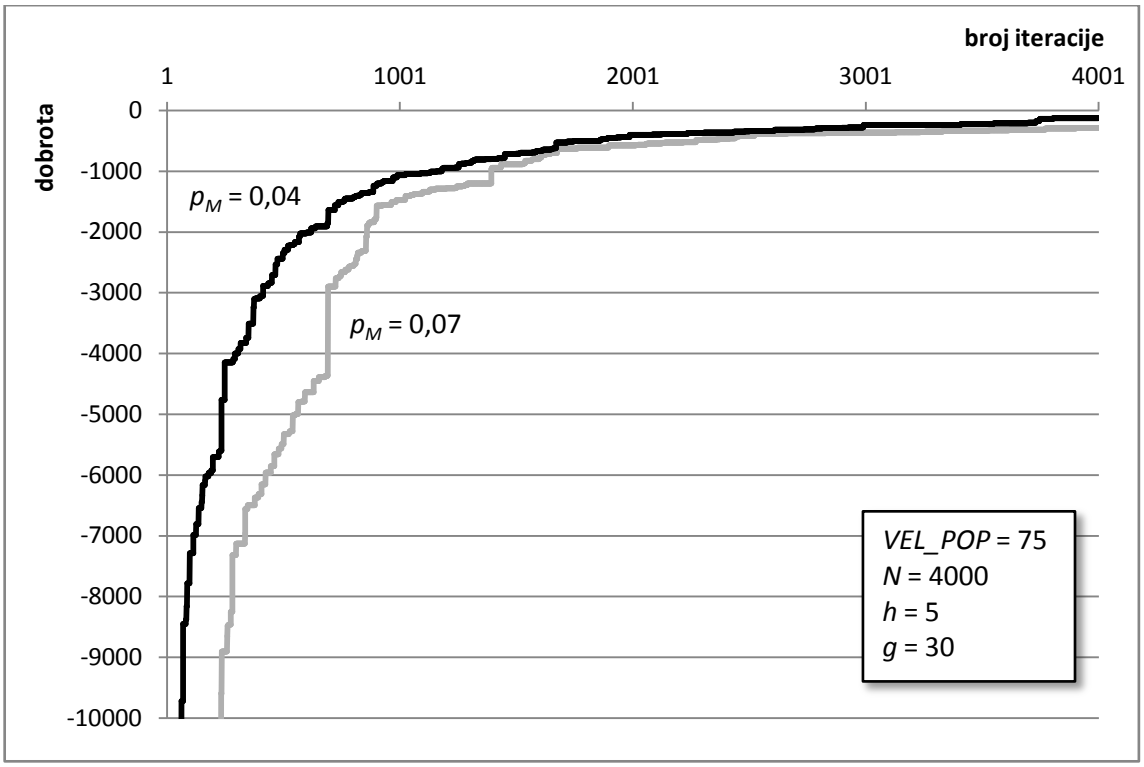
Operator mutacije tjera evolucijski proces naprijed zbog mogućnosti izbacivanja populacije iz lokalnog optimuma. Ovo je najvažniji parametar evolucijskog programa i kvaliteta konačnog rješenja je najosjetljivija na njegovu vrijednost. Stoga je zanimljivo vidjeti kako učestalost javljanja mutacije utječe na kvalitetu konačnog rješenja (Slika 6.11). Učestalost javljanja je definirana preko parametra vjerojatnosti mutacije. U analizama se vjerojatnost mutacije povećavala od 5 do 90 ‰, a broj iteracija smanjen je na 4000 kako bi se svako optimiranje moglo ponoviti bar deset puta.

Očekivano, za najmanje vrijednosti parametra, algoritam se ne približava optimumu i ostaje daleko od optimuma. Porastom vjerojatnosti mutacije i najbolje rješenje se popravilo. U rasponu od četiri do osam posto, postižu se najbolji rezultati optimiranja. Daljnjim porastom vrijednosti, dolazi do naznake ponovnog rušenja dobrote rješenja jer se sve više jedinki mutira bez mogućnosti dugotrajnijeg istraživanja područja u kojem su se mutacijom našle.



Slika 6.11. Najbolja dobivena rješenja za različite vjerojatnosti mutacije

Slika 6.12. prikazuje napredovanje najbolje jedinke za dva iznosa vjerojatnosti mutacije, za 4 % i za 7 %. Manji iznos vjerojatnosti je osigurao gotovo idealnu krivulju napretka najbolje jedinke. U početnim iteracijama se jedinka iznimno brzo poboljšava, zatim se taj trend postupno smanjuje i nakon „koljena“ praktički više nema značajnog napretka. Ovakav izgled grafa označava temeljitiju pretragu prostora rješenja i sigurnije kretanje prema globalnom optimumu. U idealnom slučaju, ovaj bi se graf poklapao s grafom eksponencijalne funkcije.



Slika 6.12. Usporedba napredovanja najbolje jedinke za vjerojatnosti mutacije 4 % i 7 %

Veći iznos vjerojatnosti mutacije (7 %, iz poglavlja 6.1.1) pokvario je izgled grafa. On sad ima dosta iznenadnih skokova i manje podsjeća na eksponencijalnu funkciju. Takav izgled naznaka je prevelike vjerojatnosti mutacije i zapravo sreće da je algoritam našao put prema boljem rješenju. Na kraju optimiranja, manja vjerojatnost mutacije dovela je algoritam do boljeg rješenja (-130,567 nasuprot -292,5 jedinica dobrote).

Iz ovih saznanja zaključuje se da manja vjerojatnost mutacije osigurava sporiji, ali stabilniji put prema globalnom optimumu. Njezin veći iznos daje nepredvidljivije kretanje najbolje jedinke i u konačnici možda gori rezultat.

6.2. Aproksimiranje većeg skupa točaka

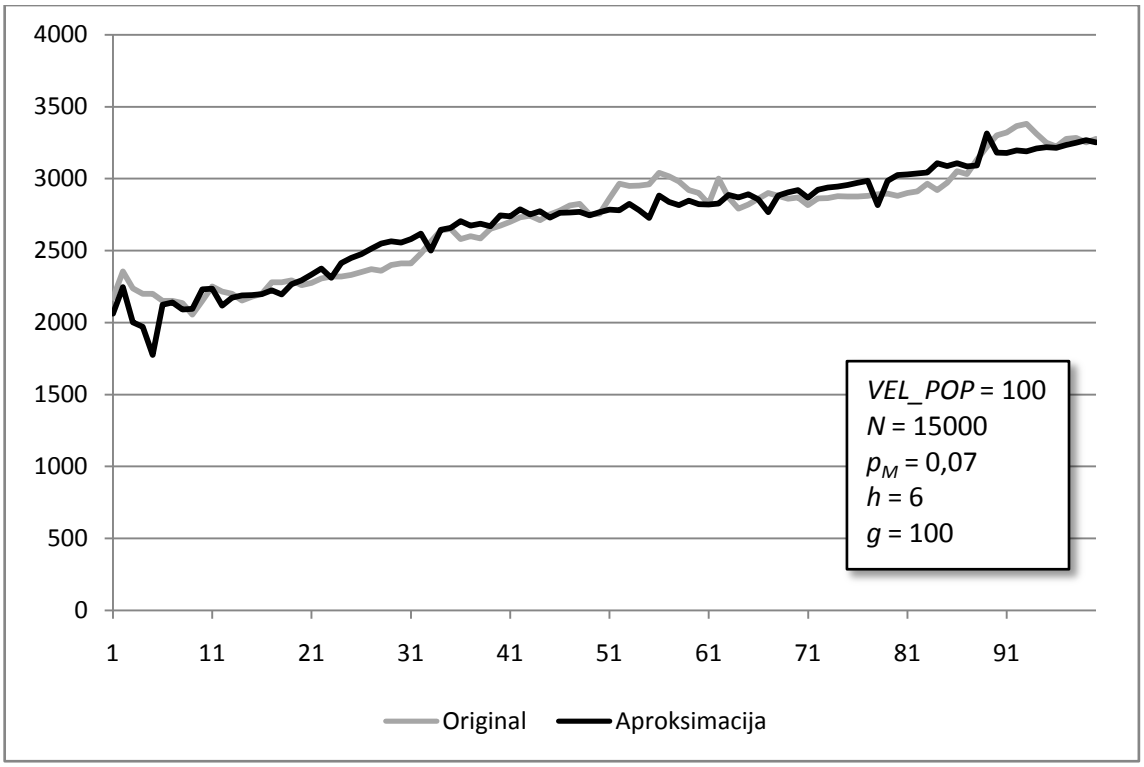
Skup od 20 točaka koji se dosad koristio, razmjerno je malen ulazni skup i sposobnost algoritma da riješi problem se demonstrira nad većim skupovima. Traženje aproksimacija većih skupova je čest slučaj u praksi, pogotovo u sustavima za predviđanje kretanja vrijednosti promatrane veličine u budućnosti. Najčešće je riječ o kretanju cijene raznih energenata ili dionica neke tvrtke.

Stoga je napravljen skup od 100 vrijednosti, ali ne slučajno odabranih, već iz stvarnog života. Riječ je o vrijednostima dionice na Zagrebačkoj burzi. Konkretno, uzete su zadnje dnevne vrijednosti dionica INA-e (simbol „INA-R-A“), iz razdoblja od 1. prosinca 2006. do 26. travnja 2007. Podaci su preuzeti sa službenih stranica Zagrebačke burze [14].

Tablica 6.4. Ulazni parametri algoritma za veći skup točaka

Parametar	Oznaka	Vrijednost
Broj iteracija	*N*	15000
Duljina glave gena	*h*	6
Broj gena	*g*	100
Povezujuća funkcija		+
Veličina populacije	*VEL_POP*	100
Vjerojatnost:		
- **mutacije kromosoma**	p_M	0,07
- **premještanja niza za umetanje**		0,1
- **premještanja niza za umetanje korijena**		0,1
- **premještanja gena**		0,1
- **križanja s jednom točkom prekida**		0,4
- **križanja s dvije točke prekida**		0,5
- **rekombinacije gena**		0,3

Tablica 6.4. popisuje ulazne parametre različite od onih u poglavlju 6.1.1. Testno računalo isto je kao i u navedenom poglavlju.



Slika 6.13. Poklapanje originalne funkcije i njezine aproksimacije za veći skup točaka

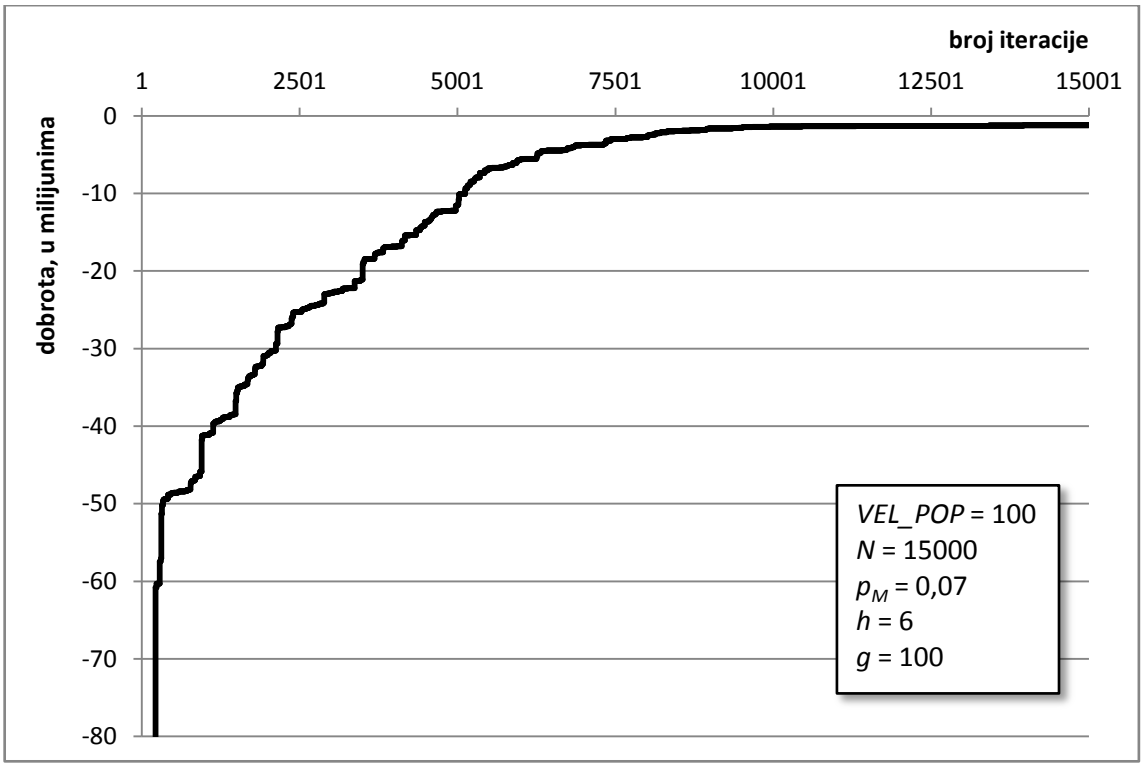
Optimizacijski proces trajao je oko 8 sati i 15 minuta. Slika 6.13. prikazuje odnos originalne funkcije i najbolje pronađene aproksimacije. Kako funkcija originala nije poznata, a funkcija aproksimacije iznimno složena (Tablica 6.5), obje su prikazane u svom jednostavnom obliku – pomoću linearnih odsječaka za vrijednosti između svakog $g(x_i)$.

Zanimljivo je uočiti kako algoritam rješava problem nedostatka konstanti u S-izrazima. Pažljiviji pregled izraza u tablici pronalazi mnoge „konstante“ izražene pomoću funkcija. Najčešći primjer je *x* podijeljen sa samim sobom, jedinica koja je argument nekoj drugoj funkciji, npr. `(TAN (/ X X))`. Ukoliko je potreban pomak, tj. izrazu se pribraja neka konstanta, onda se ona dobije iz ovakvog izraza: `(/ (+ X X) X)`, koji predstavlja vrijednost broja dva. Trenutno postoje istraživanja u kojima se programiranje genskih izraza dopunjava korištenjem pravih konstanti. Njihovo će uvođenje značajno popraviti konačna rješenja algoritma.

Kao i kod svake metode optimiranja, najbolja jedinka je jako brzo napredovala kroz prve generacije, da bi nakon prolaska „koljena“ napredovanje stagniralo u blizini globalnog optimuma ili rješenja dosežnog uz zadane ulazne parametre (Slika 6.14). Dobrota najbolje jedinke iznosila je oko -1,2 milijuna jedinica dobrote. Kako je vidljivo iz grafa, algoritam je došao do najboljeg rješenja do kojeg je mogao doći. Daljnje optimiranje ne bi puno popravilo najbolju jedinku. Bolji put je redefiniranje ulaznih parametara i ponovno pokretanje programa.

Tablica 6.5. S-izraz najbolje aproksimacije većeg skupa točaka

```
(+ (* (/ X (* X X)) (EXP (SQRT X))) (+ X X) (SQRT (EXP (SQRT X))) (* X
(SQRT (* (/ X X) X))) (/ (TAN (* (EXP X) X)) (COS (+ X X))) (* (SIN (SQRT
X)) (- (SQRT X) (+ X X))) (/ (* (* X X) (* X X)) (SQRT (EXP X))) (/ (* (* X
X) (* X X)) (SQRT (EXP X))) X X (+ (COS (EXP X)) (- (+ X X) (SQRT X))) X
(EXP (/ (SQRT X) (/ (+ X X) X))) (* (SQRT (SQRT (+ (/ X X) X))) X) (/ (* (*
X X) (* X X)) (SQRT (EXP X))) X (SQRT (EXP (* X (/ (SQRT X) X)))) (SQRT
(EXP (* X (/ (SQRT X) X)))) (* (SIN (SQRT X)) (- (SQRT X) (+ X X))) (EXP (+
(EXP (/ X X)) (EXP (/ X X)))) (EXP (/ (SQRT X) (/ (+ X X) X))) (EXP (/
(SQRT X) (/ (+ X X) X))) (EXP (/ (SQRT X) (/ (+ X X) X))) (SQRT (EXP (* X
(/ (SQRT X) X)))) (* (TAN (TAN (/ X X))) (EXP (/ X X))) (/ (* (* X X) (* X
X)) (SQRT (EXP X))) (/ (* (* X X) (* X X)) (SQRT (EXP X))) (/ (* (* X X) (*
X X)) (SQRT (EXP X))) (EXP (+ (EXP (/ X X)) (EXP (/ X X)))) (EXP (TAN (+ (/
(/ X X) X) (SIN X)))) (EXP (TAN (+ (/ (/ X X) X) (SIN X)))) X (* (COS (SQRT
X)) (- (TAN X) (+ X X))) (EXP (/ (SQRT X) (/ (+ X X) X))) (EXP (/ (SQRT X)
(/ (+ X X) X))) (EXP (/ (SQRT X) (/ (+ X X) X))) (EXP (/ (SQRT X) (/ (+ X
X) X))) (EXP (/ (SQRT X) (/ (+ X X) X))) (EXP (/ (SQRT X) (/ (+ X X) X)))
(EXP (/ (SQRT X) (/ (+ X X) X))) (EXP (/ (SQRT X) (/ (+ X X) X))) (EXP (/
(TAN (SQRT (+ X X))) X)) (EXP (+ (EXP (/ X X)) (EXP (/ X X)))) X (+ (* (/ X
X) X) X) (TAN (TAN (/ (SQRT X) (/ X X)))) X X (+ (COS (* X (+ X X))) X) X
(* (- X (/ X X)) (SQRT X)) (* (/ (* X X) (* X X)) (EXP (/ X X))) (SQRT (EXP
(SQRT X))) (SQRT (EXP (SQRT X))) (+ (TAN (TAN (/ X X))) (EXP (/ X X))) (EXP
(TAN (EXP (TAN (/ (/ X X) X))))) X X X (/ (TAN (SQRT (/ X X))) (TAN (SQRT
X))) (SQRT (EXP (SQRT X))) (SQRT (EXP (SQRT X))) X (TAN (+ (SQRT (/ (SQRT
X) X)) X)) X X (- (COS (EXP X)) (+ (SQRT X) (* X X))) (TAN (* (TAN (* X X))
(- (COS X) X))) (+ (/ (* X X) (SQRT X)) (- (+ X X) X)) (+ (COS (SQRT (COS
(+ X X)))) X) (+ (+ (TAN X) (/ X X)) X) X (SIN (SQRT X)) X (- X (- (COS X)
(/ X X))) X X (EXP (+ (COS (- X X)) (TAN (/ X X)))) X (+ X X) (SQRT (EXP (*
X (/ (SQRT X) X)))) (SQRT (EXP (* X (/ (SQRT X) X)))) (SQRT (EXP (* X (/
(SQRT X) X)))) (TAN (SQRT (SQRT (- X (- X X))))) (EXP (SQRT (TAN (+ (+ X X)
(* X X))))) (TAN (TAN (COS (SIN (- X X))))) (TAN (TAN (COS (SIN (- X X)))))
X (TAN (* (TAN X) X)) X (+ (EXP (+ (/ X X) (/ X X))) X) X (/ X (/ (/ (+ X
X) X) (SQRT X))) (- X (- (/ (TAN X) X) (/ X X))) (* (TAN (COS (- X X)))
(TAN (TAN X))) X X X X X)
```



Slika 6.14. Napredovanje najbolje jedinke kroz iteracije algoritma

Statistički gledano, rješenje s ovako velikom sumom kvadrata odstupanja ipak nije toliko loše kakvo se čini na prvi pogled. Računanjem srednjeg odstupanja pa njegovog relativnog iznosa, dobije se odstupanje od približno 4 % u odnosu na ulazni skup, a to u mnogim slučajevima može biti dovoljno dobro (Tablica 6.6).

Tablica 6.6. Kvaliteta najbolje aproksimacije većeg skupa točaka

Podatak	Vrijednost
suma kvadrata odstupanja	1 200 000
broj uzoraka	100
srednje kvadratno odstupanje	12000
srednje odstupanje	±110
prosječna vrijednost uzorka	2711,85
relativno srednje odstupanje	≈ ±4 %

6.3. Usporedba programiranja genskih izraza i genetskog algoritma

Dosadašnje analize orijentirale su se na kvalitetu rada i brzinu rada algoritma programiranja genskih izraza. U ovom poglavlju uspoređeno je s radom jednostavnog genetskog algoritma, jedne od najjednostavnijih i najbržih metoda optimiranja. Jednostavni genetski algoritam, zbog konciznog zapisa jedinke i malenog izvršnog programa, jako brzo prolazi kroz iteracije i konvergira ka globalnom optimumu.

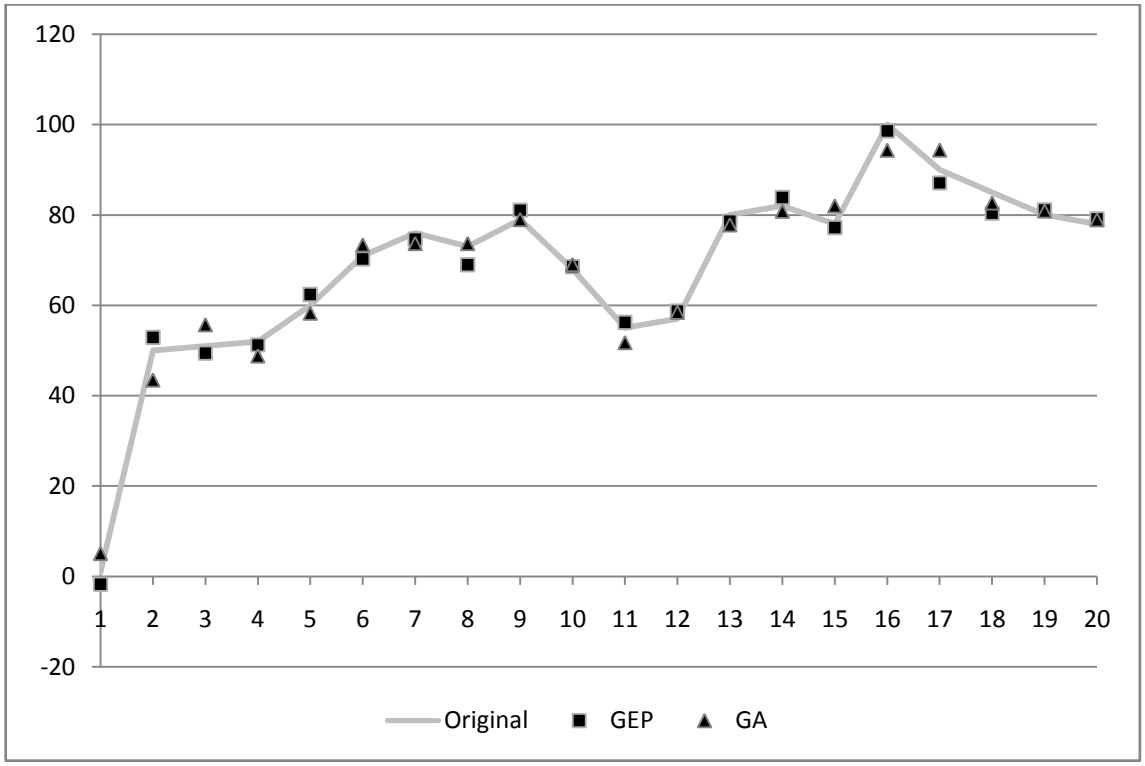
Kako bi se oba algoritma mogla testirati u jednakim uvjetima, pokrenuto je virtualno računalo u VMware okruženju. Kao operacijski sustav instalirana je Linux distribucija Ubuntu, verzija 7.1. Linux je odabran radi izbjegavanja mogućih problema s generatorom pseudo-slučajnih brojeva na Windows platformi.

Ulazni skup je isti kao u poglavlju 6.1. Od parametara programiranja genskih izraza, jedina promjena u odnosu na one iz poglavlja 6.1.1. je povećan broj iteracija na 10 tisuća. Tablica 6.7. popisuje parametre jednostavnog genetskog algoritma.

Tablica 6.7. Parametri jednostavnog genetskog algoritma

Parametar	Vrijednost
Broj iteracija	100 000
Broj bitova reprezentacije	26
Veličina populacije	70
Vjerojatnost mutacije	0,01

Rad u istim uvjetima je pokazao da je korištena implementacija genetskog algoritma drastično brža od implementacije programiranja genskih izraza. Dijeljenjem prosječnog trajanja iteracije za oba algoritma, dobije se omjer 1:3550 u korist genetskog algoritma (Tablica 6.8).



Slika 6.15. Grafički prikaz konačnih rješenja dviju metodologija

Jedan od važnijih razloga tolike razlike u brzini je izbor programskog jezika. Program genetskog algoritma je napisan u programskom jeziku C++, koristi nizove cijelih brojeva te

obavlja osnovne logičke i aritmetičke operacije nad njima. Sve su te operacije izravno podržane u hardveru računala i zbog toga se najbrže izvode. Common Lisp (tj. njegova implementacija CLISP), prvo prevodi naredbe jezika u svoj vlastiti međukod, a on se dalje prevodi u odgovarajući strojni kod tek kad se pokrene program na nekom računalu. Kako se interpretiranje međukoda obavlja za svaku naredbu, njihovo izvršavanje je dosta sporije od onih iz jezika C. Također, intenzivno korištenje listi, a ne nizova, dodatno usporava implementaciju programiranja genskih izraza.

Tablica 6.8. Usporedba konačnih rješenja dvije metodologije

Podatak	Programiranje genskih izraza	Genetski algoritam
Najbolja jedinka	-98,423	-194,984
Trajanje optimizacije	3125,3 s	8,827 s
Broj iteracija	10 000	100 000
Prosječno trajanje iteracije	0,31253 s	0,000088 s

Rješenje dobiveno radom genetskog algoritma ima dvostruko veću pogrešku od najbolje jedinke drugog algoritma, makar je odradio 10 puta više iteracija. Rješenje se može popraviti dodavanjem još sinusnih članova u aproksimacijsku funkciju prema izrazu (5.1). Slika 6.15. prikazuje iznose obiju konačnih funkcija za svaki x_i iz ulaznog niza. Radi preglednosti grafa, konačne funkcije su prikazane u točkastom obliku.

7. Zaključak

Odabir evolucijskog procesa kao osnovu nekog optimizacijskog algoritma, pokazao se kao smjela, ali dobra odluka. Oba testirana algoritma su pokazala sposobnost dostizanja jako dobrog rješenja u relativno kratkom roku. Programiranje genskih izraza, kao tema ovog diplomskog rada, za nekoliko je redova veličine sporija metoda optimiranja od genetskog algoritma, ali zato dostiže bolje konačno rješenje i ima širu primjenjivost.

Promjenom ulaznih parametara programiranja genskih izraza može se značajno popraviti konačno rješenje, ali redovito uz sporiji rad algoritma. Stoga je važan čimbenik raspoloživo vrijeme i koliko se dobra aproksimacija uopće želi postići. Uz te dvije informacije puno je lakše postaviti odgovarajući skup ulaznih parametara.

Nedostatak korištene implementacije programiranja genskih izraza je nepodržavanje konstanti u izrazima, zbog čega je značajno otežano fino namještanje rješenja. Algoritam je sam u nekim slučajevima generirao izraze koji su konstantni za svaki *x*, ali to nije pravo rješenje problema. Stoga bi budući rad stavio naglasak na popravljanje kvalitete rješenja i to na dva načina: uvođenjem konstanti u matematičke izraze te postavljanjem uvjeta neprekinutosti aproksimacijske funkcije.

Bez obzira na navedene nedostatke, programiranje genskih izraza se pokazala kao iznimno moćna metoda, jednostavna za implementaciju i korištenje te sposobna odrediti najbolje aproksimacije nepoznatog skupa točaka.

8. Literatura

1. Wikipedia.org: On the Origin of Species, s Interneta, http://en.wikipedia.org/wiki/On_the_Origin_of_Species , 12. siječnja 2008.
2. Wikipedia.org: Evolutionary algorithm, s Interneta, http://en.wikipedia.org/wiki/Evolutionary_algorithm , 17. ožujka 2008.
3. Golub, M: *Genetski algoritam – prvi dio*, FER, Zagreb, Hrvatska, s Interneta, http://www.zemris.fer.hr/~golub/ga/ga_skripta1.pdf , 2004.
4. Wikipedia.org: Evolution strategy, s Interneta, http://en.wikipedia.org/wiki/Evolution_strategy , 21. siječnja 2008.
5. Wikipedia.org: Evolutionary programming, s Interneta, http://en.wikipedia.org/wiki/Evolutionary_programming , 17. ožujka 2008.
6. Wikipedia.org: Learning classifier system, s Interneta, http://en.wikipedia.org/wiki/Learning_classifier_system , 17. ožujka 2008.
7. Wikipedia.org: Genetic algorithm, s Interneta, http://en.wikipedia.org/wiki/Genetic_algorithm , 23. siječnja 2008.
8. Golub, M: *Genetski algoritam – drugi dio*, FER, Zagreb, Hrvatska, s Interneta, http://www.zemris.fer.hr/~golub/ga/ga_skripta2.pdf , 2004.
9. Wikipedia.org: Genetic programming, s Interneta, http://en.wikipedia.org/wiki/Genetic_programming, 4. veljače 2008.
10. Ferreira, C: „Gene Expression Programming: A New Adaptive Algorithm For Solving Problems“, *Complex Systems*, vol. 13, issue 2, pp. 87-129, SAD, s Interneta, http://www.gene-expression-programming.com/webpapers/gep.pdf , 2001.
11. Golub, M: *Genetski algoritmi*, stranica o temi, s Interneta, http://www.zemris.fer.hr/~golub/ga/ga.html , 16. veljače 2008.
12. Wikipedia.org: Garbage collection (computer science), s Interneta, http://en.wikipedia.org/wiki/Garbage_collection_(computer_science), 19. veljače 2008.
13. Common-Lisp.net: *Lisp in a Box*, službene stranice projekta, s Interneta, http://common-lisp.net/project/lispbox/ , 21. veljače 2008.
14. Zagrebačka burza, službene stranice, s Interneta, http://www.zse.hr/ , 29. veljače 2008.