SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1032

OSTVARENJE KARTEZIJSKOG GENETSKOG PROGRAMIRANJA U OKRUŽENJU ZA EVOLUCIJSKO RAČUNANJE

Dino Jović

Zagreb, Svibanj 2015

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 26. veljače 2015.

Predmet: **Analiza i projektiranje računalom**

DIPLOMSKI ZADATAK br. 1032

Pristupnik: **Dino Jović (0036450078)**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **Ostvarenje kartezijskog genetskog programiranja u okruženju za evolucijsko računanje**

Opis zadatka:

Opisati kartezijsko genetsko programiranje i istražiti mogućnosti njegove primjene. Posebnu pažnju posvetiti problemu traženja Booleovih funkcija za potrebe kriptografskih algoritama. Razraditi postupak rješavanja problema uz pomoć kartezijskog genetskog programiranja. U okviru okruženja za evolucijsko računanje ECF ugraditi mehanizme intervalne aritmetike, linearne regresije i automatski definiranih funkcija. Ispitati učinkovitost u različitim konfiguracijama veličine kartezijskog prikaza i ustanoviti ovisnost konfiguracija o zadanom problemu i prikazati dobivena rješenja. Radu priložiti izvorne tekstove programa uz potrebna objašnjenja i korištenu literaturu.

Zadatak uručen pristupniku: 13. ožujka 2015.
Rok za predaju rada: 30. lipnja 2015.

Mentor:

Izv. prof. dr. sc. Domagoj Jakobović

Predsjednik odbora za diplomski rad profila:

Prof. dr. sc. Siniša Srbljić

Djelovođa:

Doc. dr. sc. Tomislav Hrkać

Sadržaj

1. Uvod

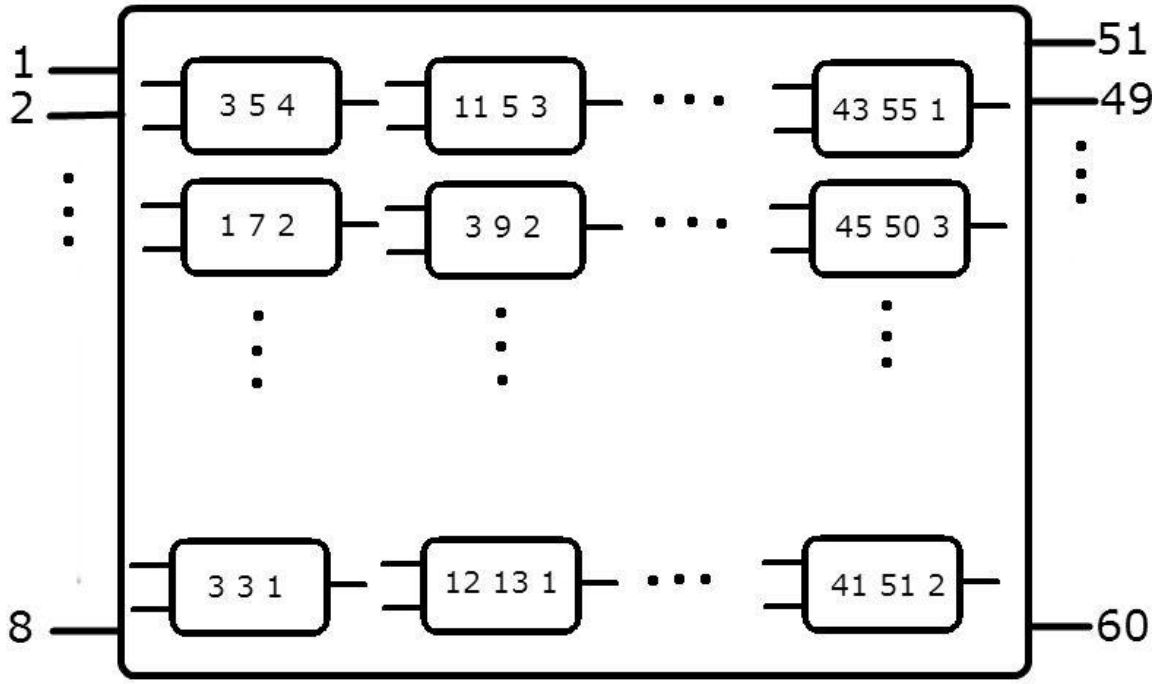
Genetsko programiranje je postupak rješavanja problema oponašajući prirodni proces evolucije. Prednosti genetskog programiranja su mogućnost rješavanja svakog problema u razumnom vremenu, rješavanje problema za koje ne postoji egzaktni način rješavanja i rješavanja problema čije bi egzaktni postupak rješavanja trajao predugo. Nedostaci genetskog programiranja su kada i ako će postupak pronaći najbolje rješenje, te kada postupak i nađe najbolje rješenje ne znamo je li to stvarno jest najbolje rješenje.

Kartezijsko genetsko programiranje je podvrsta genetskog programiranja u kojoj se genom svake jedinke prikazuje kao Kartezijev produkt jednostavnih funkcija. Kartezijskim genetskim programiranjem mogu se rješavati svi evolucijski problemi, ali koristi se većinom za rješavanje problema koji se mogu opisati kao višedimenzionalne matematičke funkcije s neograničenim brojem parametara i ograničenim brojem izlaznih parametara.

U ovom radu se rješavaju problemi simboličke regresije i nelinearnosti Booleovih funkcija kao osnovni problemi u području za testiranje implementacije Kartezijskog genetskog algoritma u okruženju za evolucijsko računanje ECF.

2. Kartezijsko Genetsko Programiranje

Kartezijsko genetsko programiranje prikazuje jedinku kao Kartezijev produkt jednostavnih funkcija koji tvore složenu funkciju. Kao i svaka vrsta genetskog programiranja, Kartezijsko genetsko programiranje ima parametre koje je potrebno podesiti kod svakog problema posebno. Kartezijsko genetsko programiranje ima dosta parametara koje treba namjestiti pa samim time i traženje dobrih parametara traje dulje nego kod osnovnih podvrsta genetskog programiranja.

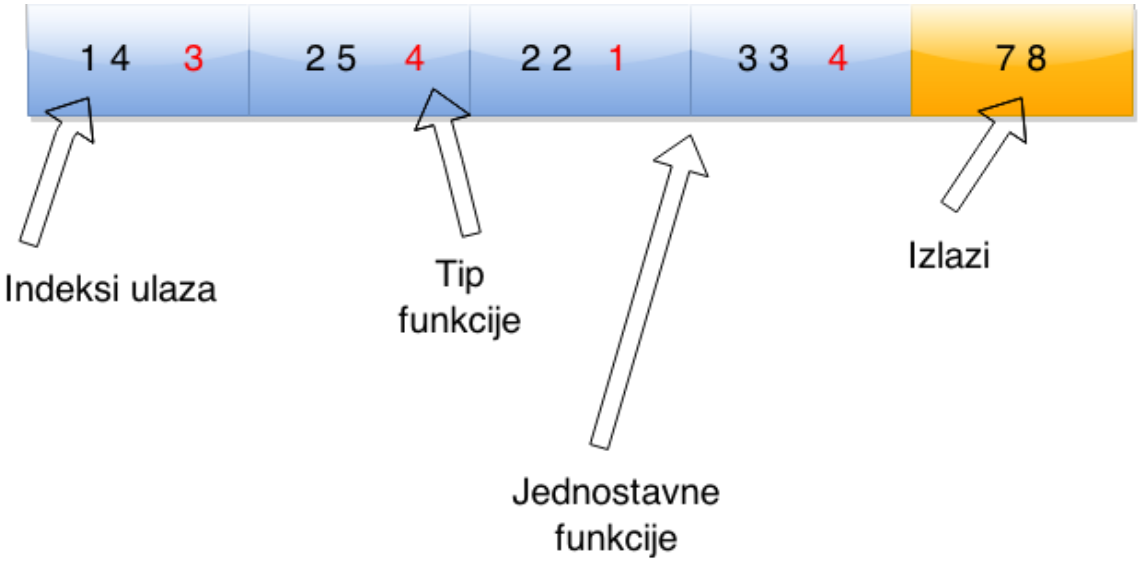


Slika 2.1. Model Jedinke u Kartezijskom genetskom algoritmu. Preuzeto iz [9].

Na slici 2.1. prikazan je model jedinke iz koje vidima da se na ulazu u jedinku nalaze se varijable koje mogu biti na ulazima jednostavnih funkcija u prvom stupcu. Svaka jednostavna funkcija je opisana nizom cijelih brojeva. Prvih N-1 elemenata niza opisuju koje su varijable ili izlazi jednostavnih funkcija u

prethodnim stupcima na ulazu trenutne funkcije. Zadnji element u nizu je indeks imena funkcije iz skupa funkcija koja će se izvršiti nad ulazima u ovu jednostavnu funkciju. Brojevi na izlazu iz cijele jedinke su indeksi onih jednostavnih funkcija čiji će se izlazi koristiti za izlaz cijele jedinke.

Jedinka je u memoriji prikazana kao polje cijelih brojeva. Prvi dio polja su jednostavne funkcije. Prvi dio je podijeljen na svaku jednostavnu funkciju posebno gdje je svaka jednostavna funkcije podijeljena još na niz ulaznih parametara i indeks funkcije iz koja se računa nad tim ulazima. Drugi dio jedinke su samo indeksi jednostavnih funkcija čiji će se izlazi koristiti kao izlazi cijele jedinke.



Slika 2.2. Jedinka u memoriji

Na slici 2.2. je prikazan izgled jedinke u memoriji. Plavi dio jednike su jednostavne funkcije a crveni dio su izlazi. Crni brojevi su indeksi ulazni varijabli ili drugih jednostavnih funkcija. Žuti brojevi su indeksi funkcija u skupu funkcija koji se koriste.

2.1. Parametri

Osnovni parametri koji moraju biti postavljeni kako bi program mogao raditi su:

- Tip podataka: Jedan od osnovni tipovi podataka kojima se prikazuju brojevi u računalu (int, double, uint). Program mora znati koji su tip podataka ulazne varijable tako da zna koje funkcije smije koristiti.

- Broj ulaznih podataka: Broj različitih varijabli koje mogu dobiti sve funkcije iz prvog stupca na svoje ulaze.
- Broj izlaznih podataka: Broje podataka koji će se dobiti kao izlaz jedne cijele funkcije.
- Maksimalan broj ulaza jednostavne funkcije: Broj ulaza koji će imati najsloženija jednostavna funkcija.
- Broj redova: Koliko će redova jednostavnih funkcije biti u Kartezijskom produktu.
- Broj stupaca: Koliko će stupaca jednostavnih funkcija biti u Kartezijskom produktu.
- Koeficijent povezanosti unatrag: Broj koji označava maksimalnu udaljenost indeksu stupca iz koje funkcija smije uzimati podatke za svoje ulaze.
- Skup funkcija: Popis svih jednostavnih funkcija koje se mogu koristiti kod računanja svake jednike.

Dodatni parametri koji nisu potrebni za rad algoritma ali mogu mu poboljšati kvalitete:

- Koeficijent mutacije: Mora biti definirana barem jedna mutacija i vjerojatnost s kojom će se ta mutacija događati.
- Dodatne mutacije: Algoritam smije imati više od jedne mutacije
- Koeficijente križanja: Definirati koja će se križanja koristiti i koja je vjerojatnost svakog od njih. Ukupna vjerojatnost mora biti 1.
- Korištenje intervalne aritmetike: (samo bool vrijednost koja označava dali se koristi ili ne) Omogućava računanje s intervalima a ne s brojevima, rješava problem dijeljenja nulom, korijena i logaritma negativnih brojeva. Značajno usporava izvođenje programa.

Na slici 2.1.1. je prikazan popis svih parametara za kartezijski genotip.

```
<Cartesian>
    <Entry key="type">uint</Entry>
    <Entry key="numvariables">2</Entry>
    <Entry key="numoutputs">1</Entry>
    <Entry key="numinputconns">2</Entry>
    <Entry key="numrows">4</Entry>
    <Entry key="numcols">4</Entry>
    <Entry key="levelsback">2</Entry>
    <Entry key="functionset">AND OR NOT XOR XNOR</Entry>
    <Entry key="mut.onepoint.prob">0.2</Entry>
    <Entry key="mut.onegate.prob">0.2</Entry>
    <Entry key="crx.gateOnePoint">0.3</Entry>
    <Entry key="crx.gateRandom">0.1</Entry>
    <Entry key="crx.gateUniform">0.1</Entry>
    <Entry key="crx.onepoint">0.3</Entry>
    <Entry key="crx.random">0.1</Entry>
    <Entry key="crx.uniform">0.1</Entry>
    <Entry key="IntervalArithmetic">true</Entry>
</Cartesian>
```

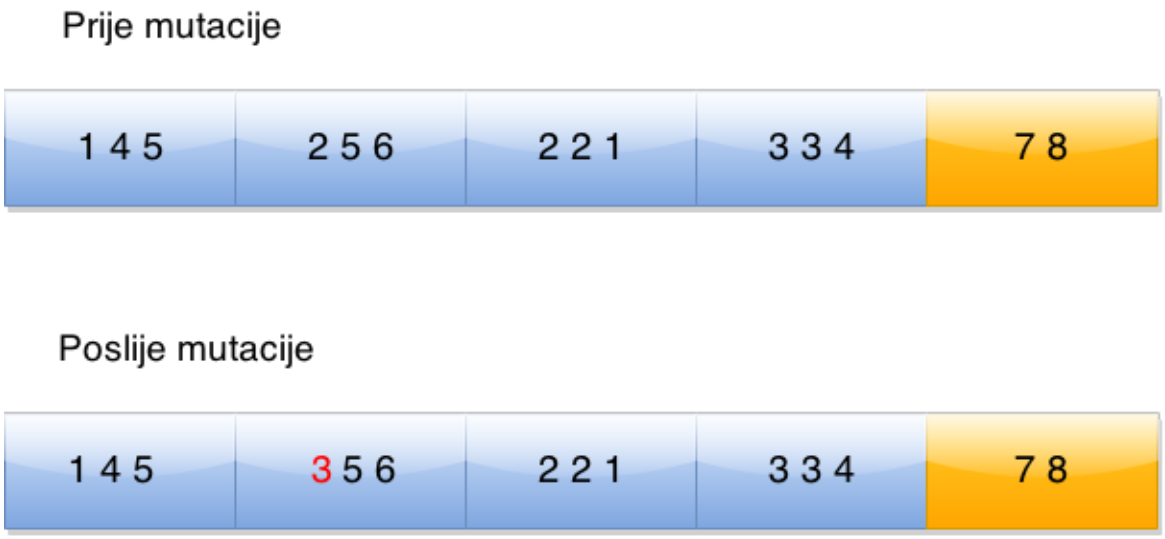
Slika 2.1.1. Parametri Kartezijskog genetskog programiranja

2.2. Mutacija

Mutacija je postupak unošenja novog genetskog materijala u populaciju jedinki, zato jer početna populacija jedinki je konačna dok je prostor rješenja beskonačan ili barem veći od broja jedinki za nekoliko redova veličina. Mutacijom se obično pogoršava jedinka, ali mutacija je najvažniji operator u svim vrstama genetskog programiranja jer služi za povećanje prostora pretraživanja i zbog toga što su mutacije bazirane na slučajnosti osiguravamo da će se nakon beskonačnog vremena pretražiti cijeli prostor rješenja.

Mutacija kod kartezijskolg genetskog algoritma nije jednostavna kao kod ostalih genetskih algoritama. To je posljedica uređenosti jedinke i zato kada se mutira indeks u polju na kojem je zapisan podatak kojoj je to funkciji iz skupa funkcija riječ. Na to mjesto smije doći samo broj koji je manji od broja funkcija koje su raspoložive. Također kada mutiramo mjesto koje govori o ulazu u funkciju moramo biti sigurni da će taj broj biti neka funkcija koja je prije trenutne ali također i da nije dalje nego što to dopušta koeficijent povezanosti unatrag.

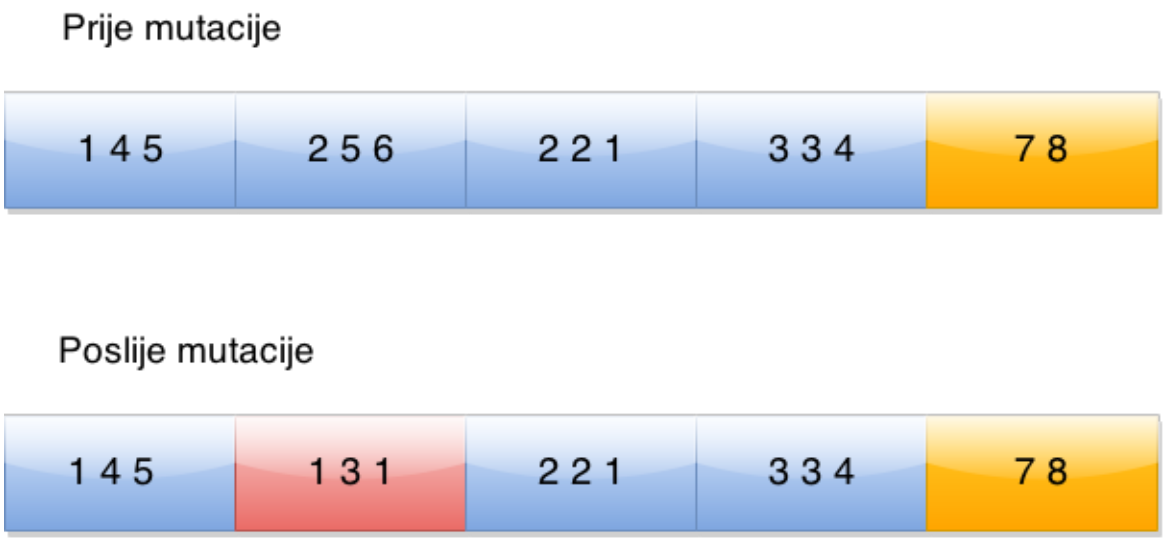
2.2.1. Mutacija jedne točke



Slika 2.2.1.1. Mutacija u jednoj točci

Na slici 2.2.1.1. prikazan je primjer moguće mutacije u jednoj točci. Mutacija u jednoj točci se izvodi tako da se odabere jedan slučajan element polja i zamjeni se slučajnim elementom iz skupa brojeva koji se mogu nalaziti na tom indeksu.

2.2.2. Mutacija jedne jednostavne funkcije



Slika 2.2.2.1. Mutacija jedne jednostavne funkcije.

Na slici 2.2.2.1 prikazan je primjer moguće mutacije jednostavne funkcije. Mutacija se izvodi tako da se slučajno odabere jedna jednostavna funkcija te se mutiraju svi njeni parametri (ulazi i matematička funkcija koja se izvršava nad tim ulazima)

2.3. Selekcija

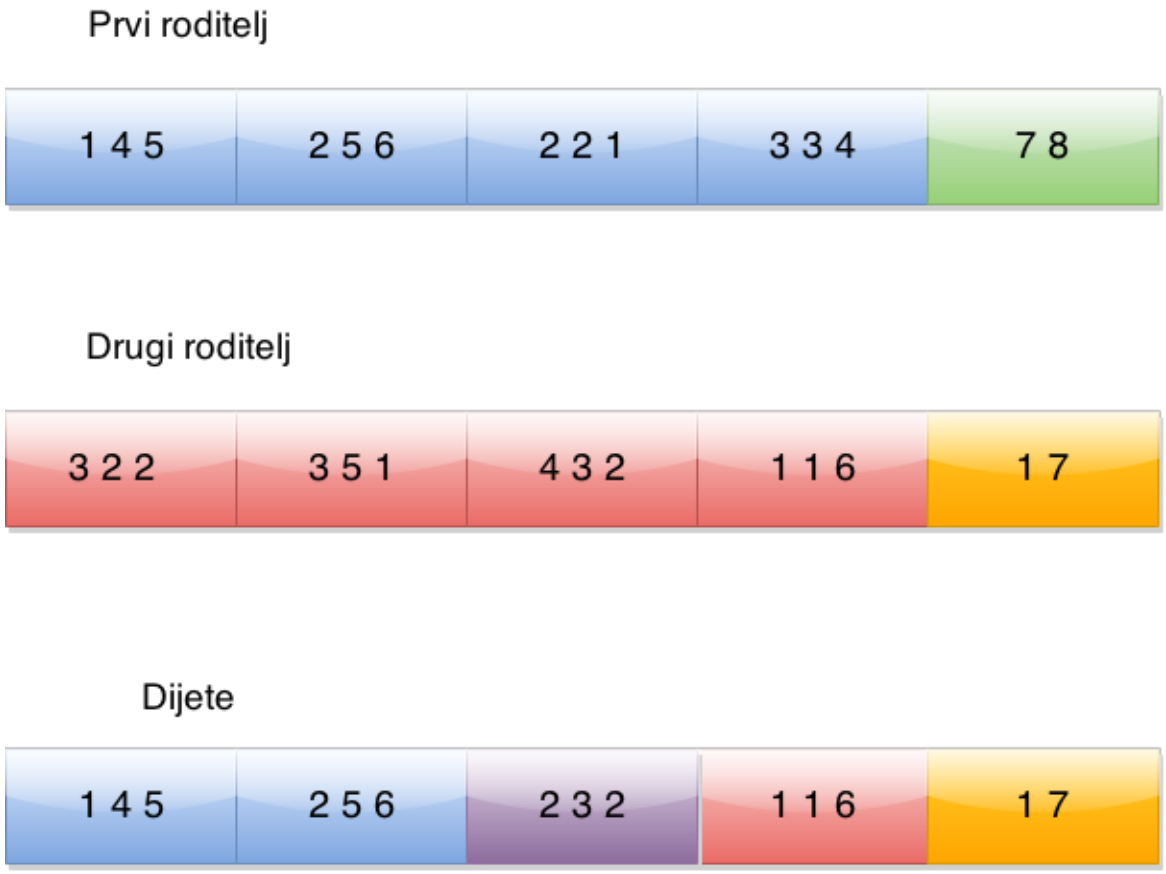
Selekcija je postupak odabira jedinki koje će se križati za slijedeću generaciju. Kod Kartezijskog genetskog programiranja mogu se koristiti sve vrste selekcije. U ovom radu je odabrana turnirska selekcija s turnirom veličine 3, te elitizam u kojem se samo najbolja jedinka prenese u slijedeću generaciju.

2.4. Križanje

Križanje je postupak u kojem se iz dvije odabrane jedinke uzimaju neki geni i stvara se nova jedinka koja bi trebala imati dobra svojstva od oba roditelja. Križanjem se smanjuje genetska raznolikost populacije ali zato dobivamo bolje jedinke.

Križanje kod kartezijskog genetskog algoritma je jednostavnije nego mutiranje jer samo kopiramo dijelove gena na istom indekse u dijete i zato jedinka zadovoljava sve uvjete koji su potrebni da bi jedinka bila u obliku Kartezijskog produkta.

2.4.1. Križanje s jednom točkom prekida na cijelom polju

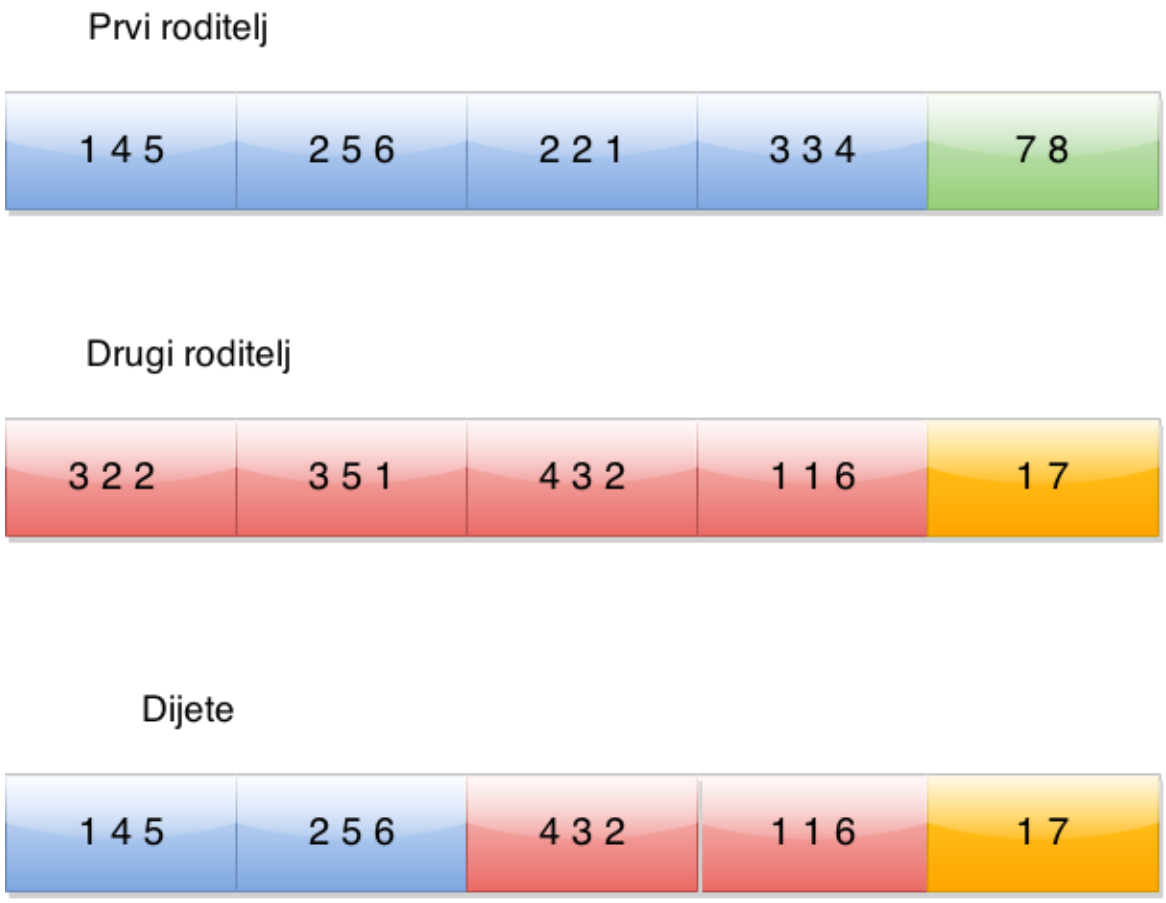


Slika 2.4.1.1. Križanje s jednom točkom prekida.

Na slici 2.4.1.1. prikazan je primjer križanja s jednom točkom prekida. Odabire se jedna točka koja će biti kao granica između dva genom te se odabire slučajan roditelj i iz njega se uzmu svi podaci do te točke. Preostali podaci se uzimaju iz drugog roditelja.

Podaci o izlazima su uzeti iz drugog roditelja osim ako je točka prekida toliko pomaknuta da dođe do dijela genoma s izlazima.

2.4.2. Križanje s jednom točkom prekida po jednostavnim funkcijama

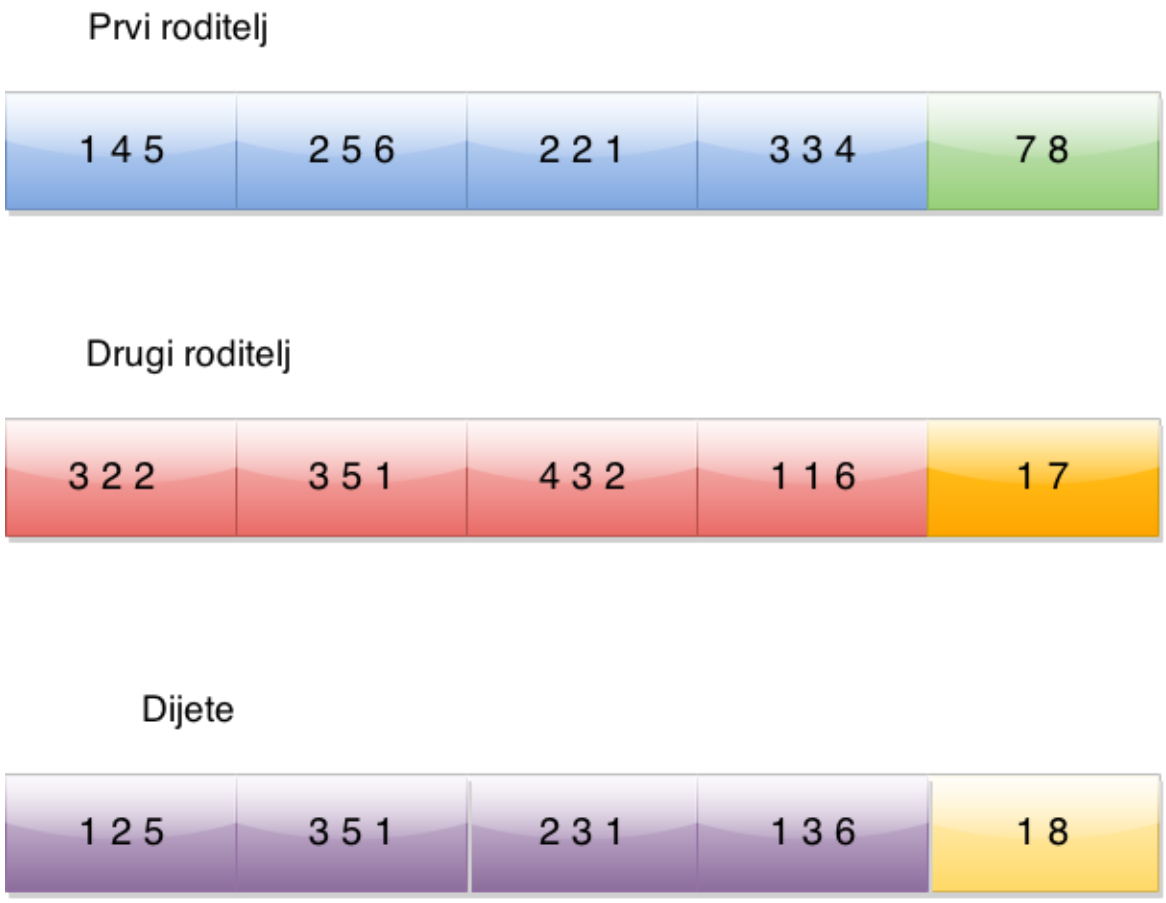


Slika 2.4.2.1. Križanje s jednom točkom prekida po jednostavnim funkcijama.

Na slici 2.4.2.1. prikazan je primjer križanja s jednom točkom prekida po jednostavnim funkcijama. Odabire se jedna točka koja mora biti na granici između dvije jednostavne funkcije te se slučajno odabire prvi roditelj i prenosi se dio genoma do te točke u dijete. Dijete dobije drugi dio genoma od drugog roditelja uključujući i izlaze.

Izlazi se uvijek prenose iz drugog roditelja.

2.4.3. Uniformno križanje na cijelom polju

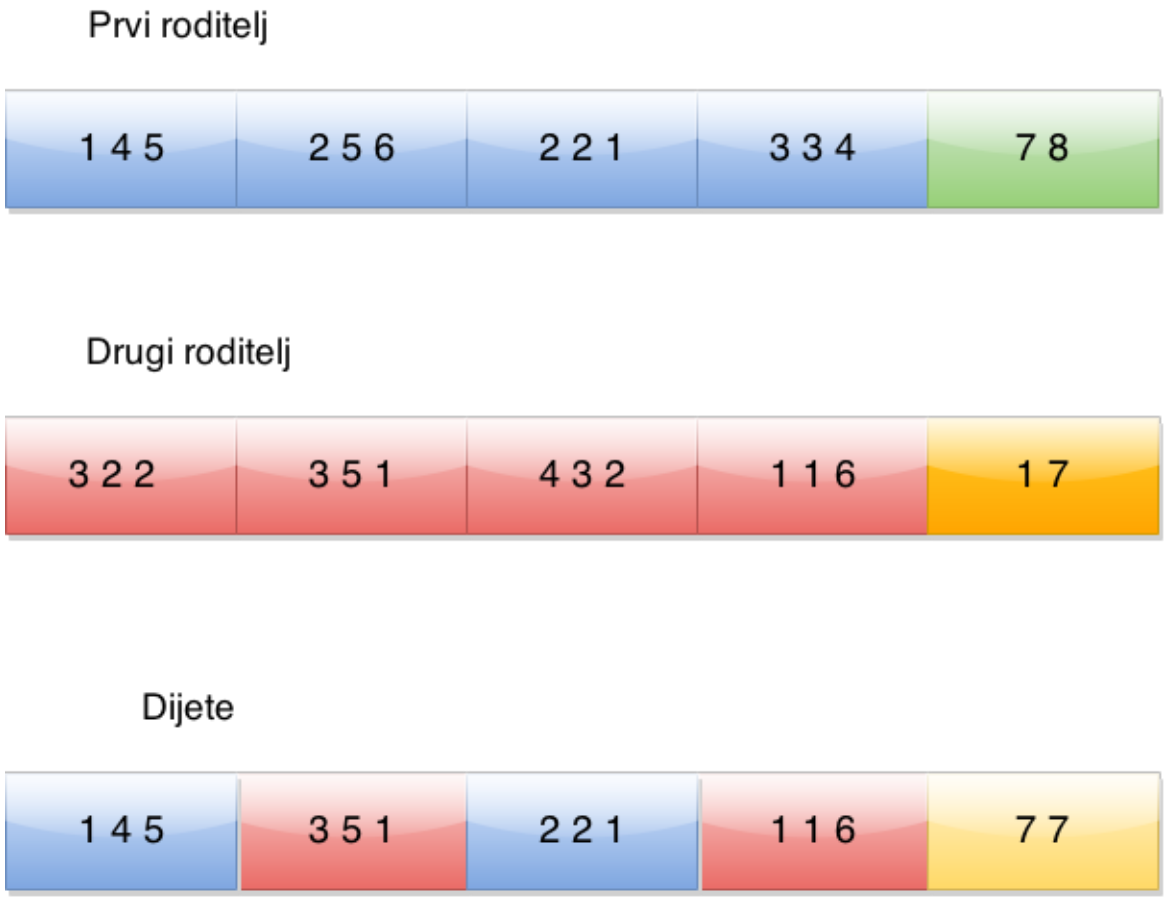


Slika 2.4.3.1. Uniformno križanje po cijeloj jedinci.

Na slici 2.4.3.1. prikazan je primjer uniformnog križanja. Slučajno se odabire prvi roditelj i dijete dobije sve elemente koji su sa parnim indeksom iz prvog roditelja. Neparne elemente dobije od drugog roditelja.

Izlazi se dijele tako da ih dijete dobije po istom principu parno neparno.

2.4.4. Uniformno križanje po jednostavnim funkcijama



Slika 2.4.4.1. Uniformo križanje po jednostavnim funkcijama.

Na slici 2.4.4.1. prikazan je primjer uniformnog križanja po jenostavnim funkcijama. Slučajno se odabire prvi roditelj i dijete dobije sve parne jednostavne funkcije od prvog roditelja. Neparne jednostavne funkcije dobije od drugog roditelja.

Izlaze dijete dobiva kao i kod uniformnog križanja tj. dobije svaki izlaz on nekog roditelja ovisno o parnosti indeksa na kojem se nalazi taj izlaz.

2.4.5. Slučajno križanje na jednostavnim funkcijama

Slika 2.4.5.1. Slučajno križanje po cijelom polju.

Na slici 2.4.5.1. prikazan je primjer slučajnog križanja. Za svaki element polja se odabire slučajan roditelj i on se prenosi u dijete. Izlazi se prenose po istom principu.

2.4.6. Slučajno križanje po jednostavnim funkcijama

Slika 2.4.6.1. Slučajno križanje po jednostavnim funkcijama.

Na slici 2.4.6.1. prikazan je primjer slučajnog križanja po jednostavnim funkcijama. Za svaku jednostavnu funkciju se odabire slučajan roditelj i svi podaci o toj jednostavnoj funkcije se prenesu u dijete.

Za svaki izlaz se posebno odabire iz kojeg će se roditelja preuzeti.

3. Intervalna Aritmetika

Intervalna aritmetika je postupak zaštite od računanja funkcija koje nisu definirane na cijelom intervalu realnih brojeva.

Problem u genetskom programiranju koji koristi matematičke funkcije je u tome što u funkciju može doći bilo koji broj, pa tako možemo imati dijeljenje s nulom ili logaritam negativnih brojeva.

Standardni pristup je pisanje zaštićenih funkcija koji će na početku provjeravati jesu li ulazni podaci u području domene. Ako su podaci u domeni onda se normalno računa, a ako nisu onda se vraća neka prethodno definirana vrijednost. Međutim ta vrijednost je samo stvar dogovora i nema matematičko opravdanje; npr. dijeljenje s nulom je nedefinirano ali u genetskom algoritmu nula može doći kao djelitelj. Mogući dogovori su da se vraća djeljenik ili neku konstantu poput broja 1.

Intervalna aritmetika je metoda računanja razvijena za postavljanje granice na pogreške zaokruživanja i pogreške u mjerenju s ciljem da dobiveni rezultati imaju veće pouzdanje. Svaki broj se u intervalnoj aritmetici predstavlja kao raspon mogućih vrijednosti.

Umjesto da se koristi nepouzdan podatak za računanje koristi se dvije granice između kojih bi trebao biti podatak sa jako velikom vjerojatnošću. Intervalna aritmetika se provodi tako da se matematičke operacije zamjene s intervalima operacijama.

- $[a, b] + [c, d] = [a + c, b + d]$
- $[a, b] - [c, d] = [a - d, b - c]$
- $[a, b] \cdot [c, d] = [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)]$
- $\frac{[a, b]}{[c, d]} = \left[\min\left(\frac{a}{c}, \frac{a}{d}, \frac{b}{c}, \frac{b}{d}\right), \max\left(\frac{a}{c}, \frac{a}{d}, \frac{b}{c}, \frac{b}{d}\right)\right]$

Slika 3.1. Osnovne intervalne operacije.

Na slici 3.1. su prikazane osnovne zamjene za standardne operacije zbrajanja, oduzimanja, množenja i dijeljenja. Ostale operacije se mogu izvesti iz njih.

Problem kod intervalne aritmetike je jako veliko usporavanje programa zbog dvostruko većeg broja računanja. Taj se problem može donekle riješiti ako intervalnu aritmetiku koristimo samo na funkcijama za koje je moguće dobiti graške.

Dodatno treba naglasiti da postoje funkcije koje matematički mogu primiti sve realne brojeve, a na računalu ne mogu. Takva operacija je obično zbrajanje jer na

računalu može doći do prekoračenja opsega brojeva koje je moguće prikazati na računalu pa će samim time rezultat biti netočan.

Ime Funkcije	Simbol	Intervalna Aritmetika
Zbrajanje	+	DA
Oduzimanje	-	DA
Množenje	*	DA
Dijeljenje	/	DA
Sinus	sin	NE
Kosinus	cos	NE
Apsolutno	abs	NE
Pozitivno	poz	NE
Korijen	sqrt	DA
Eksponent	exp	DA
Logaritam	log	DA
I	AND	NE
Ili	OR	NE
Ni	NAND	NE
Nili	NOR	NE
Ne	NOT	NE
Eksluzivno ili	XOR	NE
Eksluzivno nili	XNOR	NE
If	If	NE
Nema operacije	nop	NE
Automatski generirane funkcije	agf	PONEKAD
Konstante	const	NE

Tablica 3.1. Popis funkcija i dali se koristi intervalna aritmetika

Iz tablice 3.1. vidimo za koje je funkcije potrebna intervalna aritmetika. Logičke funkcije ne koriste intervalnu aritmetiku jer pretpostavljaju da su svi brojevi različiti od nule jednaki jedan.

Nop (No Operation) je funkcija koja ne radi ništa samo proslijedi svoj ulaz na izlaz.

Agf (Automatski Generirana Funkcija) je kombinacija ostalih funkcija koju program sam napravi pa ovisno o tome od kojih je funkcija napravljena preuzima i njihovo korištenje intervalne aritmetike.

4. Linearno Skaliranje

Linearno skaliranje je postupak u kojem pomaknemo dobivenu izlaznu funkciju s koordinatnih osi na ravninu koja je najbolje rješenje problema koje je moguće dobiti polinomom prvog stupnja.

Da bi mogli koristiti linearno skaliranje, moramo za svaku točku koja je ulaz znati i točno koji je rezultat.

Prvo moramo odrediti ravninu na kojoj će ležati naša funkcije a za to na trebaju nagib i odsječci na koordinatama.

$$b = \frac{\Sigma[(t - \bar{t})(y - \bar{y})]}{\Sigma[(y - \bar{y})^2]}$$

$$a = \bar{t} - b\bar{y}$$

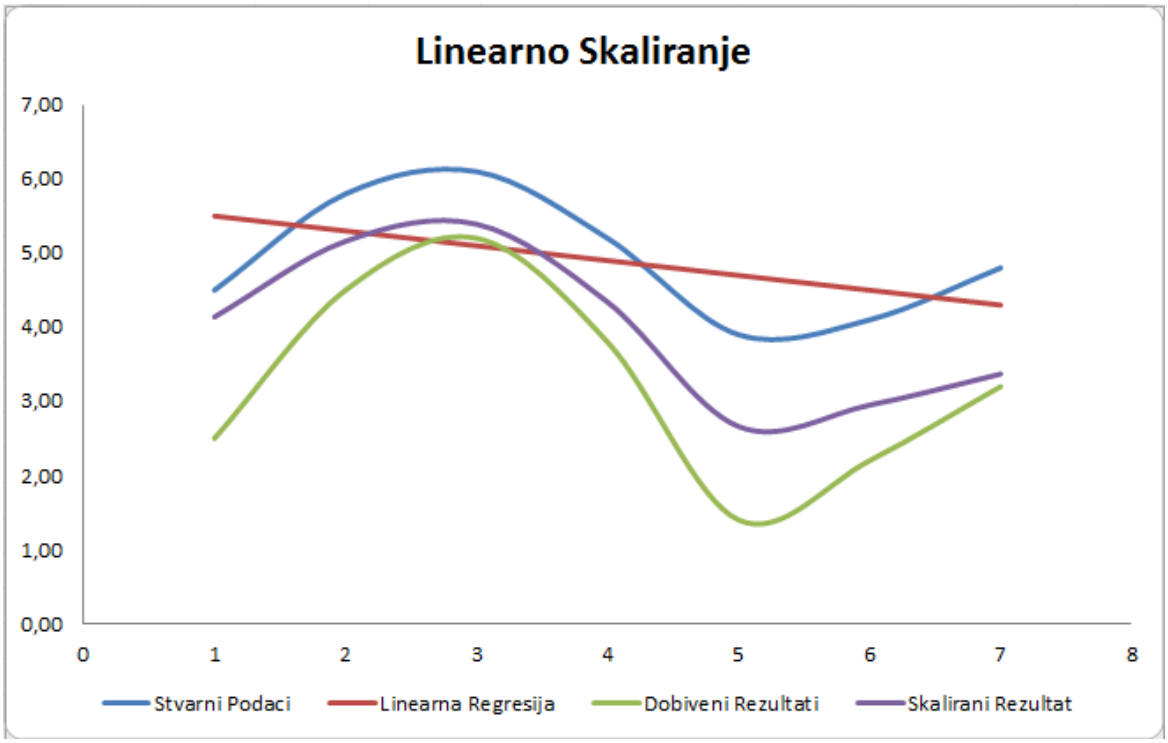
Nakon toga još samo skaliramo izlaze i dodamo ih na ravninu. Npr. koristimo MSE (Mean Squared Error).

$$MSE(t, a + by) = \frac{1}{N} * \sum_{i}^{N} (a + by - t^2)$$

Gdje su:

- $t - ciljana\ vrijednost$
- $\bar{t} - srednja\ ciljana\ vrijednost$
- $y - dobivena\ vrijednost$
- $\bar{y} - srednja\ dobivena\ vijednost$
- $b - nagib\ ravnine$
- $a - odsječan\ ravnine\ na\ koordinatnim\ osima$

Linearno skaliranje ovisi o očekivanim izlazima problema koji se rješava i kao takvo je nemoguće za implementaciju u okruženju te se očekuje od korisnika da ga napiše za svoj problem.



Slika 4.1. Linearno skaliranje

Na slici 4.1. vidimo primjer linearnog skaliranja. Plava linija je stvarna funkcija koju želimo simulirati i zelena funkcija koji smo dobili. Crvena linija je linearna regresija stvarnih podataka a ljubičasta linija je rezultat koji dobijemo nakon linearnog skaliranja.

5. Simbolička Regresija

Simbolička regresija je problem u kojem se pokušava pronaći analitički oblik funkcije ako poznajemo njene vrijednosti u nekoliko točaka. Dodatno smijemo koristiti samo ograničenim brojem matematičkih funkcija.

Funkcije za simboličku regresiju na kojima će biti testirano kartezijsko genetsko programiranje su:

$$1.)\ \log(x+1)+\log(x^2+1)$$

$$2.)\ \sin(x)+\sin(y^2)$$

$$3.)\ 2*\sin(x)*\cos(y)$$

$$4.)\ x*y+\sin\big((x+1)*(y-1)\big)$$

$$5.)\ \frac{8}{2+x^2+y^2}$$

$$6.)\ \frac{x^3}{5}+\frac{y^3}{2}-x-y$$

Dobrota ovih problema je ocjenjuje se kao MSE (mean square error). Srednja kvadratna pogreška. To je srednja vrijednost razlike kvadrata dobivene i očekivane vrijednosti. Srednja kvadratna pogreška se računa:

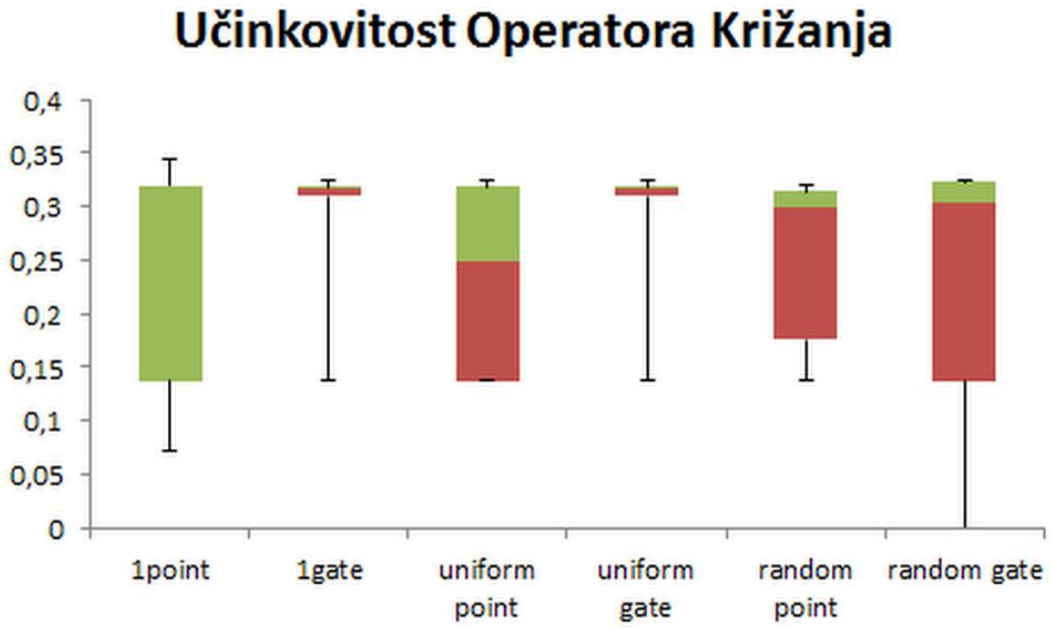
$$\mathrm{MSE}=\frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i-Y_i)^2$$

Skup točaka na kojim su provedena testiranja je skup simetričan ono nule u obje varijable x i y. Gdje su obje nepoznanice iz intervala -10 do 10. Sa razmakom od 2. Što znači da cijela mreža ulaznih podataka sadrži 121 podatak. Ulazni podatci za funkciju svi pomaknuti prema pozitivnim brojevima kako bi broj ulaznih

podataka ostao isti i riješio se problem nedefiniranosti logaritma za negativne brojeve.

5.1. Odabir parametara

Za odabir operatora križanja i mutacije radilo se testiranje u kojem je bila dopuštena samo jedna od tih operacija te se pustilo algoritam da radi na problemu za simboličku regresiju problem broj 6 dok ne izvrši 100000 evaluacija, te se svaki pokus pokretao 10 puta. Dodano su se na isti način određivali i parametri za veličinu populacije i dimenzije cijelog kartezijskog sklopa. Skup jednostavnih funkcija je uvijek bio minimalni skup svih funkcija koje su korištene u tom problemu.

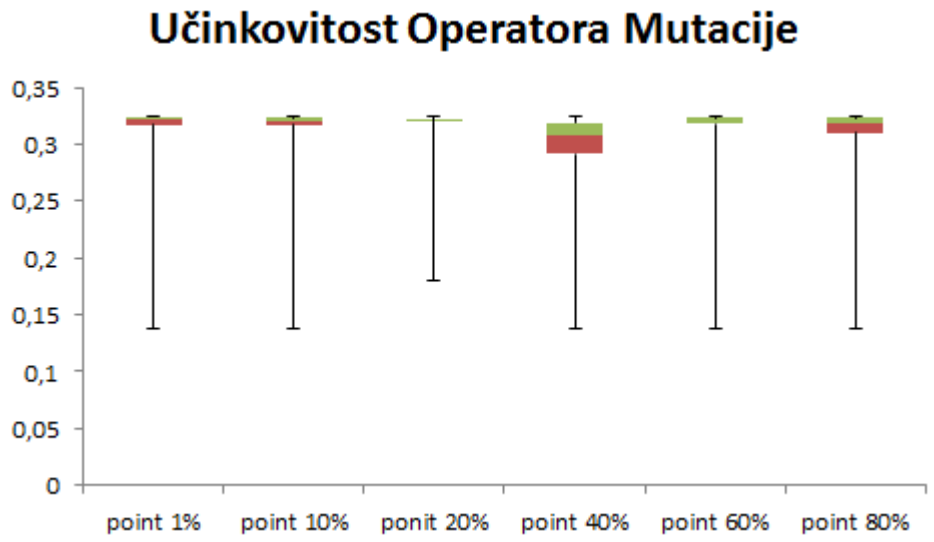


Slika 5.1.1. Rezultati testiranja različitih operatora križanja.

	1 točka po elementima	1 točka po funkcijama	Uniformno po elementima	Uniformno po funkcijama	Slučajno po elementima	Slučajno po funkcijama
Min	0,071694	0,137810	0,137810	0,137810	0,137810	0
Q1	0,137810	0,309515	0,137810	0,309515	0,176030	0,137810
Med	0,137810	0,317486	0,248137	0,316613	0,298455	0,302827
Q3	0,319954	0,318991	0,318235	0,317860	0,314247	0,322958
Max	0,343930	0,324533	0,324533	0,324533	0,319378	0,324533

Slika 5.1.1. Rezultati križanja po kvantilima.

Iz rezultata vidimo da su najbolji rezultati postignuti na križanju s jednom točkom prekida po svakom elementu jedinke, dok su oba slučajna križanja i uniformo križanje po jednostavnim funkcija postigli usporedive rezultate s najboljim s razlikom da su imali neka pokretanja kada su rezultati lošiji. Važno je napomenuti da je dobrota najbolje jedinke u prvog generaciji koja se dobije slučajnošću oko 350.

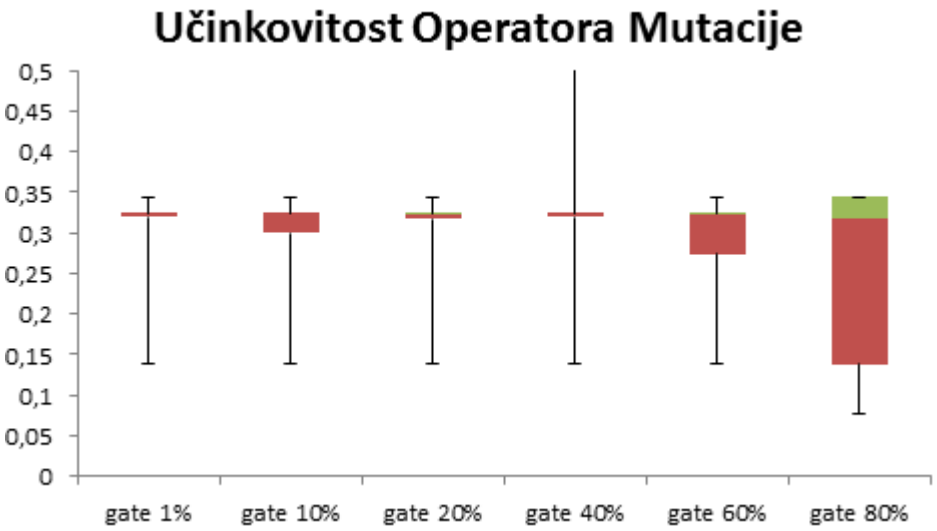


Slika 5.1.2. Rezultati testiranja operatora mutacije s jednom točkom prekida sa različitom vrijednosti koeficijenta mutacije.

	1%	10%	20%	40%	60%	80%
Min	0,137810	0,137810	0,179538	0,137810	0,137810	0,137810
Q1	0,317019	0,317019	0,319378	0,291166	0,318235	0,309889
Median	0,321956	0,319378	0,319378	0,307150	0,318807	0,318540
Q3	0,324533	0,323244	0,319538	0,319092	0,323244	0,323244
Max	0,324533	0,324533	0,324533	0,324533	0,324533	0,324533

Tablica 5.1.2. Rezultati mutacija s jednom točkom prekida po kvantilima.

Iz rezultata mutacija prikazanim na slici 5.1.2. i tablici 5.1.2. vidimo da je najbolja bila mutacija s vjerojatnosti od 40% .



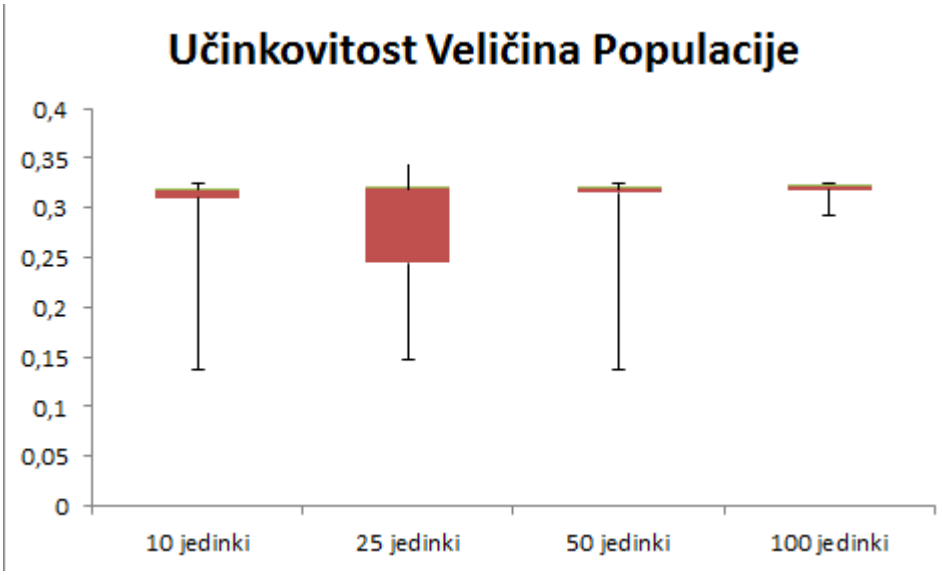
Slika 5.1.3. Učinkovitost operatora mutacije nad jednostavnim funkcijama

	1%	10%	20%	40$	60%	80%
Min	0,137810	0,137810	0,137810	0,137810	0,137810	0,076319
Q1	0,31981	0,29969	0,316644	0,31981	0,273829	0,13781
Median	0,324533	0,324533	0,321956	0,324533	0,322062	0,318235

Q3	0,324533	0,324533	0,324533	0,324533	0,324533	0,34393
Max	0,343930	0,343930	0,343930	1,659820	0,343930	0,343930

Tablica 5.1.3. Rezultati mutacije na jednostavnim funkcijama

Iz slike i tablice 5.1.3. vidimo da je za mutaciju po jednostavnim funkcija potrebna veća vjerojatnost da bi rezultati bili ekvivalenti mutaciji po jednoj točci. Te ovdje vidimo jedini primjer u kojem dobrota nije bila smanjena ispod 1.

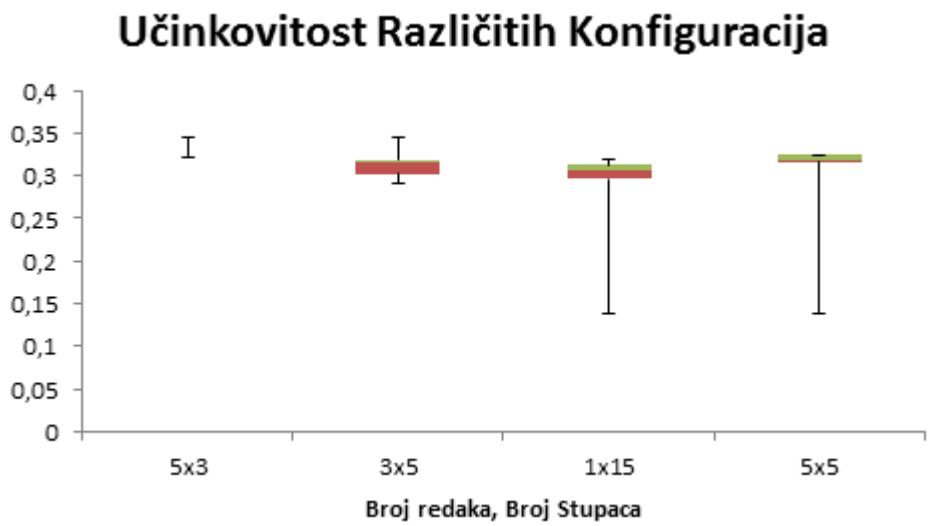


Slika 5.1.4. Učinkovitost veličina populacije

	10 jedinki	**25 jedinki**	**50 jedinki**	**100 jedinki**
Min	0,137810	0,146393	0,137810	0,292870
Q1	0,309921	0,244446	0,315077	0,318521
Median	0,318235	0,318807	0,318807	0,321956
Q3	0,319347	0,319378	0,319538	0,324533
Max	0,324533	0,343930	0,324533	0,324533

Tablica 5.1.4. Učinkovitost veličina populacije

Na slici 5.1.4. i tablici 5.1.4 vidimo da je najbolja veličina populacije jedinki 25



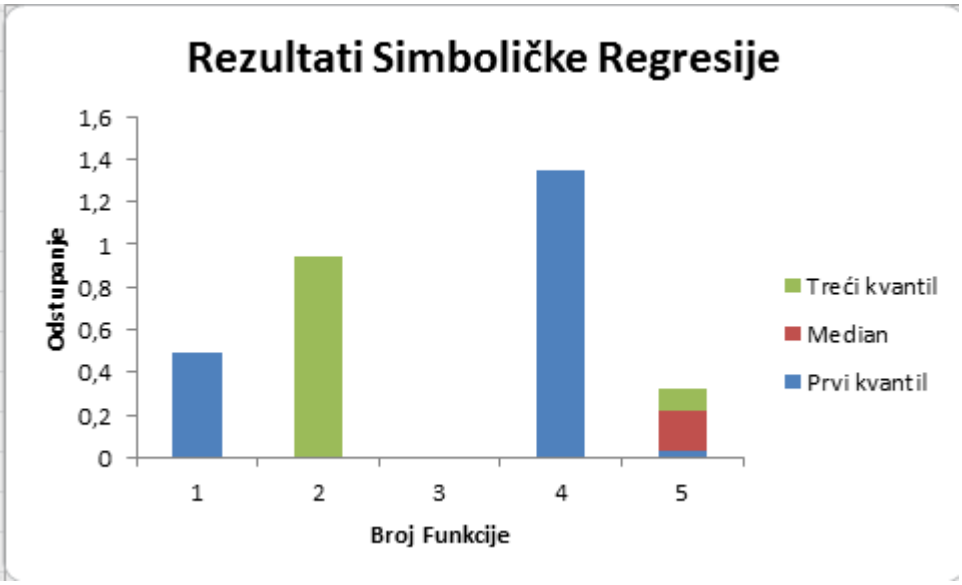
Slika 5.1.5.Učinkovitost različitih konfiguracija cijelog sklopa

	5x3	**3x5**	**1x15**	**5x5**
Min	0,320944	0,290690	0,137810	0,137810
Q1	0,324533	0,301656	0,296469	0,316002
Median	0,324533	0,315319	0,306219	0,318540
Q3	0,324533	0,319121	0,312646	0,324533
Max	0,343930	0,343930	0,319378	0,324533

Tablica 5.1.5. Rezultati učinkovitosti različitih testiranja

Na slici 5.1.5. i tablici 5.1.5. prikazani su rezultati za različite konfiguracije (broj redaka i stupaca). Izlazi iz cijelog sklopa je mogao biti samo ograničen broj jednostavnih funkcija kako bi se osigurala ravnopravnost između sklopova s različitim brojem jednostavnih funkcija. Povećan broj jednostavnih funkcija nije pridonio poboljšanju. A najbolja konfiguracija je bila ona s jednim redim i maksimalnom povezanošću unatrag.

5.2. Rezultati



Slika 5.2.1. Rezultati simboličke regresije

Na slici 5.2.1. su prikazani rezultati simboličke regresije za prvih pet funkcija. Šesta funkcija nije prikazana na slici zbog mjerila u kojem su njezini rezultati. Podaci o šestoj funkciju su dani u tablici 5.2.1. Funkcija šest zbog velikih raspona u kojima se nalaze rezultati kodomene ima veće apsolutne pogreške.

	Funkcija1	Funkcija2	Funkcija3	Funkcija4	Funkcija5	Funkcija6
Min	0,487259	0	0	1,344500	0	118,797000
Q1	0,487259	0	0	1,344500	0,034453	123,213000
Median	0,487259	0	0	1,344500	0,222480	135,094000
Q2	0,487259	0,944423	0	1,344500	0,322959	172,704250
Max	0,487259	1,259230	0	1,344500	0,324533	221,593000

Tablica 5.2.1. Rezultati simboličke regresije

Iz rezultata u tablici 5.2.1. možemo vidjeti da je na funkcijama dva, tri i pet pronađena točna funkcija. Kod funkciji tri vidimo da je točna funkcija uspješno pronađena u svim pokretanjima. Funkcije jedan i četiri su zapinjale u lokalnom minimumu blizu nule. Na funkcijama pet i šest je pronađeno dosta različitih

rezultata. Od kojih je u funkcije pet pronađen i točan rezultat dok u funkciji šest nije.

6. Logičke Funkcije

Logičke funkcije su funkcije koje u domeni i kodomeni imaju samo dva elementa, istinu i laž. Takve funkcije se koriste u kriptografiji, gdje moraju imati funkciju preslikavanja što više nelinearnu kako bi otežali dekriptiranje. Te funkcije moraju zadovoljavati dodatne parametre od kojih je nelinearnost najvažniji. U ovom radu ćemo optimizirati ove parametre ovim funkcijama dobrote:

1. Suma balansiranosti i nelinearnosti. Funkcija zadovoljava karakteristiku propagacije stupnja t ako su sve derivacije vektora koje imaju Hammingovu težinu manju od t balansirane, što veće, to bolje.
2. Suma balansiranosti, nelinearnosti, korelacijskog imuniteta i algebarskog imuniteta. Udaljenost između Booleove funkcije i skupa funkcija sa linearnom strukturom (služi za GAC) - što manji, to bolje.
3. Suma balansiranosti, nelinearnosti, korelacijskog imuniteta, algebarskog imuniteta i algebarskog stupnja. Uz uvjet da je balansiranost veća ili jednaka nuli i korelacijski imunitet je veći ili jednak dva. Ako uvjet nije zadovoljen onda samo suma balansiranosti i korelacijskog imuniteta. Što veće to bolje.

Objašnjena značajki:

- Balansiranost: Balansirana logička funkcija je ona funkcija koja će u tablici istinitosti imati jednak broj nula i jedinca , što znači da je vjerojatnost pojave nule i jedinice na izlazu jednaka 0.5. Zbog toga što nemaju pristranost nekom rezultatu takve funkcije su poželjne u kriptografiji. [10]
- Nelinearnost: Nelinearnost je udaljenost funkcije od skupa afinih funkcija s m varijabli gdje se udaljenost računa kao minimalna Hammingova udaljenost. [11]
- Korelacijski imunitet: je mjera do kojeg su stupnja izlazi nepovezani s ulazima. Za funkciju možemo reći da je korelacijski imuna s vrijednošću m ako je svaki podskup sa m ili manje varijabli u skupu n gdje vrijedi da je m

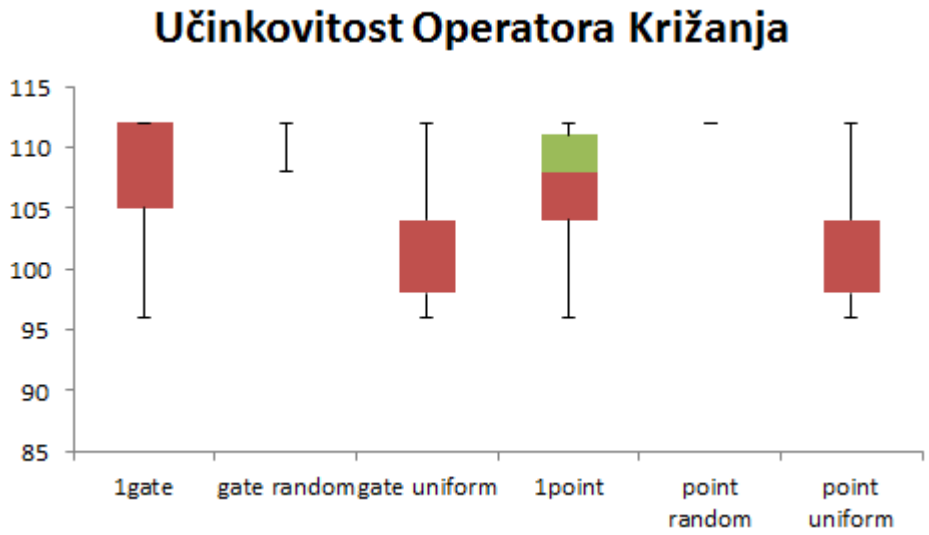
manji od n statistički nepovezan sa rezultatima funkcije nad cijelim skupom od n varijabli. [12]

- Algebarski imunitet je otpornost logičke funkcije na algebarski napad ako je imunitet funkcije prenizak i ona se koristi u kriptografiji, zaštita se može lako probiti. Algebarski imunitet je računski najzahtjevnija kriptografska značajka. [13] [15]
- Algebarski stupanj: Pretpostavimo da je Hamming-ova težina monoma definirana kao broj varijabli koje su u njemu. Algebarski stupanj monoida je jednak njegovoj težini, dok je algebarski stupanj funkcije jednak najvećem algebarskom stupnju njenih monoida. [14]

6.1. Odabir parametara

Za odabir operatora križanja i mutacije provedena su ispitivanja na funkciji dobrote broj dva. Svaki pokretanje je trajalo 100000 evaluacija te je svaki operator pokrenut 10 puta.

Dodatno provedena su ispitivanja o veličini populacije, obliku sklopa i skupu jednostavnih funkcija koje je moguće koristi. Kod ispitivanja oblika sklopa uvijek je omogućena maksimalna povezanost unatrag.

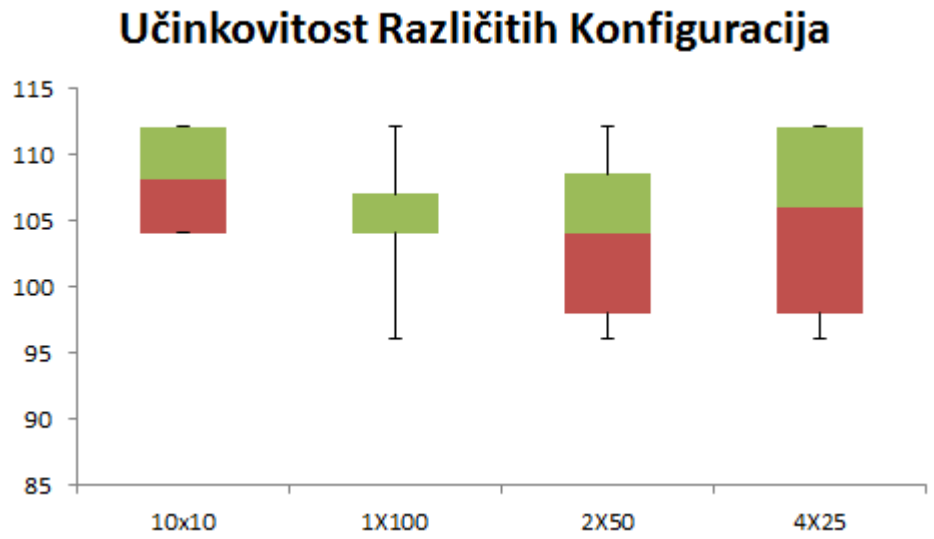


Slika 6.1.1 Učinkovitost operatora križanja

	1gate	gate random	gate uniform	1point	point random	point uniform
Min	96	108	96	96	112	96
Q1	105	112	98	104	112	98
Median	112	112	104	108	112	104
Q3	112	112	104	111	112	104
Max	112	112	112	112	112	112

Tablica 6.1.1. Rezultati operatora križanja

Iz tablice i slike 6.1.1. vidimo da su najbolji rezultati postignuti na križanjima koja su temeljena na slučajnim odabiru.



Slika 6.1.2. Učinkovitost različitih konfiguracija

	10x10	1X100	2X50	4X25
Min	104	96	96	96
Q1	104	104	98	98
Median	108	104	104	106

Q3	112	107	108,5	112
Max	112	112	112	112

Tablica 6.1.2. Rezultati različitih konfiguracija

Na slici i tablici 6.1.2. prikazani su rezultati za različite konfiguracije. Svaka konfiguracija je imala maksimalni koeficijent povezanosti unatrag dopušten svojim oblikom. Najbolji rezultati su postignuti korištenjem konfiguracije s jednakim brojem redova i stupaca.

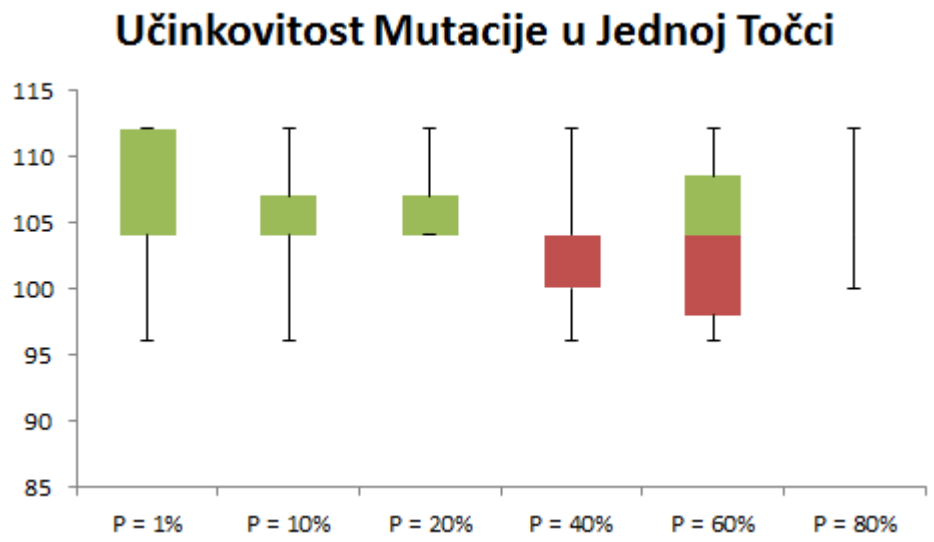
Slika 6.1.3. Učinkovitost različitih skupova jednostavnih funkcija.

	and, or	**or, not**	**and, or, not**	**and, or, not, xor, xnor**	**and, or, nor, nand, xor, xnor, not**	**and, or, not, nor. nand**	**nor, nand, xor, xnor**
Min	84	80	88	96	96	80	11
Q1	88	96	88	96	96	88	96
Median	88	96	96	100	96	94	104

Q3	92	96	96	104	96	96	107
Max	92	96	96	112	100	96	112

Tablica 6.1.3. Rezultati ispitivanja različitih skupova funkcija.

Iz slike i tablice 6.1.3. vidi se da je najvažniji parametar za rezultat ispitivanja skup funkcija koje algoritam smije koristiti. Te najvažnije su funkcije XOR i XNOR. Ali moraju postojati još neke funkcije jer ove nisu dopuštene ali također ako je previše funkcija dopušteno rezultati postaju lošiji.

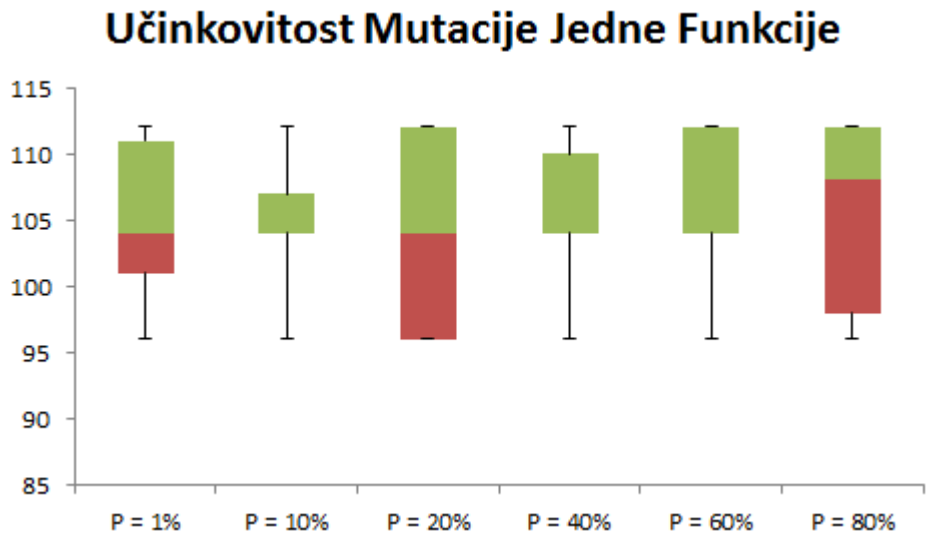


Slika 6.1.4. Učinkovitost mutacije u jednoj točci.

	P = 1%	**P = 10%**	**P = 20%**	**P = 40%**	**P = 60%**	**P = 80%**
Min	96	96	104	96	96	100
Q1	104	104	104	100	98	104
Median	104	104	104	104	104	104
Q3	112	107	107	104	108,5	104
Max	112	112	112	112	112	112

Tablica 6.1.4. Rezultati ispitivanja mutacije u jednoj točci.

Iz slike i tablice 6.1.4. možemo vidjeti da su mutacije približno jednake. Najbolja je mutacija s 1% a poslije postaju sve lošije mutacije, iako su razlike gotovo zanemarive.

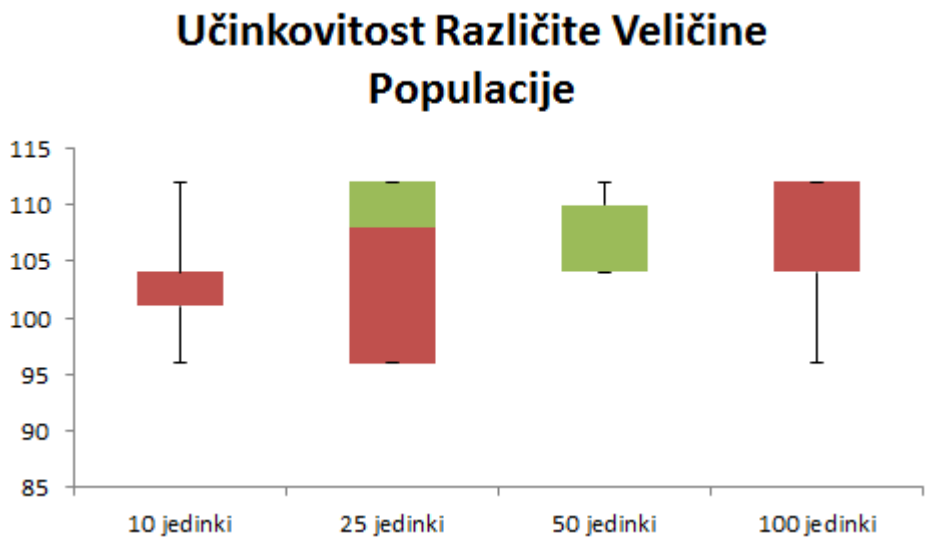


Slika 6.1.5. Učinkovitost mutacije jedne funkcije

	P = 1%	P = 10%	P = 20%	P = 40%	P = 60%	P = 80%
Min	96	96	96	96	96	96
Q1	101	104	96	104	104	98
Median	104	104	104	104	104	108
Q3	111	107	112	110	112	112
Max	112	112	112	112	112	112

Tablica 6.1.5. Rezultati ispitivanja mutacije jedne funkcije

Na slici i tablici 6.1.5. Prikazani su rezultati ispitivanja mutacije jedne funkcije. Može se vidjeti da, za razliku od mutacije jedne točke, su postignuti rezultati bolji s većim parametrom mutacije.



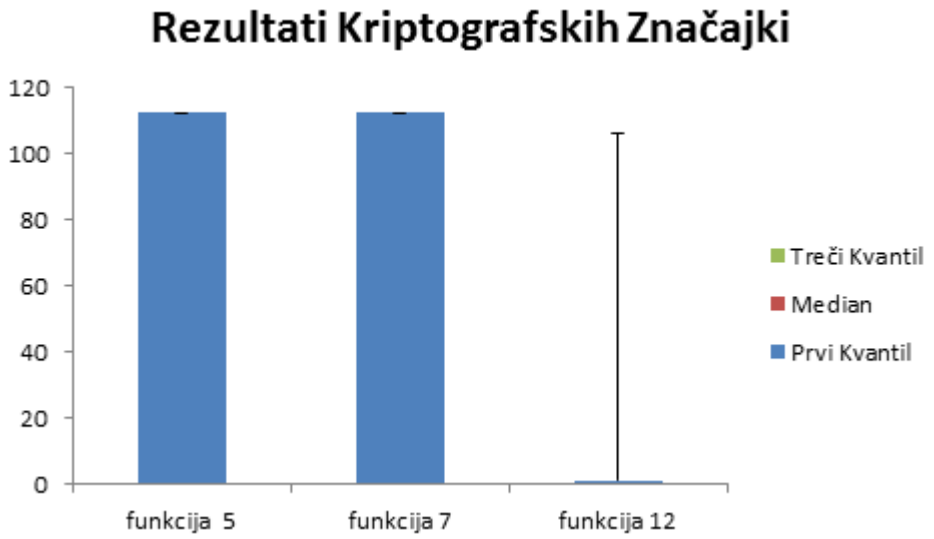
Slika 6.1.6. Učinkovitost različitih veličina populacije.

	10 jedinki	**25 jedinki**	**50 jedinki**	**100 jedinki**
Min	96	96	104	96
Q1	101	96	104	104
Median	104	108	104	112
Q3	104	112	110	112
Max	112	112	112	112

Tablica 6.1.6. rezultati ispitivanja različitih veličina populacije.

Na slici i tablici 6.1.6. prikazani su rezultati ispitivanja iz kojih se može vidjeti da su najbolji rezultati postignuti kada je bila veličina populacije najveća.

6.2. Rezultati



Slika 6.2.1. Rezultati kriptografskih značajki

	funkcija 5	**funkcija 7**	**funkcija 12**
Min	112	112	0
Q1	112	112	1
Median	112	112	1
Q3	112	112	1
Max	112	112	106

Tablica 6.2.1.

Na slici i tablici 6.2.1. prikazani su rezultati ispitivanja kriptografskih značajki. Vidi se da su na funkcijama 5 i 7 dobiveni rezultati isti kao i najbolja pokretana tijekom ispitivanja dok su za funkciju 12 rezultati bili malo lošiji u odnosu na ostale funkcije u najboljem slučaju, a u prosječnom slučaju rezultati su bili lošiji.

7. Upute za korištenje

Kako bi mogao koristiti kartezijsko genetsko programiranje korisnik treba implementirati evaluacijsku klasu i dati konfiguracijske parametre algoritmu.

7.1. Parametri

```
<ECF>
    <Algorithm>
        <SteadyStateTournament>
            <Entry key="tsize">3</Entry>
        </SteadyStateTournament>
    </Algorithm>
    <Genotype>
        <Cartesian>
            <Entry key="type">double</Entry>
            <Entry key="numvariables">2</Entry>
            <Entry key="numoutputs">1</Entry>
            <Entry key="numinputconns">2</Entry>
            <Entry key="numrows">5</Entry>
            <Entry key="numcols">3</Entry>
            <Entry key="levelsback">3</Entry>
            <Entry key="functionset">+ - * / const 2 5</Entry>
            <Entry key="mut.onepoint.prob">0.2</Entry>
            <Entry key="mut.onegate.prob">0.2</Entry>
            <Entry key="crx.gateOnePoint">1.0</Entry>
            <Entry key="IntervalArithmetic">true</Entry>
        </Cartesian>
    </Genotype>
    <Registry>
        <Entry key="population.size">50</Entry>
        <Entry key="term.maxgen">10000000</Entry>
        <Entry key="term.eval">100000</Entry>
        <Entry key="log.level">3</Entry>
        <Entry key="log.frequency">1</Entry>
        <Entry key="log.filename">log.txt</Entry>
    </Registry>
</ECF>
```

Slika 7.1.1. Podržani parametri

Na slici 7.1.1. je prikazana datoteka s svim parametrima potrebnim za pokretanje programa. Kartezijski genom ima svoje parametre koji su objašnjeni u poglavlju o

parametrima. Ovdje će bi dane samo neke upute o korištenju. Broj varijabli je usko povezan s implementacijom i tijekom evolucije svakoj jedinki moraju biti poslane sve varijable koje očekuje. Koeficijent povezanosti unatrag mora biti jednak ili manji od broja stupaca. Suma vjerojatnosti svih križanja mora biti jednaka jedan. Funkcije mogu iza sebe imati broj koji govori koliko će ta jednostavna funkcija imati ulaza, a taj broj mora biti manji ili jednak parametru numinputcons. Konstante iza sebe mora imati brojeve koji će biti njihove vrijednosti. Agf mora imati broj koliko će različitih funkcija biti dopušteno generirati. Ako se koriste agf-ovi moraju biti definirani dodatni kartezijski genotipi koji će definirati agf. Zadnji dio parametra opisuju dodatne podatke o načinu zapisa rezultata, to je nepromijenjeno u odnosu na izvorni ECF.

7.2. Implementacija evaluacijskog razreda

```
#include <cmath>
#include "ecf/ECF.h"
#include "ecf/FitnessMin.h"
#include "SymbRegEvalOp.h"


bool SymbRegEvalOp::initialize(StateP state)
{

void SymbRegEvalOp::registerParameters(StateP state)
{

FitnessP SymbRegEvalOp::evaluate(IndividualP individual)
{
    FitnessP fitness (new FitnessMin);
    cart::Cartesian* cartesian = (cart::Cartesian*) individual->getGenotype().get();

    std::vector<double> result = std::vector<double>();
    std::vector<double> inputs = std::vector<double>();

    //postavljanje ulaza

    cartesian->evaluate(inputs, &result);

    //računanje dobrote

    fitness->setValue(outFitness);

    return fitness;
}
```

Slika 7.2.1. Implementacija klase

Na slici 7.2.1. je prikazan primjer razreda za evaluaciju koji korisnik mora napisati. Razred mora uključiti glavnu ECF-ovu klasu i klasu FitnessMin ili FitnessMax ovisno o tome želimo li da rezultat bude što veći ili manji. Također mora nasljeđivati ECF-ovu klasu EvaluateOp. Ta klasu mora implementirati tri funkcije.

- Initialize: funkcija koja se poziva samo jednom tijekom kreiranja genoma. U njoj se ne mora napraviti ništa ali je pogodna za kreiranje svih primjera na kojima će se izvoditi evaluacija.
- RegisterParametars: funkcija u kojoj trebaju biti inicijalizirani svi dodatni parametri koje je korisnik sam definirao.
- Evaluate: Funkcija se poziva za svaku jedinku na početku moraju biti definirani genotip i tip dobrote te dva vektora jedan za ulaze drugi za izlaze jedinke. Nakon toga korisnik treba postaviti ulaze u jedan vektor i pozvati na jedinki operaciju evaluate. Na kraju kad korisnik izračuna dobrotu jedinke mora postaviti njenu vrijednost u objekt dobrote i vratiti je.

8. Zaključak

Kartezijsko genetsko programiranje se pokazalo dobrim za neke probleme i lošim za neke problem, kako što je očekivano zbog no free lunch teorema.

Na problemu simboličke regresije postignuti su jako dobri rezultati te na nekim funkcijama i najbolji mogući rezultati.

Na problemu logičkih funkcija postignuti su ne dovoljno dobri rezultati. Razlog toga je vrsta problema koji je bio kombinatorni po svojoj naravi a, kartezijski genetski algoritam nije osmišljen za kombinatorne probleme. Što je vidljivo iz funkcije 12 gdje u većini slučajeva nije uspio ostvariti uvjete potrebne za dodatnu parametre. Za takve probleme preporuča se genetska permutacija brojeve koja je osmišljena za kombinatorne probleme.

9. Literatura

1. Kontrec, M. Ocjena učinkovitosti operatora križanja u genetskom programiranju, Diplomski rad, Fakultet Elektrotehnike i Računarstva, 2014.
2. Šantl, D. Simbolička regresija, Diplomski seminar, Fakultet Elektrotehnike i Računarstva, 2013.
3. Picek, S. Jakobović, D. Golub, M; Evolving Cryptographically Sound Boolean Functions, GECCO13, Amsterdam, 2013
4. Keijzer, M. Improving Symbolic Regression with Interval Arithematic and Linear Scaling, Free University Amsterdam, 2003
5. Picek S; bool1.c; GPboolean; svn://smaug.zemris.fer.hr/ecfapp/GPboolean; 05.04.2014.
6. Budin L. Golub M. Jakobovic D. Jelenkovic L.; Operacijski sustavi; 1. Izdanje; ELEMENT d.o.o.; Zagreb 2010.
7. B Schneier, J. Wiley & Sons; Applied Cryptography 2nd eddition: protocols, algorithms, and source code in C; John Wiley & Sons, Inc. New York, NY, USA ©1995
8. Brajer I.; Automatsko oblikovanje kombinacijskih mreža; Diplomski rad; Fakultet Elektrotehnike i Računarstva; 2011.
9. Jović D.; Optimizacija u Kriptografiji, Diplomski seminar, Fakultet Elektrotehnike i Računarstva. 2014.
10. Balanced Boolian function, 20.03.2013, http://en.wikipedia.org/wiki/Balanced_boolean_function, 22.05.2015
11. Rodier F. On the nonlinearity of boolean functions, Institut de Math´ematiques de Luminy – C.N.R.S.
12. Correlation Imunity, 30.10.2014, http://en.wikipedia.org/wiki/Correlation_immunity, 22.05.2015
13. Zhang X.M; Pieprzyk J; Zheng Y. On Algebraic Immunity and Annihilitators, Centre for Advanced Computing – Algorithams and Cryptografy Departmant of Computing, Macquaire University Sydney, 2006

14. McLaughlin J; Clark J. A. Evolving balanced Boolean functions with optimal resistance to algebraic and fast algebraic attacks, maximal algebraic degree, and very high nonlinearity, 2013
15. McCay M. E; Butler J.T; Stanica P. Computing Algebraic Immunity by Reconfigurable Computer, Department of Electrical and Computer Engineering Department of Electrical and Computer Engineering Department of Applied Mathematics Naval Postgraduate School, Department of Applied Mathematics Naval Postgraduate School, 2011
16. Interval Arithemtic, 21.06.2015, https://en.wikipedia.org/wiki/Interval_arithmetic 27.06.2015
17. Mean Square Error 22.06.2015 https://en.wikipedia.org/wiki/Mean_squared_error 27.06.2015

10. Sažetak

U ovom radu je objašnjeno kako radi kartezijsko genetsko programiranje, i objašnjeno je korištenje kartezijskog genetsko programiranja u okviru za evolucijsko računanje. Obrađena je evolucija na primjerima simboličke regresije i logičkih funkcija u kriptografiji. Objašnjene su značajke logičkih funkcija koje želimo i danu su rezultatu provedenih testiranja. Uz rad su priloženi tablice i grafovi svih rezultata te je na kraju dan popis literature i izvorni kod.

10.1. Ključne riječi

Evolucijski, genetski, Kartezijski, algoritmi, simbolička regresija, logičke funkcije, linearno skaliranje, intervalna aritmetika, kriptografija.

11. Abstract

This paper explains how the Cartesian genetic programming works, and discussed the use of Cartesian genetic programming in the context of evolutionary computation. Evolution is tested on examples of symbolic regression and boolian functions in cryptography. Explains the features of boolian functions that we want in cryptography and gives result of tests. With paper are presented tables and graphs of the results at the end paper is list of references and source code.

11.1. Key words

Evolutionary, genetic, Cartesian, algorithms, symbolic regression, Boolean functions, linear scaling, interval arithmetic, cryptography.