SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 3853

Optimizacija Booleovih funkcija za kriptografske postupke

Marina Krček

Zagreb, lipanj 2015.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA ZAVRŠNI RAD MODULA

Zagreb, 8. ožujka 2015.

ZAVRŠNI ZADATAK br. 3853

Pristupnik: **Marina Krček (0036470282)**
Studij: Računarstvo
Modul: Računarska znanost

Zadatak: **Optimizacija Booleovih funkcija za kriptografske postupke**

Opis zadatka:

Opisati problem konstrukcije Booleovih funkcija pogodnih za kriptografsku uporabu kao optimizacijski problem. Istražiti mogućnosti različitih prikaza Booleovih funkcija u postupcima optimizacije. Razviti programski sustav za optimizaciju Booleovih funkcija primjenom različitih evolucijskih algoritama. Usporediti učinkovitost ostvarenih algoritama s obzirom na zadane kriterije optimizacije. Radu priložiti izvorne tekstove programa, dobivene rezultate uz potrebna objašnjenja i korištenu literaturu.

Zadatak uručen pristupniku: 13. ožujka 2015.
Rok za predaju rada: 12. lipnja 2015.

Mentor:
Izv. prof. dr. sc. Domagoj Jakobović

Djelovođa:
Doc. dr. sc. Tomislav Hrkać

Predsjednik odbora za završni rad modula:
Prof. dr. sc. Siniša Srbljić

Zahvala

Ovaj rad izrađen je pod vodstvom mentora prof. dr. sc. Domagoja Jakobovića.

Zahvaljujem mentoru na uloženom vremenu i trudu, sugestijama, prijedlozima i svoj ostaloj pomoći koju je pružio pri izradi ovog rada.

Sadržaj

1. Uvod

Booleove funkcije imaju veliku ulogu u kriptografiji, te trebaju zadovoljavati određena svojstva kako bi bile pogodne za upotrebu u kriptografskim postupcima. Booleove funkcije treba optimirati, tj. pronaći one funkcije koje najbolje zadovoljavaju određena svojstva. Za optimizaciju funkcija koriste se različiti evolucijski algoritmi. Evolucijski algoritmi najčešće koriste vektore kontinuiranih ili cjelobrojnih varijabli. Iz tog razloga željeli bismo Booleove funkcije prikazati takvim vektorima. U ovom radu istražit će se način interpretacije navedenih vektora kao tablice istinitosti, tj. kao interpretacije Booleovih funkcija te će se kroz ispitivanja proučiti dobiveni rezultati.

U sljedećim poglavljima objašnjen je problem rada i ideja njegovog rješenja te je opisano programsko rješenje. Nadalje, prikazan je pregled i analiza dobivenih rezultata te zaključak koji je donesen iz njih. Brojnim ispitivanjima željelo se pronaći parametre kojima će se postići najbolji rezultati.

2. Optimizacija Booleovih funkcija za kriptografske postupke

2.1 Booleove funkcije

Booleove funkcije su funkcije oblika $f : B^k \to B$, gdje je $B = \{0, 1\}$ domena i k nenegativni cijeli broj. Prikaz Booleovih funkcija je pomoću tablice istinitosti (engl. *truth table*) koja se u programima najčešće zapisuje kao niz bitova. Booleova funkcija s n ulaznih varijabli ima tablicu istinitosti od 2^n elemenata. Cilj ovog rada je isprobati različite prikaze Booleovih funkcija, točnije vektorima s kontinuiranim varijablama te vektorima s cjelobrojnim varijablama kako bi se za optimizaciju mogli koristiti brojni evolucijski algoritmi koji koriste navedene vektore.

2.2 Booleove funkcije u kriptografiji

Tri su osnovna načina za generiranje Booleovih funkcija za korištenje u kriptografiji: algebarska konstrukcija, nasumično generiranje i heuristička konstrukcija te razne kombinacije navedena tri načina.

Algebarske konstrukcije koriste matematičke metode za kreiranje Booleovih funkcija s dobrim kriptografskim svojstvima. Nasumično generiranje Booleovih funkcija jednostavna je i brza metoda, pa se zato često koristi, ali nerijetko se njome generiraju Booleove funkcije s neoptimalnim svojstvima za kriptografsku upotrebu. Heurističke metode su jednostavne i učinkovite za kreiranje velikog broja Booleovih funkcija s vrlo dobrim kriptografskim svojstvima. Evolucijski algoritmi postižu dobre rezultate u generiranju Booleovih funkcija za kriptografiju. Isprobavani su mnogi evolucijski algoritmi sa zapisom Booleovih funkcija kao tablice istinitosti u nizu bitova, a u ovom radu nastoji se prikazivati Booleove funkcije kao realne vektore i vektore cijelih brojeva kako bi bilo moguće iskoristiti što više evolucijskih algoritama koji postoje u ECF-u (engl. *Evolutionary Computation Framework*).

3. Programsko rješenje

3.1 Korištene tehnologije

Rad je podijeljen na dva dijela, pa su napravljena dva projekta stvorena u Visual Studiu s ECF bibliotekom. Prvi projekt koristi genotip koji već postoji u ECF-u – *FloatingPoint*, a u drugom projektu je korišten novi genotip – *IntGenotype*.

Projekti su pisani programskim jezikom C++ u kojem je pisan i sam ECF, ali u pisanom dijelu ovog rada prikazani su samo pseudokodovi određenih dijelova programskog rješenja radi boljeg objašnjenja. Konfiguracijske datoteke imaju strukturu XML datoteke te su jednostavne za korisnika, dijelovi konfiguracijskih datoteka su umetnuti u ovaj rad kako bi se bolje objasnio način korištenja ovog rada.

3.2 Prikaz pomoću vektora kontinuiranih varijabli

Booleove funkcije se najčešće prikazuju pomoću niza bitova koji su zapravo tablica istinitosti. Tablicu istinitosti želimo prikazati pomoću kontinuiranih varijabli. Ideja je da se koristi genotip *FloatingPoint* koji predstavlja vektor realnih varijabli. Zadatak ovog rada bio je osmisliti neki odnos između niza bitova tablice istinitosti i kontinuiranih brojeva. Ideja jednog takvog odnosa objašnjena je u ovom radu i dio je programskog rješenja.

Korisnik zadaje broj varijabli Booleove funkcije te broj varijabli u *FloatingPoint* genotipu. Iz tih podataka saznaje se veličina tablice istinitosti te koliko bitova će predstavljati jedan element, tj. broj iz vektora realnih vrijednosti. Broj elemenata tablice istinitosti jednak je $2^{broj\ varijabli\ Booleove\ funkcije}$, dok broj bitova koji predstavlja jednu vrijednost iz vektora iznosi $decode = \frac{veličina\ tablice\ istinitosti}{broj\ varijabli\ realnog\ vektora}$.

Evolucijski algoritmi koji se koriste stvaraju početnu populaciju realnih vektora, što znači da je prije slanja u neku funkciju svojstva potrebno dobivene realne vrijednosti dekodirati u bitove, tj. u polje cijelih brojeva, u ovom slučaju zapravo samo 0 i 1.

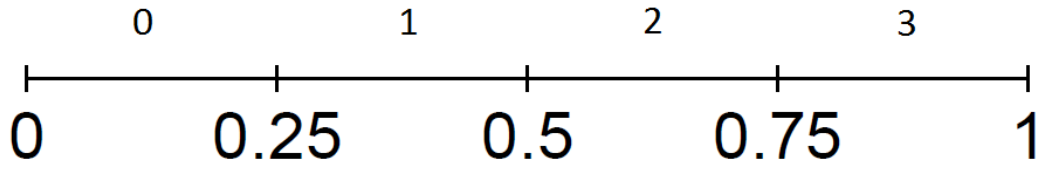
Iz realne vrijednosti prvo se dobiva cijeli broj u dekadskom sustavu koji se potom pretvara u binarni zapis. Svaka vrijednost realnog vektora pripada određenom

intervalu između 0 i 1, taj interval računa se kao $interval = \frac{1}{2^{decode}}$. Intervali predstavljaju određene cjelobrojne brojeve. U implementaciji svaki element realnog vektora podijeli se s izračunatim intervalom i uzme najmanje cijelo od rezultata $\left\lfloor \frac{vrijednost\ iz\ realnog\ vektora}{interval} \right\rfloor$, te se na taj način izračunaju cijeli brojevi koje potom treba pretvoriti u binarni zapis.

Funkcije koje opisuju svojstva Booleovih funkcija kao argumente primaju vektor, tj. polje s elementima cjelobrojnih vrijednosti, pa iz tog razloga cijele brojeve pretvaramo u binarni sustav.

Implementirane su dvije opcije dekodiranja vektora realnih brojeva – binarno i grayevo kodiranje. Opciju koju želi bira korisnik u konfiguracijskoj datoteci pod *encoding*.

Za primjer pretpostavimo da je korisnik za vektor realnih brojeva zadao donju granicu na 0, gornju na 1 i dimenziju, tj. broj varijabli na 4. Zatim neka je zadao broj Booleovih varijabli na 3, što nam daje tablicu istinitosti od $2^3 = 8$ elemenata. Iz dobivenih podataka izračunat će se da svaki element realnog vektora treba prikazati s $decode = \frac{veličina\ tablice\ istinitosti}{broj\ varijabli\ realnog\ vektora} = \frac{8}{4} = 2$ bita, te da će interval biti jednak $interval = \frac{1}{2^{decode}} = \frac{1}{2^2} = 0.25$. Dakle, za navedeni primjer vrijednosti iz realnog vektora koje su između 0 i 0.25 označavat će cijeli broj 0, vrijednosti između 0.25 i 0.5 označavat će cijeli broj 1, između 0.5 i 0.75 broj 2, te između 0.75 i 1 broj 3. Slika 3.1 prikazuje objašnjeno razmišljanje.



Slika 3.1 Vizualno objašnjenje odnosa između vektora realnih i cijelih brojeva

Funkcija koja obavlja ovu pretvorbu, tj. od primljenog vektora realnih vrijednosti stvara novi u kojem će biti odgovarajuće cjelobrojne vrijednosti u pseudokodu izgleda ovako:

```
void Binary_FP_to_INT(individual, numOfFPvar, interval) {
 dohvati vektor realnih brojeva
 za svaki element iz dohvaćenog vektora
     pom = najmanje_cijelo(vrijednost_elementa/interval)
     spremi pom u vektor cijelih brojeva
}
```

U ovom trenutku dekodiranja postoji vektor čiji su elementi diskretne vrijednosti u dekadskom sustavu koje je potrebno pretvoriti u binarni sustav. Pretvorbom u binarni sustav stvara se vektor s vrijednostima 0 i 1 koji će predstavljati tablicu istinitosti koja se šalje u funkcije svojstava za kriptografiju i time je dekodiranje završeno. Slijedi pseudokod za binarno dekodiranje.

```
vektor<int> BinaryTruthTable(vel_TT, numFPvar, dekod) {
     inicijaliziraj vektor binarno na veličinu vel_TT
     k = 0
     od zadnjeg do prvog elementa cjelobrojnog vektora {
          za svaki od 0 do dekod {
               ako element_polja % 2 == 1 {
                    podijeli element polja s 2
                    binarno[k] = 1
                    k++
               }
               inače {
                    podijeli element polja s 2
                    binarno[k] = 0
                    k++
               }
          }
          ako k == vel_TT
               izadi iz petlje
     }
     invertirati elemente vektora binarno
     vrati binarno
}
```

Grayevo kodiranje implementirano je uz pomoć binarnog jer vrijedi da ako se nad binarnim kodom posmaknutim u desno za 1 i početnim binarnim kodom (bez pomaka u desno) napravi isključivo „ili“ (engl. *xor*) dobije grayev kod, tj. vrijedi jednakost: $grayev\ kod = (binarni\ kod \gg 1)\ XOR\ binarni\ kod$. Pseudokod koji koristi navedenu jednakost prikazan je u nastavku.

```
vektor<int> GrayTruthTable(binarno, vel_TT, numFPvar,
dekod) {
     inicijaliziraj vektor gray na veličinu vel_TT
```

```
        inicijaliziraj pomoćni vektor na veličinu dekod
        za i od 0 do veličine binarno uz i+=dekod {
             za j od 0 do dekod {
                  ako i + j == dekod + i - 1
                       pomoćni[j] = 0
                  inače
                       pomoćni[j] = binarno[i + j + 1]
                  ako binarno[i + j] == pomoćni[j]
                       gray[i + j] = 0
                  inače ako bin[i + j] != pom[j]
                       gray[i + j] = 1
             }
        }
        invertirati elemente gray vektora
        vrati gray
}
```

Kako bi bilo jasnije, svi koraci pretvorbe od vektora realnih brojeva do tablice istinitosti prikazani su na konkretnom primjeru. Neka ostanu isti zadani podaci kao u prošlom primjeru, dakle realne vrijednosti generiraju se iz intervala između 0 i 1, a dimenzija vektora neka je 4. Booleova funkcija neka ima 3 varijable, dakle tablica istinitosti bit će veličine $2^3 = 8$ elemenata. Argumenti funkcija pretvorbe u tablicu istinitosti su $decode = \frac{veličina\ tablice\ istinitosti}{broj\ elemenata\ realnog\ vektora} = \frac{8}{4} = 2$ bita, i $interval = \frac{1}{2^{decode}} = \frac{1}{2^2} = 0.25$.

U populaciji neka je generiran realni vektor s vrijednostima 0.709311, 0.63918, 0.138128 i 0.481571 (redom po indeksima od 0 do 3) te će se na tom vektoru pokazati koraci objašnjenog dekodiranja. Dakle, prvi korak je pretvorba realnih vrijednosti u diskretne cjelobrojne vrijednosti dijeljenjem s intervalom te uzimanjem najmanjeg cijelog. Vektor cjelobrojnih vrijednosti za ovaj slučaj iznosit će: $\left\lfloor \frac{0.709311}{0.25} \right\rfloor = \lfloor 2.837244 \rfloor = 2$, $\left\lfloor \frac{0.63918}{0.25} \right\rfloor = \lfloor 2.55672 \rfloor = 2$, $\left\lfloor \frac{0.138128}{0.25} \right\rfloor = \lfloor 0.552512 \rfloor = 0$ i $\left\lfloor \frac{0.481571}{0.25} \right\rfloor = \lfloor 1.926284 \rfloor = 1$. Sada je potrebno napraviti tablicu istinitosti tako da se brojevi 2, 2, 0 i 1 pretvore u binarni sustav i zapišu u vektor cijelih brojeva koji će sadržavati samo brojeve 0 i 1. U binarnom kodiranju vektor tablice istinitosti će redom od indeksa 0 do indeksa 7 izgledati: 10100001. Ukoliko je zadano grayevo kodiranje, dobiveni vektor poslat će se u funkciju za dobivanje tablice istinitosti u grayevom kodiranju. Nakon te obrade tablica istinitosti će slijedno od indeksa 0 do 7 biti: 11110001. Tako stvorena

tablica istinitosti šalje se na obradu u funkcije svojstava po kojima se optimiraju Booleove funkcije za upotrebu u kriptografskim postupcima.

3.3 Prikaz pomoću vektora cjelobrojnih varijabli

Za prikaz tablice istinitosti pomoću cijelih brojeva, napravljen je novi genotip koji umjesto realne vrijednosti varijabli koje sadrži *FloatingPoint* genotip koristi cjelobrojne vrijednosti. Dakle, novi genotip nazvan *IntGenotype* je vektor s vrijednostima varijabli tipa *integer* (cijeli broj). Za novi genotip treba također odrediti broj varijabli, tj. elemenata u vektoru, te gornju i donju granicu između kojih se biraju vrijednosti koje će popuniti vektor. U *IntGenotype* uključene su jednostavne funkcije križanja i mutacije koje koriste neki evolucijski algoritmi.

Pseudokod za funkciju križanja izgleda ovako:

```
bool mate(gen1, gen2, child) {
 za i od 0 do veličine gen1 {
     a = nasumična realna vrijednost iz intervala [0,1]
     ako a manje od 0.5
          child[i] = gen1[i]
     inače
          child[i] = gen2[i]
 }
vrati true
}
```

Funkcija mutacije zapisana pseudokodom:

```
bool mutate(gen) {
 size = veličina vektora
 a = nasumičan cijeli broj iz intervala [0,size)
 d = donja granica vektora
 g = gornja granica vektora
 gen[a] = nasumičan cijeli broj iz intervala [d,g]
vrati true
}
```

U ovom slučaju, algoritmi će stvarati populaciju vektora s cjelobrojnim vrijednostima te sada nema potrebe za posebnim dekodiranjem iz realne vrijednosti u cijeli broj, nego se odmah pristupa pretvorbi u binarni sustav, što čini dekodiranje jednostavnijim od prošlog – potrebno je manje koraka do konačne tablice istinitosti. Broj bitova koji će označavati pojedinu vrijednost iz vektora cijelih brojeva računa se na isti način kao kod prikaza pomoću vektora kontinuiranih varijabli: $decode =$

$\frac{broj\ elemenata\ tablice\ istinitosti}{broj\ elemenata\ cjelobrojnog\ vektora}$. Također, postoje dva načina kodiranja, binarno i grayevo kodiranje koje odabire korisnik. Pseudokodovi za funkcije u kojima dobivene vektore cijelih brojeva iz populacije pretvaramo u tablicu istinitosti predstavljenu poljem cijelih brojeva (0 i 1 su jedine vrijednosti koje će to polje sadržavati), tj. binarni sustav jednake su kao i kod prikaza pomoću realnog vektora.

3.4 Konfiguracijska datoteka i način upotrebe programskog rješenja

Konfiguracijska datoteka služi da se njome odabere algoritam kojim će se optimirati Booleove funkcije te za zadavanje parametara odabranog algoritma, genotipa i same Booleove funkcije. Parametre odabranog algoritma korisnik mora proučiti u dokumentaciji izabranog algoritma. Ovaj rad istražuje različite prikaze Booleovih funkcija, pa su detaljnije objašnjeni parametri koji određuju Booleove funkcije i njihov prikaz.

U prvom slučaju kada koristimo *FloatingPoint* genotip treba se zadati: *lbound*, *ubound* i *dimension*. *Dimension* je broj varijabli vektora, tj. veličina vektora, a *lbound* i *ubound* su donja i gornja granica koje označavaju interval iz kojeg se odabiru realni brojevi koji će se upisati u vektor. Primjer upisivanja objašnjenih parametara u konfiguracijsku datoteku izgleda:

```
<Genotype>
     <FloatingPoint>
          <Entry key="lbound">0</Entry>
          <Entry key="ubound">1</Entry>
          <Entry key="dimension">8</Entry>
     </FloatingPoint>
</Genotype>
```

Za korištenje drugog genotipa – *IntGenotype*, zapis u konfiguracijskoj datoteci treba izgledati:

```
<Genotype>
     <IntGenotype>
          <Entry key="lbound">0</Entry>
          <Entry key="ubound">15</Entry>
          <Entry key="size">8</Entry>
     </IntGenotype>
</Genotype>
```

Jedina razlika je što se kod genotipa *IntGenotype* zadaje veličina vektora pod drugačijim nazivom – *size*, umjesto *dimesion* koji je bio kod *FloatingPoint* genotipa.

Ostali parametri koji se trebaju upisati su *bool_variables* i *encoding*. *Bool_variables* je zapravo broj varijabli Booleove funkcije, to je n u veličini tablice istinitosti koja iznosi 2^n. Kako je prije napomenuto, implementirana su dva načina kodiranja brojeva dekadskog sustava u binarni – uobičajeno binarno (težinsko) i grayevo kodiranje (Hammingova udaljenost susjednih brojeva je 1). Korisnik bira želi li binarno ili grayevo kodiranje tako da u konfiguracijskoj datoteci pod *encoding* upiše ili *binary* ili *gray*. Sljedećim odsječkom korisnik je odabrao Booleovu funkciju s 8 varijabli, te binarno kodiranje tablice istinitosti.

```
<Registry>
     <Entry key="bool_variables">8</Entry>
     <Entry key="encoding">binary</Entry>
</Registry>
```

S obzirom da želimo optimirati Booleove funkcije za potrebe u kriptografiji, one trebaju zadovoljiti neka svojstva koja su programski ostvarena, te će se koristiti u ovom radu. Kako bi se navedeno programsko ostvarenje moglo koristiti potrebno ga je uključiti u projekte ovoga rada. Korišteno ostvarenje zapravo je niz funkcija svojstava te se određena funkcija bira putem konfiguracijske datoteke pod *function*.

```
<Registry>
     <Entry key="function">5</Entry>
</Registry>
```

4. Rezultati

4.1 Jednostavni primjeri

Programsko rješenje prvo je ispitano na vrlo jednostavnim primjerima radi provjere programskog rješenja i saznanja o tome kako zadati što bolje parametre za teže optimizacijske probleme. Jednostavni primjer je zapravo bilo zadavanje određene Booleove funkcije te pronalaženje zadane funkcije zapisane u *FloatingPoint* ili *IntGenotype* genotipu koja će imati najmanju razliku u bitovima.

Genotipom *FloatingPoint* isprobani su algoritmi *SteadyStateTournament* (SST), *RouletteWheel* (RW), *GenHookeJeeves* (GHJ), *OptIA* (OIA), *Clonalg* (CA), *DifferentialEvolution* (DE), *ArtificialBeeColony* (ABC) i *ParticleSwarmOptimization* (PSO).

Za mali broj varijabli, na primjer 4 varijable realnog vektora i 4 Booleove varijable, svi algoritmi odmah nađu zadanu funkciju, osim *ArtificialBeeColony* i *ParticleSwarmOptimization* koji u 500 000 evaluacija uspiju naći Booleovu funkciju s razlikom u jednom ili dva bita. Jednako je koristili binarno ili grayevo kodiranje.

Recimo da je u kodu zadana Booleova funkcija s tablicom istinitosti: $[1011101000100111]$ što je u prikazu polja s cijelim brojevima jednako - $[11\ 10\ 2\ 7]$ jer koristimo binarno kodiranje. U ovom primjeru jedan cijeli broj predstavljen je s 4 bita tablici istinitosti. Jedno od mogućih rješenja koje je dobiveno ispitivanjem je vektor $[0.716098\ 0.648802\ 0.132136\ 0.499484]$. Provjerimo: $interval = \frac{1}{2^{decode}} = \frac{1}{2^4} = 0.0625$, dakle ovaj vektor realnih brojeva pretvaramo u vektor s cjelobrojnim vrijednostima i dobijemo $\left[\left\lfloor\frac{0.716098}{0.0625}\right\rfloor \left\lfloor\frac{0.648802}{0.0625}\right\rfloor \left\lfloor\frac{0.132136}{0.0625}\right\rfloor \left\lfloor\frac{0.499484}{0.0625}\right\rfloor\right] = [11\ 10\ 2\ 7]$ što je upravo tražena Booleova funkcija.

Na primjeru s malo varijabli nisu se vidjele razlike, pa je sljedeći korak u ispitivanju bio povećanje broja varijabli, tj. veličine vektora koji se optimiraju.

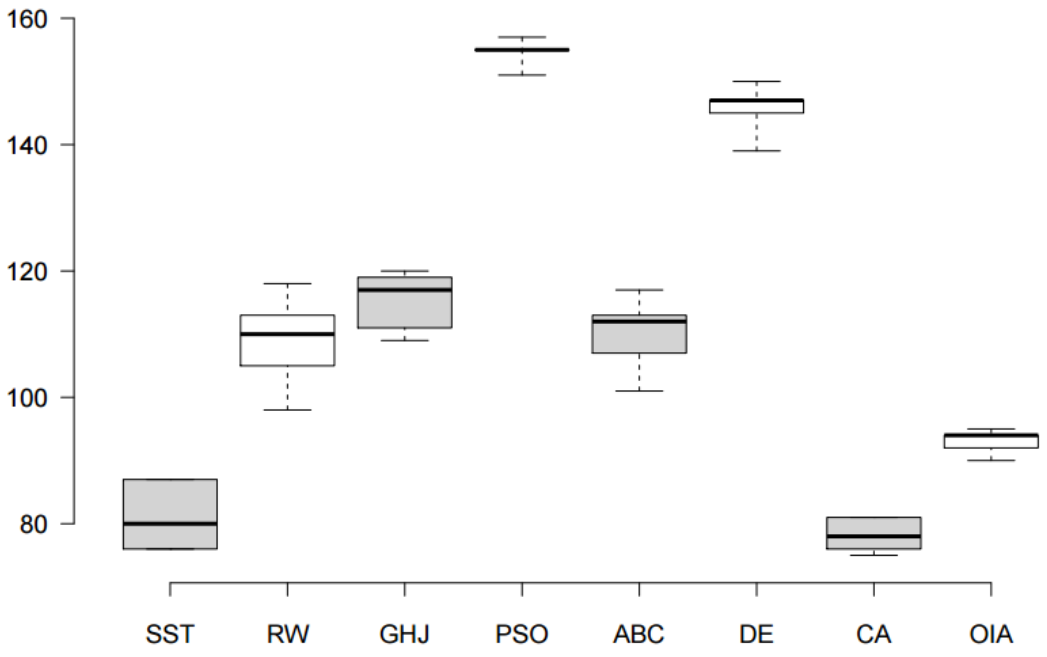
Zadali smo broj Booleovih varijabli na 8, te veličinu realnog vektora na 16 kako bi mogli rezultate usporediti s genotipom *IntGenotype*, što znači da u tom slučaju 16 bitova odgovara jednoj varijabli, tj. jednoj vrijednosti iz realnog vektora. Veličina

populacije je 500, a uvjet zaustavljanja 500 000. Svako ispitivanje ponovljeno je 5 puta. Rezultati u ovom primjeru predstavljaju broj bitova u kojima se pronađena funkcija razlikuje od zadane. Binarnim kodiranjem dobili smo rezultate zapisane u tablici 4.1. U ovoj i svim sljedećim tablicama prikazane su kratice algoritama, te oznake *min* (engl. *minimum*) kao najmanje, *avg* (engl. *average*) kao prosječne i *max* (engl. *maximum*) kao najveće dobivene vrijednosti, radi preglednosti tablice i samih rezultata. Također, kratice algoritama koriste se i na vizualnim prikazima rezultata.

Tablica 4.1 Rezultati za 8 Booleovih i 16 realnih varijabli

FP,binary	SST	RW	GHJ	PSO	ABC	DE	CA	OIA
min	76	98	109	151	101	139	75	90
avg	81.2	108.8	115.2	154.6	110	145.6	78.2	93
max	87	118	120	157	117	150	81	95

Iz tablice 4.1 i slike 4.1 vidi se da su algoritmi *Clonalg* i *SteadyStateTournament* pronašli najbolje rezultate dok su algoritmi *ParticleSwarmOptimization* i *DifferentialEvolution* bili najlošiji.



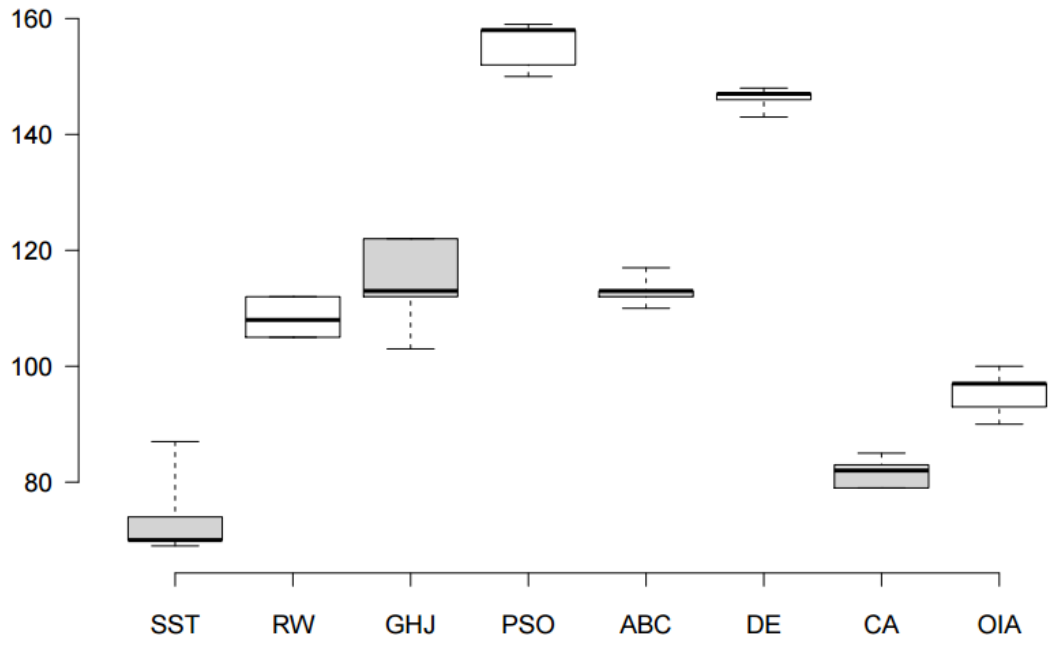
Slika 4.1 Rezultati za 8 Booleovih i 16 realnih varijabli dobiveni binarnim kodiranjem

Tablica 4.2 prikazuje rezultate dobivene s istim brojem varijabli, ali grayevim kodiranjem. Za algoritme *SteadyStateTournament*, *RouletteWheel* i *GenHookeJeeves* grayevo kodiranje pokazalo se bolje nego binarno, dok je za ostale algoritme binarno kodiranje bio bolji odabir.

Tablica 4.2 Rezultati za 8 Booleovih i 16 realnih varijabli

FP, gray	SST	RW	GHJ	PSO	ABC	DE	CA	OIA
min	69	105	103	152	110	143	79	90
avg	74	108.4	114.4	155.4	113	146.2	81.6	95.4
max	87	112	122	159	117	148	85	100

Sa slike 4.2 uočavamo da su i grayevim kodiranjem algoritmi *Clonalg* i *SteadyStateTournament* pronašli bolje rezultate od ostalih, te su algoritmi *ParticleSwarmOptimization* i *DifferentialEvolution* opet bili najlošiji.



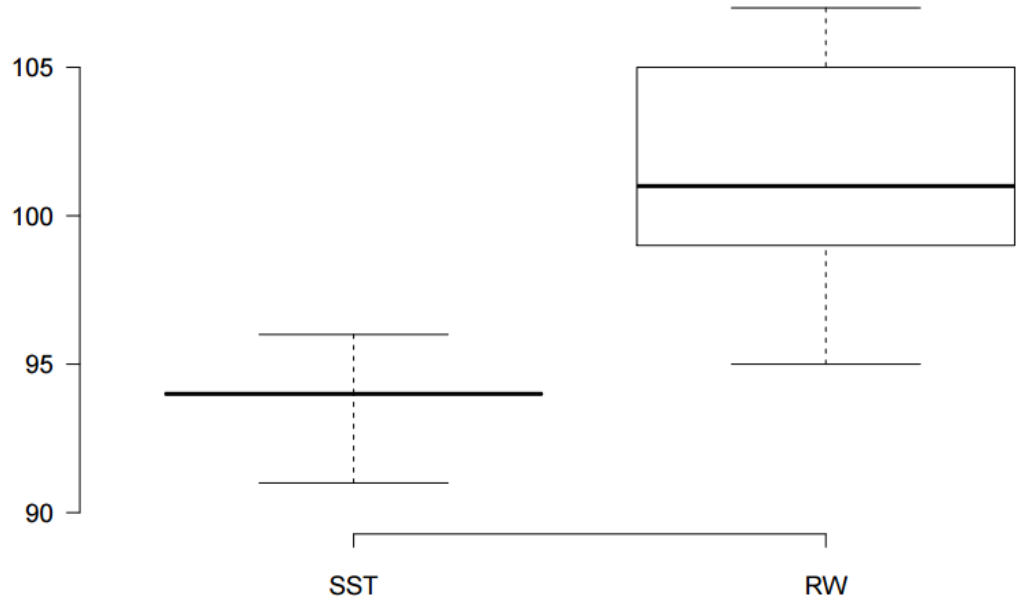
Slika 4.2 Rezultati za 8 Booleovih i 16 realnih varijabli dobiveni grayevim kodiranjem

IntGenotype je novi genotip i ne mogu se ispitati svi navedeni algoritmi koje smo isprobavali s podržanim *FloatingPoint* genotipom. No, *SteadyStateTournament* (SST) i *RouletteWheel* (RW) ne ovise o tipu genotipa, pa se ta dva algoritma ispituju s vektorom cijelih brojeva. Broj varijabli zadaje se kao i kod ispitivanja s *FloatingPoint* genotipom, dakle broj Booleovih varijabli stavljen je na 8, dok je veličina vektora cijelih brojeva bila 16, što je ista veličina koje je bio i realni vektor.

Tablica 4.3 prikazuje rezultate s binarnim kodiranjem gdje uočavamo da se i kod *IntGenotype* genotipa *SteadyStateTournament* pokazao bolji od algoritma *RouletteWheel*. No, uspoređujući rezultate vidimo da algoritmi koristeći *FloatingPoint* genotip pronalaze bolje funkcije, tj. funkcije s manjom razlikom u bitovima od zadane Booleove funkcije. Slika 4.3 vizualizira dobivene rezultate iz tablice 4.3.

Tablica 4.3 Rezultati za 8 Booleovih i 16 cjelobrojnih varijabli

INT,binary	**SST**	**RW**
min	91	95
avg	93.8	101.4
max	96	107

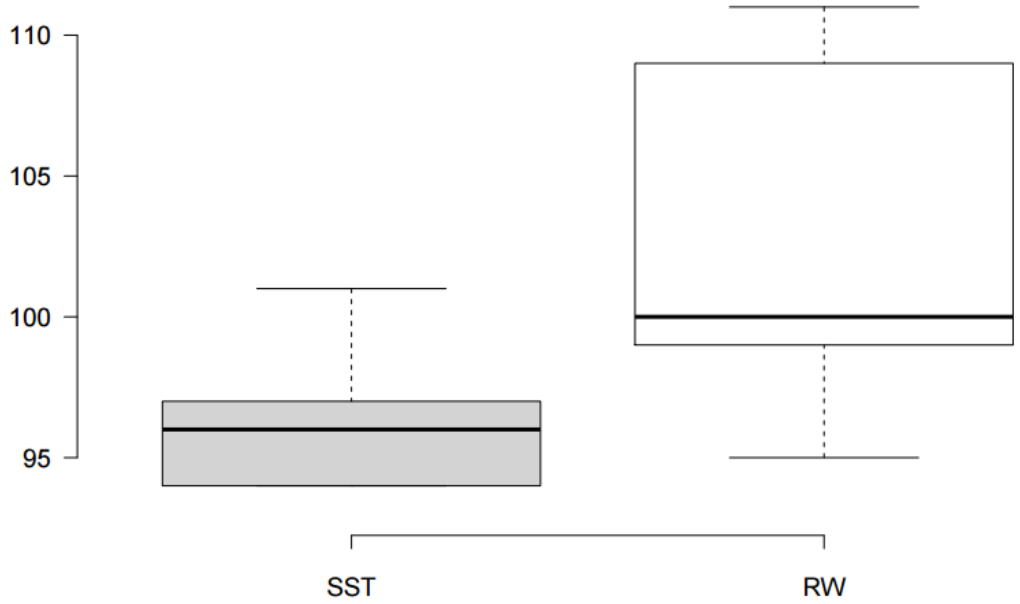


Slika 4.3 Vizualizirani rezultati za 8 Booleovih i 16 cjelobrojnih varijabli uz binarno kodiranje

Isprobano je i grayevo kodiranje te su rezultati zapisani u tablici 4.4 i vizualizirani slikom 4.4. Iz rezultata se vidi da je u oba algoritma binarno kodiranje pronašlo bolje funkcije, što je dakle za ova dva algoritma u slučaju s *FloatingPoint* genotipom obrnuto, gdje je grayevo kodiranje pokazalo bolje rezultate. Također, opet se potvrđuje da su rezultati s *IntGenotype* genotipom lošiji od onih s *FloatingPoint* genotipom.

Tablica 4.4 Rezultati za 8 Booleovih i 16 cjelobrojnih varijabli

INT,gray	SST	RW
min	94	95
avg	96.4	102.8
max	101	111



Slika 4.4 Vizualizirani rezultati za 8 Booleovih i 16 cjelobrojnih varijabli uz grayevo kodiranje

4.2 Svojstva Booleovih funkcija

Neka od svojstava Booleovih funkcija koja su ispitana za provjeru pomoću programskog ostvarenja koje koristimo u radu su balansiranost, korelacijski imunitet, algebarski stupanj i imunitet, nelinearnost i drugi. Svojstva ispitana u ovom radu su nelinearnost, karakteristika propagacije i funkcija koja kombinira korelacijski imunitet, algebarski stupanj i balansiranost. Nelinearnost se izražava kao najmanja Hammingova udaljenost između Booleove funkcije i najbolje afine aproksimacije funkcije. Hammingova udaljenost između dva niza bitova jednake duljine je broj bitova u kojima se dva niza razlikuju. Afina funkcija je funkcija oblika

$$f(x) = Ax + b.$$

Dakle, afine funkcije su zapravo pravci. Tangenta je primjer afine funkcije koja aproksimira funkciju čija je tangenta u točki u kojoj se dodiruju.

Gornja granica za svojstvo nelinearnosti određena je sljedećim izrazom:

$$N_f \leq 2^{n-1} - 2^{\frac{n}{2}-1},$$

gdje je n broj varijabli Booleove funkcije. Za ovo svojstvo dobrota (engl. *fitness*) bit će:

$$fitness = N_f.$$

Svojstvo nelinearnosti u ovom radu označavati ćemo s *f1*, radi lakšeg snalaženja u prikazu rezultata.

Karakteristiku propagacije označavat ćemo kao *f2*. Ovim svojstvom traži se balansirana funkcija s najvećom mogućom nelinearnošću i korelacijskim imunitetom jednakim nuli. Izraz izgleda:

$$fitness = BAL + N_f,$$

gdje BAL predstavlja stupanj balansiranosti funkcije.

Dobrota za svojstvo koje kombinira korelacijski imunitet, algebarski stupanj i balansiranost bit će označavana kao *f3* u ovom radu i izgleda:

$$fitness = BAL + t + \delta_{t,2}(N_f + AI + d),$$

gdje je t stupanj korelacijskog imuniteta za koji vrijedi

$$t \leq n - d - 1,$$

ako je funkcija balansirana, a ako nije onda na desnoj strani ove nejednakosti stoji samo $n - d$. S time da d predstavlja algebarski stupanj, a n broj ulaza.

Algebarski imunitet ograničen je odozgo brojem ulaznih varijabli, točnije:

$$AI \leq \left\lceil \frac{n}{2} \right\rceil,$$

a $\delta_{t,2}$ znači da se izraz u zagradi dodaje samo ako je $t = 2$.

4.3 Rezultati s vektorom realnih brojeva

Prije navedeni algoritmi isprobani su s *FloatingPoint* genotipom na funkcijama svojstava nelinearnosti, karakteristike propagacije i funkciji koja kombinira korelacijski imunitet, algebarski stupanj i balansiranost. Za te funkcije traže se najveće vrijednosti (engl. *maximum*) za razliku od prijašnjih jednostavnijih primjera gdje se tražila najmanja razlika u bitovima (engl. *minimum*).

Za sva ispitivanja veličina populacije je 500, a uvjet zaustavljanja postavljen je na 500 000 evaluacija.

Rezultati provedenih ispitivanja prikazani su u jednakim tablicama kao i kod rezultata za jednostavne primjere, dakle uvedene su kratice algoritama te su ispisane najveća, najmanja i prosječna dobivena vrijednost u 5 ponavljanja.

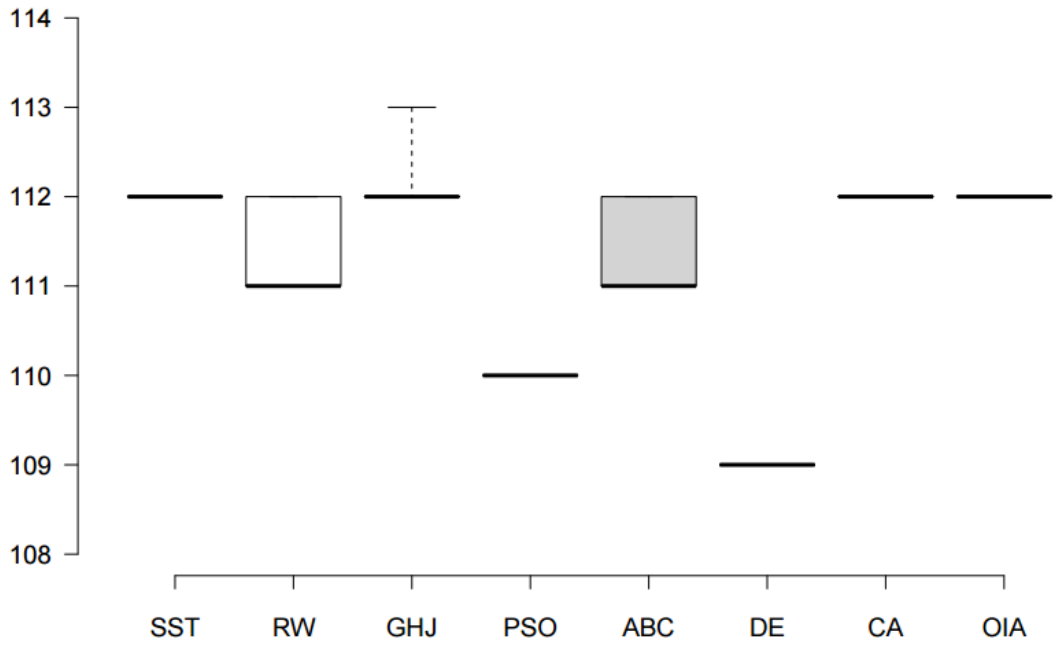
Prva ispitana funkcija je funkcija *f1* (svojstvo nelinearnosti), dobivene rezultate prikazuju tablica 4.5 i tablica 4.6. U konfiguracijskoj datoteci zadan je broj varijabli Booleove funkcije na 8, te broj elemenata realnog vektora također na 8. Interval za odabir vrijednosti realnog vektora je $[0, 1]$. Radi usporedbe ispitana su oba načina kodiranja – binarno i grayevo. U tablicama je naznačeno koje kodiranje je korišteno, te funkcija čije rezultate prikazuje.

Za svojstvo nelinearnosti – funkciju *f1* najveću vrijednost 113 dobili su algoritmi *GenHookeJeeves* i *Clonalg*, ali različitim kodiranjem. Za algoritme *ParticleSwarmOptimization* i *Clonalg* grayevim kodiranjem pronađena je bolja funkcija, dok je ostalim algoritmima binarnim kodiranjem postignuto bolje ili jednako rješenje.

Tablica 4.5 Rezultati za 8 Booleovih i 8 realnih varijabli

f1,binary	SST	RW	GHJ	PSO	ABC	DE	CA	OIA
min	112	111	112	110	111	109	112	112
avg	112	111.4	112.2	110	111.4	109	112	112
max	112	112	113	110	112	109	112	112

Slika 4.5 prikazuje rezultate funkcije *f1* s binarnim kodiranjem gdje uočavamo da su najlošije rezultate postigli algoritmi *ParticleSwarmOptimization* i *DifferentialEvolution*, dok su ostali zapravo pronašli podjednako dobre rezultate, u skladu s prijašnjim istraživanjima. Može se primijetiti da su navedena dva algoritma bila najlošija i kod ispitivanja jednostavnih primjera.



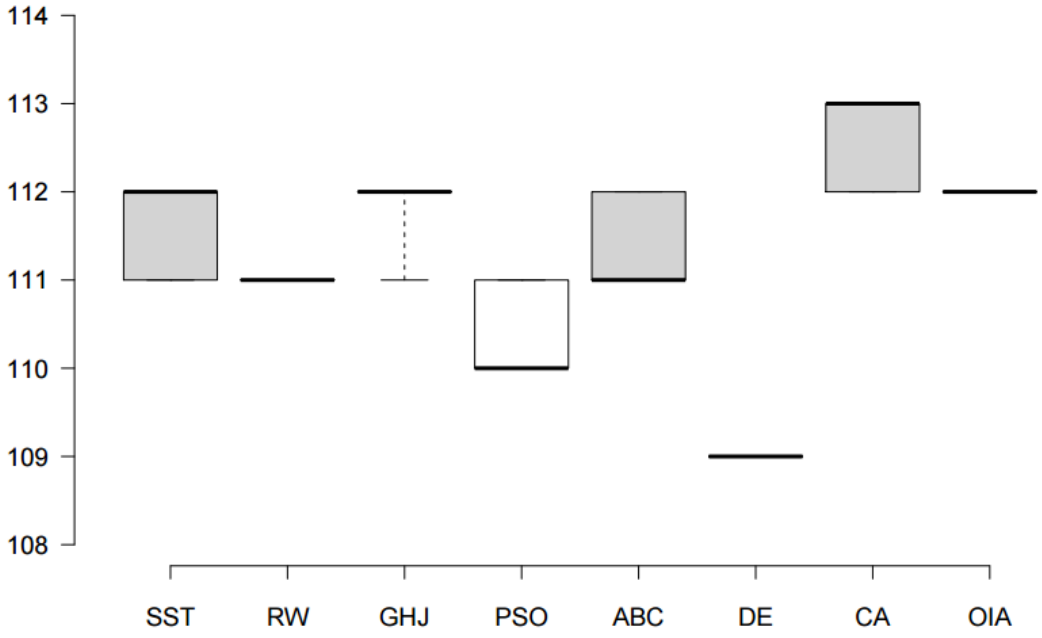
Slika 4.5 Rezultati funkcije f1 uz binarno kodiranje realnog vektora

Kod ispitivanja funkcije *f1* grayevim kodiranjem algoritam *Clonalg* pronašao je bolje rezultate te se pokazao najboljim što vidimo u tablici 4.6. Na slici 4.6 jasno se vidi da se algoritam *ParticleSwarmOptimization* grayevim kodiranjem približio rezultatima

ostalih algoritama dok je *DifferentialEvolution* dobio jednake rezultate binarnim i grayevim kodiranjem te je najlošiji.

Tablica 4.6 Rezultati za 8 Booleovih i 8 realnih varijabli

f1, gray	SST	RW	GHJ	PSO	ABC	DE	CA	OIA
min	111	111	111	110	111	109	112	112
avg	111.6	111	111.8	110.4	111.4	109	112.6	112
max	112	111	112	111	112	109	113	112

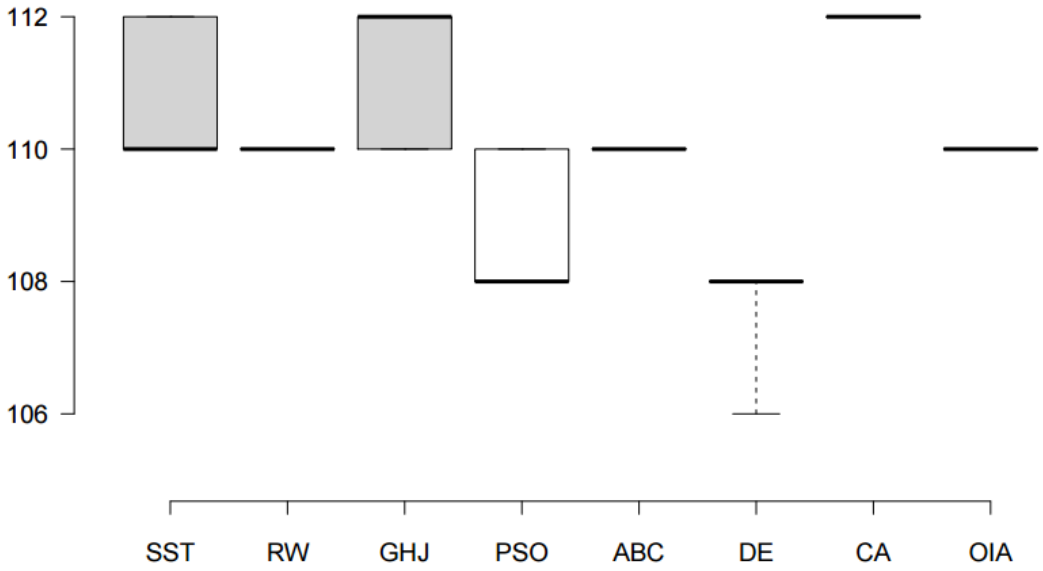


Slika 4.6 Rezultati funkcije f1 uz grayevo kodiranje realnog vektora

Za funkciju *f2* binarno kodiranje pokazalo se bolje za *GenHookeJeeves* algoritam, a lošije za algoritme *DifferentialEvolution* i *OptIA*. Ostalim algoritmima dobiveni su identični rezultati što se vidi sa slika 4.7 i 4.8. Opet se može uočiti činjenica da su *ParticleSwarmOptimization* i *DifferentialEvolution* dobivali najlošije rezultate, dok algoritam *Clonalg* pokazuje najbolje.

Tablica 4.7 Rezultati za 8 Booleovih i 8 realnih varijabli

f2,binary	SST	RW	GHJ	PSO	ABC	DE	CA	OIA
min	110	110	110	108	110	106	112	110
avg	110.8	110	111.2	108.8	110	107.6	112	110
max	112	110	112	110	110	108	112	110

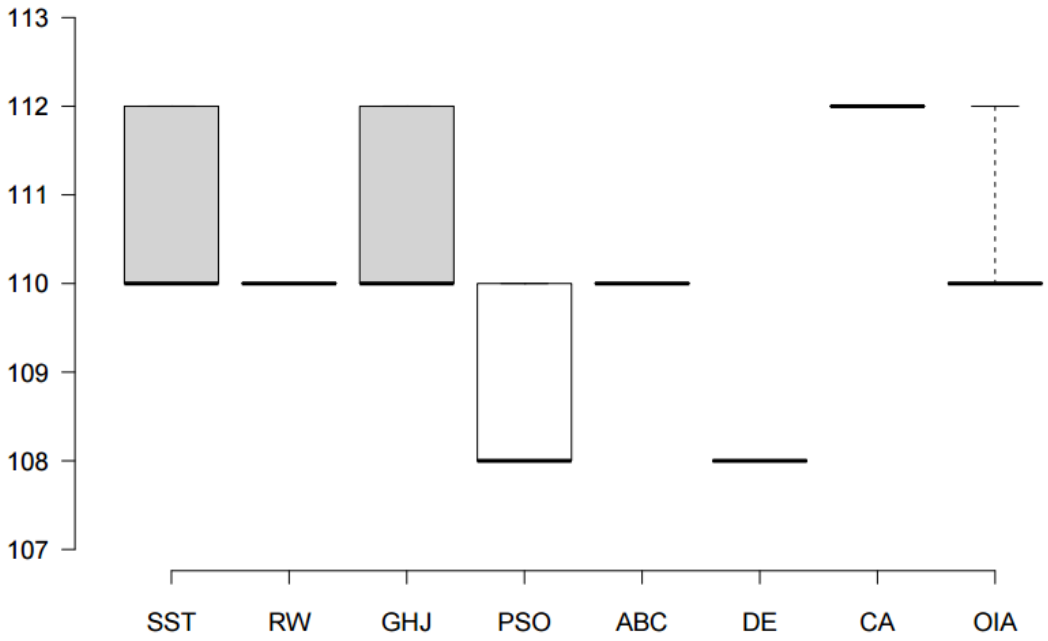


Slika 4.7 Rezultati funkcije f2 uz binarno kodiranje realnog vektora

U tablici 4.8 su zapisani rezultati ispitivanja funkcije *f2* (karakteristika propagacije) grayevim kodiranjem koje se pokazalo bolje samo za *GenHookeJeeves* algoritam.

Tablica 4.8 Rezultati za 8 Booleovih i 8 realnih varijabli

f2, gray	SST	RW	GHJ	PSO	ABC	DE	CA	OIA
min	110	110	110	108	110	108	112	110
avg	110.8	110	110.8	108.8	110	108	112	110.4
max	112	110	112	110	110	108	112	112



Slika 4.8 Rezultati funkcije f2 uz grayevo kodiranje realnog vektora

Istraživanja provedena prije ovog rada koja su za slične probleme optimirali funkciju *f3* (kombinira korelacijski imunitet, algebarski stupanj i balansiranost) evolucijskim algoritmima dobili su slične rezultate kao u ovom radu. No, ta su istraživanja pokazala da su za ovu funkciju neke druge metode optimizacije pronašle puno bolje rezultate.

U tablicama 4.9 i 4.10 uočavamo da su svi algoritmi dobili jednake rezultate i binarnim i grayevim kodiranjem.

Tablica 4.9 Rezultati za 8 Booleovih i 8 realnih varijabli

f3,binary	SST	RW	GHJ	PSO	ABC	DE	CA	OIA
min	0	0	0	0	0	0	0	0
avg	0	0	0	0	0	0	0	0
max	0	0	0	0	0	0	0	0

Kod ispitivanja moglo se primijetiti da zapravo većina algoritama pronađe svoj krajnji rezultat u prvoj generaciji, te da je problem što tijekom evaluiranja nema napretka. Grayevo kodiranje nije poboljšalo rezultate što vidimo iz tablice 4.10.

Tablica 4.10 Rezultati za 8 Booleovih i 8 realnih varijabli

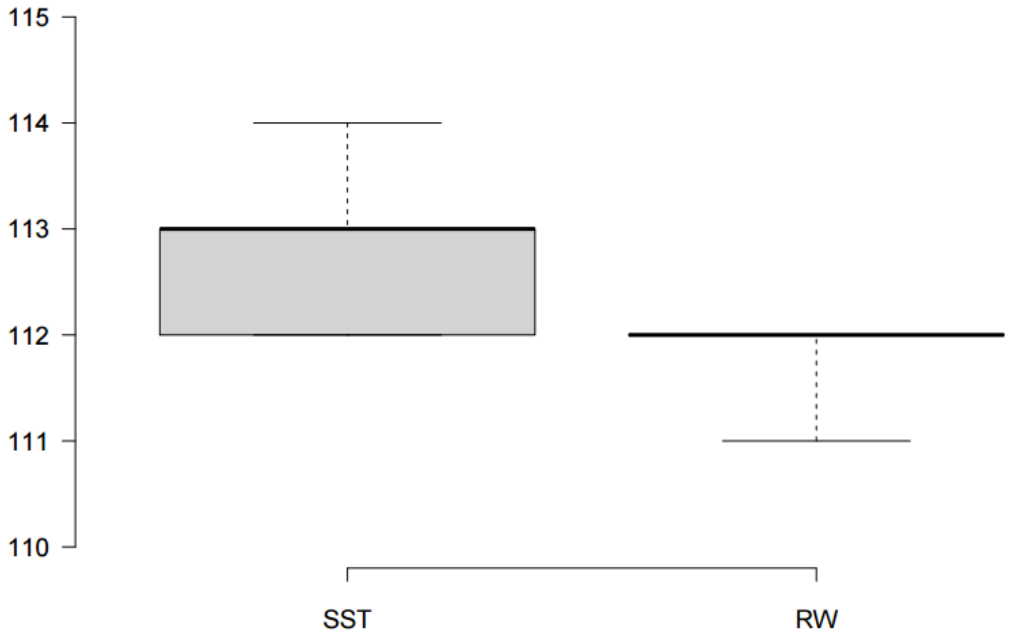
f3, gray	SST	RW	GHJ	PSO	ABC	DE	CA	OIA
min	0	0	0	0	0	0	0	0
avg	0	0	0	0	0	0	0	0
max	0	0	0	0	0	0	0	0

Pomoću vektora realnih brojeva ispitano je na algoritmima *SteadyStateTournament* i *RouletteWheel* ponašanje kada korisnik zada broj Booleovih varijabli na 8, a veličinu realnog vektora na 16, kada će manji broj bitova predstavljati jednu varijablu realnog vektora, radi usporedbe s korištenjem vektora cijelih brojeva. Interval je ostavljen isti - $[0, 1]$.

Ispitana je funkcija *f1* te su dobiveni rezultati prikazani u tablicama 4.11 i 4.12. U ovom slučaju za algoritam *SteadyStateTournament* grayevo kodiranje se pokazalo boljim odabirom, dok je za algoritam *RouletteWheel* suprotna situacija. Također, može se primijetiti da su u ovom slučaju kada imamo 8 Booleovih i 16 realnih varijabli bolji rezultati nego kada se ispitivalo s 8 Booleovih i 8 realnih varijabli. Slika 4.9 prikazuje rezultate za funkciju *f1* dobivene binarnim kodiranjem.

Tablica 4.11 Rezultati za 8 Booleovih i 16 realnih varijabli

f1,binary	SST	RW
min	112	111
avg	112.8	111.8
max	114	112



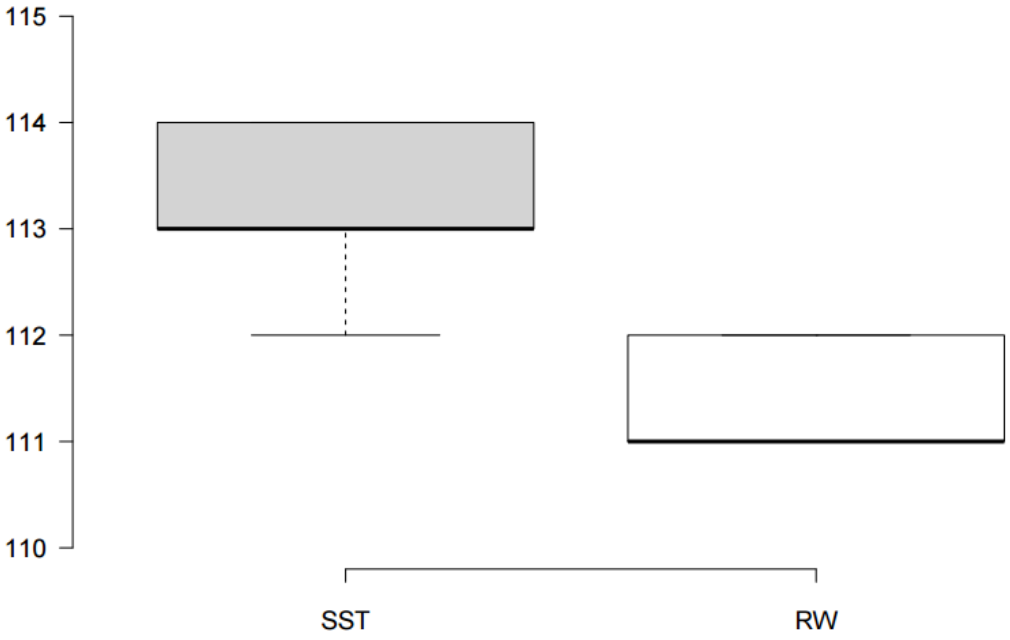
Slika 4.9 Rezultati funkcije f1 uz binarno kodiranje realnog vektora

Grayevim kodiranjem algoritam *SteadyStateTournament* je pronašao rezultate koje u prijašnjem ispitivanju s 8 Booleovih i 8 realnih vektora nije pronašao niti jedan ispitani algoritam.

Tablica 4.12 Rezultati za 8 Booleovih i 16 realnih varijabli

f1, gray	**SST**	**RW**
min	112	111
avg	113.2	111.4
max	114	112

Slika 4.10 prikazuje rezultate funkcije *f1* dobivene grayevim kodiranjem gdje jasno vidimo kako je algoritam *SteadyStateTournament* pronašao puno bolje rezultate od algoritma *RouletteWheel*.

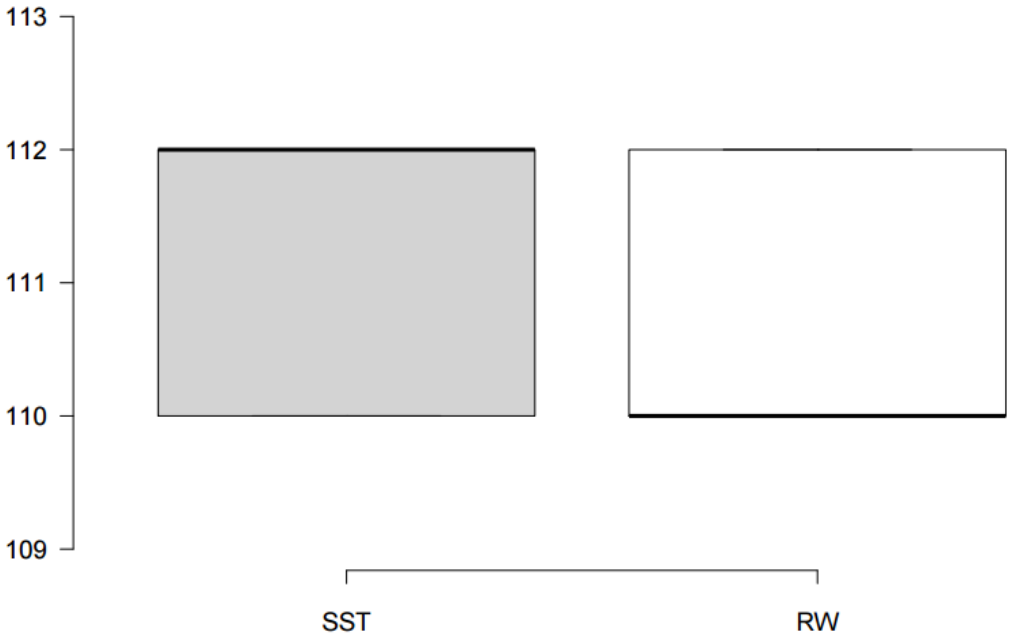


Slika 4.10 Rezultati funkcije f1 uz grayevo kodiranje realnog vektora

Kod ispitivanja funkcije *f2* oba algoritma binarnim kodiranjem dobila su malo bolje rezultate nego grayevim kodiranjem. No, ovo ispitivanje je pokazalo da za funkciju *f2* bolje odgovara odabir 8 Booleovih i 8 realnih varijabli za oba kodiranja – binarno i grayevo. Rezultati su prikazani u tablicama 4.13 i 4.14.

Tablica 4.13 Rezultati za 8 Booleovih i 16 realnih varijabli

f2,binary	**SST**	**RW**
min	110	110
avg	111.2	110.8
max	112	112

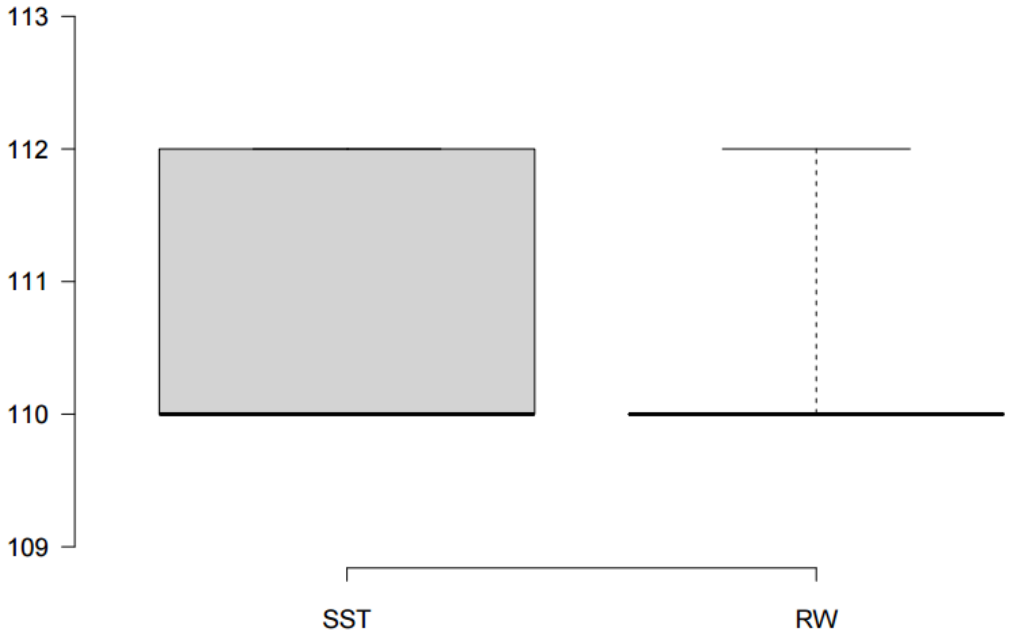


Slika 4.11 Rezultati funkcije f2 uz binarno kodiranje realnog vektora

Na slikama 4.13 i 4.14 vidimo izjednačenost oba ispitana algoritma prema dobivenim rezultatima.

Tablica 4.14 Rezultati za 8 Booleovih i 16 realnih varijabli

f2, gray	**SST**	**RW**
min	110	110
avg	110.8	110.4
max	112	112



Slika 4.12 Rezultati funkcije f2 uz grayevo kodiranje realnog vektora

4.4 Rezultati s vektorom cijelih brojeva

S obzirom da algoritmi *SteadyStateTournament* (SST) i *RouletteWheel* (RW) ne ovise o tipu genotipa, ta se dva algoritma ispituju s vektorom cijelih brojeva. U ovom slučaju korisnik treba paziti da interval iz kojeg se stvaraju novi vektori (*ubound* i *lbound*) odgovara izabranim brojevima varijabli. Na primjer, ako zadamo 8 Booleovih varijabli tablica istinitosti će imati 256 bita i ako zadamo da vektor cijelih brojeva ima 32 varijable (*size* = 32), tada bi trebalo staviti *lbound* = 0 i *ubound* = 255, jer će 8 bita predstavljati jednu vrijednost iz *IntGenotype* vektora, što znači da najviše mogu imati vrijednost 255. Ukoliko ovo nije zadovoljeno, korisnik će dobiti upozorenje da to treba ispraviti u konfiguracijskoj datoteci.

Iste funkcije svojstava – *f1*, *f2* i *f3* ispitane su i ovim genotipom, također s oba kodiranja (binarno i grayevo). Broj Booleovih funkcija postavljen je na 8, a broj cjelobrojnih varijabli na 16, te donja granica intervala na 0 i gornja na 65535 zbog spomenutog razloga.

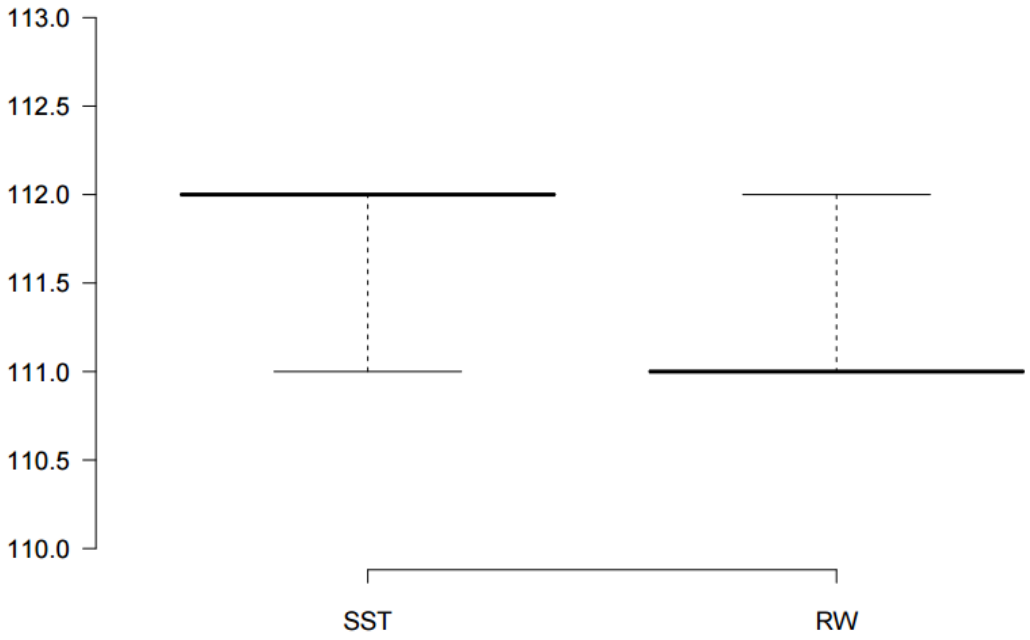
Rezultati dobiveni ispitivanjem funkcije *f1* prikazani su u tablicama 4.15 i 4.16. Za algoritam *SteadyStateTournament* uočavamo da nije došlo do razlike u rezultatima

kod ispitivanja binarnim i grayevim kodiranjem, dok je za *RouletteWheel* algoritam u jednom pokušaju pronađen bolji rezultat binarnim kodiranjem.

Tablica 4.15 Rezultati za 8 Booleovih i 16 cjelobrojnih varijabli

f1,binary	**SST**	**RW**
min	111	111
avg	111.8	111.2
max	112	112

Slika 4.13 prikazuje da je *SteadyStateTournament* algoritam pronašao malo bolje funkcije od algoritma *RouletteWheel*.

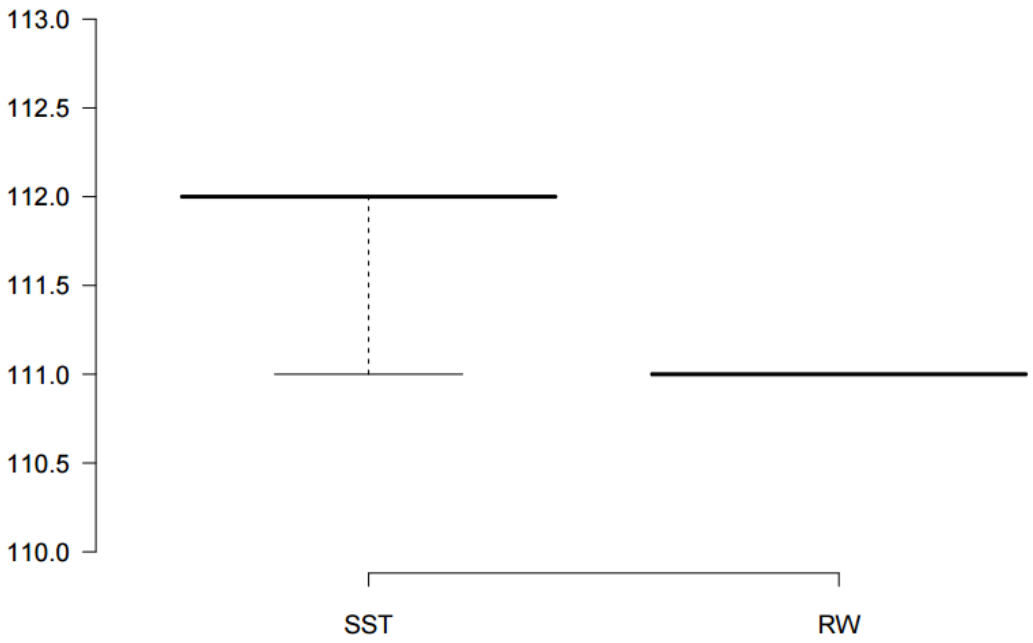


Slika 4.13 Rezultati funkcije f1 uz binarno kodiranje vektora cijelih brojeva

Tablica 4.16 Rezultati za 8 Booleovih i 16 cjelobrojnih varijabli

f1, gray	SST	RW
min	111	111
avg	111.8	111
max	112	111

Sa slike 4.16 može se uočiti da je *SteadyStateTournament* grayevim kodiranjem pronašao jednake rezultate, dok je *RouletteWheel* algoritam pronašao još lošije.



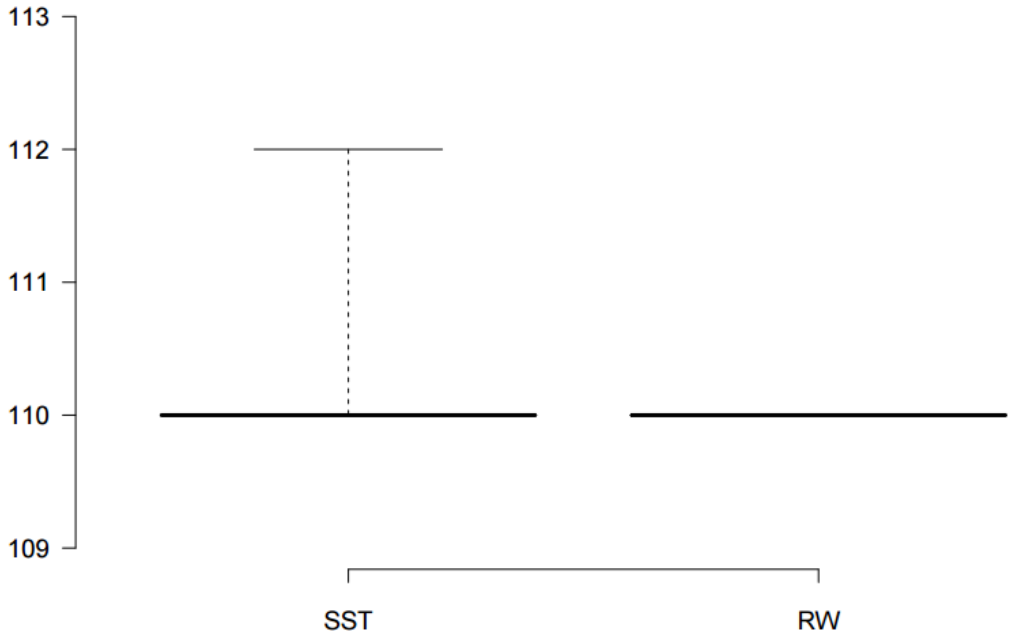
Slika 4.14 Rezultati funkcije f1 uz grayevo kodiranje vektora cijelih brojeva

Ispitivanjem algoritama na funkciji *f2* uočavamo da se ponovno binarno kodiranje pokazalo boljim za *RouletteWheel* algoritam, dok kod *SteadyStateTournament* algoritma nema razlike. U tablicu 4.17 i tablicu 4.18 upisani su rezultati dobiveni ispitivanjem.

Tablica 4.17 Rezultati za 8 Booleovih i 16 cjelobrojnih varijabli

f2,binary	**SST**	**RW**
min	110	110
avg	110.4	110
max	112	110

Sa slike 4.17 vidimo da su zapravo oba algoritma dobila vrlo slične rezultate.

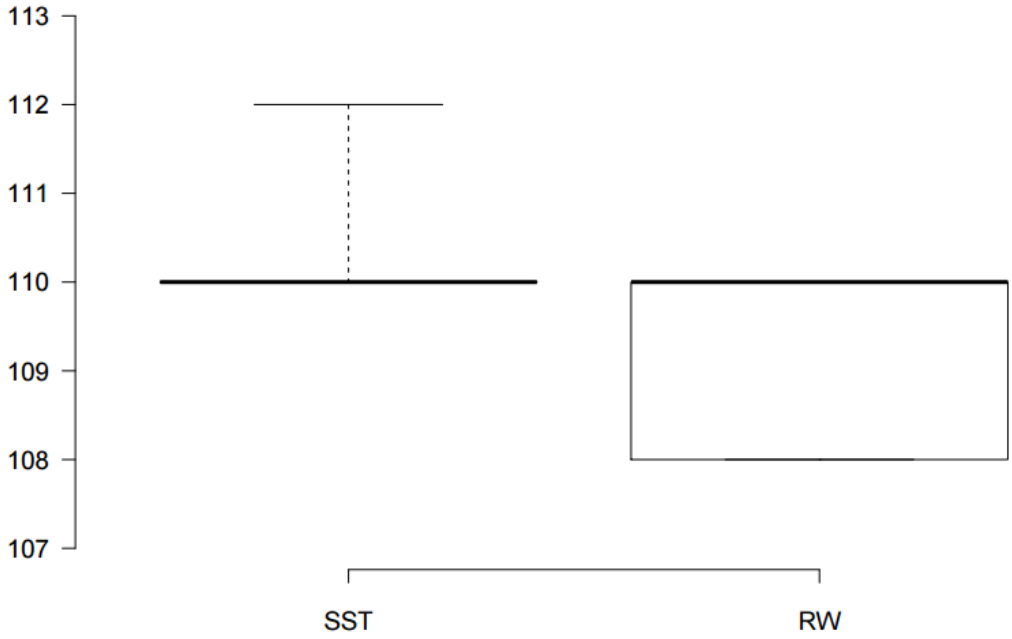


Slika 4.15 Rezultati funkcije f2 uz binarno kodiranje vektora cijelih brojeva

Tablica 4.18 Rezultati za 8 Booleovih i 16 cjelobrojnih varijabli

f2, gray	**SST**	**RW**
min	110	108
avg	110.4	109.2
max	112	110

Usporedbom slika 4.17 i 4.18 vidimo da je algoritam *SteadyStateTournament* dobio jednake rezultate binarnim i grayevim kodiranjem, dok je algoritam *RouletteWheel* grayevim kodiranjem pronašao lošije funkcije.



Slika 4.16 Rezultati funkcije f2 uz grayevo kodiranje vektora cijelih brojeva

Ispitivanjem funkcija *f1* i *f2* ispitivanjem primjećujemo da se korištenje vektora cijelih brojeva pokazalo lošije od korištenja realnog vektora, što se vidi usporedbom tablica 4.15 i 4.16 s tablicama 4.11 i 4.12 za funkciju *f1*, te usporedbom tablica 4.17 i 4.18 s tablicama 4.13 i 4.14 za funkciju *f2*. Binarnim i grayevim kodiranjem bolji rezultati postignuti su *FloatingPoint* genotipom iako je razlika manja kod funkcije *f2*, nego kod funkcije *f1* gdje se *FloatingPoint* genotip pokazao puno boljim odabirom.

Rezultati ispitivanja provedenih nad funkcijom *f3* prikazani su u tablicama 4.19 i 4.20. Oba algoritma u prvim generacijama pronađu vrijednost 0 i dalje ne napreduju, osim u jednom pokušaju gdje je *SteadyStateTournament* algoritam grayevim kodiranjem uspio pronaći vrijednost 1 što vidimo u tablici 4.20.

Tablica 4.19 Rezultati za 8 Booleovih i 16 cjelobrojnih varijabli

f3,binary	**SST**	**RW**
min	0	0
avg	0	0
max	0	0

Tablica 4.20 Rezultati za 8 Booleovih i 16 cjelobrojnih varijabli

f3, gray	**SST**	**RW**
min	0	0
avg	0.2	0
max	1	0

5. Zaključak

S obzirom na prijašnja istraživanja, rezultati u ovom radu ne odstupaju, tj. očekivana su. Činjenica da rezultati nisu ispod prosjeka je pozitivna jer to znači da se zapis tablice istinitosti pomoću vektora realnih i cijelih brojeva može koristiti kod optimizacije Booleovih funkcija za kriptografske postupke.

Kroz analizu rezultata uočavamo da za *FloatingPoint* genotip za funkciju *f1* bolje rezultate dobivamo kada koristimo kombinaciju s 8 Booleovih i 16 realnih varijabli nego kombinaciju s 8 Booleovih i 8 realnih varijabli, dok je za funkciju *f2* slučaj obrnut. Iz toga može se zaključiti da brojevi varijabli ovise o funkcijama koje se ispituju. Također, je li bolje koristiti binarno ili grayevo kodiranje iz dobivenih rezultata nije moguće zaključiti jer na primjer algoritam *SteadyStateTournament* s 8 Booleovih i 16 realnih varijabli za funkciju *f1* grayevim kodiranjem pronalazi bolje rezultate, a kod funkcije *f2* to čini binarnim kodiranjem. No, ipak za *RouletteWheel* algoritam kroz ispitivanja se može uočiti da je uvijek pronalazilo ili iste rezultate ili malo bolje binarnim kodiranjem. Dakle, za *RouletteWheel* algoritam grayevo kodiranje donosilo je samo lošije rezultate.

S obzirom da su za funkciju 12 svi algoritmi prikazivali jednake rezultate, možemo zaključiti da za tu funkciju ispitani evolucijski algoritmi općenito ne daju dobre rezultate. Isključivši sada rezultate za funkciju 12, možemo uočiti da za sva ispitivanja algoritmi *SteadyStateTournament* i *Clonalg* pronalaze najbolje rezultate.

Ispitivanjima kojima smo htjeli usporediti rezultate prikaza Booleovih funkcija pomoću vektora realnih i cijelih brojeva zaključujemo da je genotip *FloatingPoint* u svim ispitivanjima pokazao da je bolji od *IntGenotype* genotipa, tj. da je prikaz realnim vektorom korisniji za optimizaciju Booleovih funkcija u kriptografiji.

Ideja prikaza Booleovih funkcija realnih vektorom ili vektorom cijelih brojeva objašnjena u ovom radu vrlo je jednostavna, te se zasigurno mogu pronaći drugačiji načini pretvorbe realnog vektora ili vektora cijelih brojeva u tablicu istinitosti koji bi mogli pronalaziti i bolja rješenja. No, sigurno je da je to veliko područje koje se može istraživati, te je korisno zbog velikog broja optimizacijskih algoritama koji bi se mogli iskoristiti.

6. Literatura

[1] S. Picek (2015), *Applications of Evolutionary Computation to Cryptology*

[2] *http://en.wikipedia.org/wiki/Boolean_function*

[3] *http://web.math.pmf.unizg.hr/nastava/difraf/dif/p_o11.pdf* [predavanje]

Optimizacija Booleovih funkcija za kriptografske postupke

Sažetak

S obzirom da se Booleove funkcije najčešće prikazuju nizom bitova koji čine tablicu istinitosti, radi većih mogućnosti upotrebe raznih optimizacijskih algoritama koji koriste vektore realnih ili cijelih brojeva, ovaj rad opisuje prikaz Booleovih funkcija pomoću tih vektora. Booleove funkcije prikazane takvim vektorima optimirale su se evolucijskim algoritmima. Dobiveni rezultati prikazani su tablicama i vizualizirani grafovima, te su detaljno analizirani.

Ključne riječi: optimizacija, Booleove funkcije, vektor realnih brojeva, vektor cijelih brojeva

Optimization of Boolean functions for cryptographic applications

Abstract

Considering that Boolean functions are most often presented as series of bits that make up a truth table, for greater possibilities of using various optimization algorithms using vectors of real or integer numbers, this paper describes interpretation of Boolean functions using these vectors. Boolean functions presented with such vectors are optimized with evolutionary algorithms. Obtained results are shown in tables and graphs and are analyzed in detail.

Keywords: optimization, Boolean functions, vector of real numbers, vector of integer numbers