SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 4835

Modeliranje logičkih funkcija bez mogućnosti kopiranja evolucijskim algoritmima

Maša Burda

Zagreb, lipanj 2017.

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

Zagreb, 5. ožujka 2017.

ZAVRŠNI ZADATAK br. 4835

Pristupnik: **Maša Burda (0036485960)**
Studij: Računarstvo
Modul: Računalno inženjerstvo

Zadatak: **Modeliranje logičkih funkcija bez mogućnosti kopiranja evolucijskim algoritmima**

Opis zadatka:

Opisati problem oblikovanja modela logičkih funkcija bez mogućnosti kopiranja u uređajima za sigurnu komunikaciju. Istražiti vrste modela ponašanja zadanih logičkih funkcija i vezane optimizacijske probleme. Ostvariti programski sustav za optimizaciju modela zadanih funkcija temeljen na evolucijskim algoritmima te algoritmu evolucijske strategije. Ispitati učinkovitost preciznosti dobivenih modela s obzirom na veličinu polazne funkcije, preciznost prikaza i parametre optimizacijskih algoritama. Radu priložiti izvorne tekstove programa, dobivene rezultate uz potrebna objašnjenja i korištenu literaturu.

Zadatak uručen pristupniku: 10. ožujka 2017.
Rok za predaju rada: 9. lipnja 2017.

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

Djelovođa:
Prof. dr. sc. Danko Basch

Predsjednik odbora za završni rad modula:
Prof. dr. sc. Mario Kovač

Zahvaljujem prof. dr. sc. Domagoju Jakobovi\'cu, pod čijim je mentorstvom izrađen ovaj rad, na korisnim konzultacijama i pomoći u izradi rada, a najviše na otvaranju mogućnosti rada u području evolucijskog računanja i produbljivanju mojeg znanja o tom području.

SADRŽAJ

1. Uvod

Najvažnija stavka u sigurnosti i zaštiti podataka su, posebice u današnjem dobu, u kojem se poruke razmjenjuju globalnim računalnim i komunikacijskim mrežama, metode šifriranja podataka. Jedna od metoda je korištenje logičkih funkcija bez mogućnosti kopiranja, čiji je engleski naziv *physically unclonable funcions* - skraćeno PUF. PUF je hardverski implementiran sustav koji se ne može reproducirati zbog svojih posebnih karakteristika, a one uključuju varijabilnost kašnjenja žica i parazitskih kašnjenja logičkih vrata u proizvedenim integriranim krugovima. Takve logičke funkcije mogu biti implementirane sa smanjenim brojem vrata u usporedbi s tradicionalnim kriptografskim funkcijama. Navedene karakteristike predstavljaju povoljne mogućnosti za primjenu u kriptografiji i generiranju jedinstvenih kriptografskih ključeva.

U ovom radu korišten je program za simulaciju modela PUF-a, a optimizacija tog modela provodila se pomoću evolucijskih algoritama koji postoje u okruženju ECF (Evolutionary Computation Framework) i pomoću implementiranog algoritma evolucijske strategije naziva CMA-ES (*Covariance Matrix Adaptation Evolution Strategy*). Cilj ispitivanja provedenih nad tim algoritmima bio je utvrditi učinkovitost ostvarenih modela s obzirom na veličinu polazne funkcije, preciznost prikaza i parametre optimizacijskih algoritama. Za pojedine algoritme bilo je potrebno pronaći potrebnu preciznost s obzirom na navedene parametre kako bi se pronašlo zadovoljavajuće rješenje. Brojna ispitivanja služila su za utvrđivanje učinkovitosti određenih optimizacijskih algoritama na zadanom problemu i kako bi se mogla provesti usporedba između korištenih algoritama.

Sljedeće (drugo) poglavlje odnosi se na opis logičkih funkcija bez mogućnosti kopiranja i na moguće načine njihove izvedbe. U trećem poglavlju ukratko su opisani evolucijski algoritmi i njihova podjela i povezanost s problemom optimizacije PUF-ova. U četvrtom poglavlju dan je pregled algoritama evolucijske strategije, s posebnim naglaskom na algoritam CMA-ES i njegovu implementaciju. Peto poglavlje detaljno

opisuje tijek provedenih ispitivanja i rezultate tih ispitivanja.

2. Logičke funkcije bez mogućnosti kopiranja

Ideja korištenja svojstvenih slučajnih fizičkih značajki za prepoznavanje objekata, sustava i ljudi nije nova. Primjerice, identifikacija otisaka prstiju čovjeka datira barem od devetnaestog stoljeća. U osamdesetim i devedesetim godinama dvadesetog stoljeća korišteni su slučajni uzorci papira i optičkih oznaka za jedinstvenu identifikaciju novčanica. Formalizacija tog koncepta uvedena je već početkom dvadeset prvog stoljeća, prvo kao fizičke jednosmjerne funkcije, fizičke slučajne funkcije i konačno kao fizički nereproduktibilne funkcije - PUF. U godinama nakon ovog uvoda predložen je sve veći broj novih vrsta PUF-ova, s tendencijom prema integriranijim konstrukcijama. Praktična važnost PUF-a za sigurnosne primjene prepoznata je od samog početka, s posebnim naglaskom na svojstva nemogućnosti fizičkog kopiranja [1].

U posljednjih nekoliko godina narasla je popularnost korištenja logičkih funkcija bez mogućnosti kopiranja (skraćenica - PUF) za izradu jedinstvenih ključeva u uređajima za sigurnu komunikaciju. Zbog njihovih svojstava, često se povlači paralela između korištenja PUF-ova u kriptografiji i korištenja otiska prsta čovjeka za njegovu jedinstvenu identifikaciju. Imajući to na umu, PUF možemo definirati kao izraz prirođene i nereproduktibilne osobine specifične za instancu fizičkog objekta [2]. To zapravo znači da su PUF-ovi inovativni sklopovi koji se baziraju na izvlačenju svojstava iz fizičkih značajki integriranih krugova. Ove funkcije iskorištavaju varijabilnost kašnjenja žica i parazitsko kašnjenje logičkih vrata u proizvedenim krugovima, a mogu se provesti s redukcijom reda veličine broja vrata u usporedbi s tradicionalnim kriptografskim funkcijama [3]. Upravo te posebne karakteristike (zakašnjenje žica i tranzistora), koje se razlikuju od čipa do čipa i koje su ostvarene različitim dizajnom modela PUF-a, mogu omogućiti nisku cijenu autentifikacije pojedinačnih integriranih krugova i generiranje tajnih ključeva za kriptografske operacije [4]. Javni PUF (PPUF, public physically unclonable function) je PUF stvoren tako da je njegova simulacija izvediva,

ali traje vrlo dugo čak i kada su na raspolaganju dovoljni računalni resursi. Korištenjem PPUF-ova razvijena je konceptualno nova tajna razmjena ključeva i protokol javnog ključa koji je otporan na nepoželjne fizičke napade (npr. reverzno inženjerstvo) i ne koristi nedokazane matematičke pretpostavke. Pametna uporaba PPUF hardvera i djelomične simulacije omogućuje veliku prednost stranaka koje komuniciraju nad napadačem [5].

2.1. Općenite karakteristike izvedbe PUF-ova

PUF je općenito izveden kao funkcija ostvarena fizičkim komponentama, a svaka fizička komponenta ima karakteristike koje u njihovoj proizvodnji mogu varirati (npr. spomenuta kašnjenja). Takve karakteristike su nepredvidljive i zbog toga PUF-ove ne možemo kopirati (klonirati). PUF je implementiran tako da generira skup odgovora na skup postavljenih pitanja (*challenge-response* ovjera autentičnosti). *Challenge-response* ovjera je skup protokola u kojima jedna strana postavlja pitanje (*challenge*), a druga strana treba dati točan odgovor (*response*) na to pitanje. Na strukturu se pošalje fizički stimulans (ulaz) na koji se dobije reakcija (izlaz) uređaja – nepredvidljiva, ali ponavljajuća (za isti stimulans uvijek se dobije isti izlaz, ista reakcija). Taj stimulans je zapravo pitanje (*challenge*), a reakcija PUF-a, tj. izlaz je odgovor (*response*) [6].

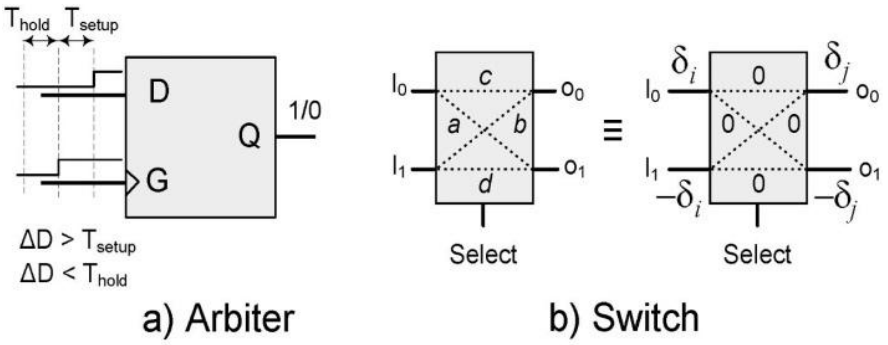
Pitanje-odgovor (*challenge-response*) ponašanje PUF-a drastično se mijenja kad je oštećeno, primjerice, od strane napadača. Uz njihovo svojstvo nemogućnosti kopiranja, to čini PUF-ove pogodnima da budu sredstvo za sigurno pohranjivanje ključeva. Umjesto pohrane ključeva u digitalnom obliku u memoriji uređaja, ključ se može izvući iz PUF-a ugrađenog u uređaj. Samo kada je ključ potreban (npr. za protokol provjere autentičnosti), izvlači se iz PUF-a i odmah se briše kada više nije potreban. Na taj način ključ je prisutan u uređaju minimalnu količinu vremena i time je manje osjetljiv na fizičke napade. Nažalost, nedavna analiza pokazala je da su mnoge od najsuvremenijih PUF struktura osjetljive na različite sigurnosne napade [7].

PUF mora biti lako evaluirati (izračunati njegov izlaz), što znači da fizički uređaj mora moći evaluirati funkciju u kratkom vremenu. Također bi ga trebalo biti teško opisati. Stoga iz ograničenog broja vjerodostojnih fizičkih mjerenja ili upita odabranih CRP-ova (*challenge-response* parova) napadač koji više nema uređaj i koji može koristiti samo ograničenu količinu resursa (vrijeme, novac i sl.) može dobiti zanemarivu

količinu informacija o odgovoru na slučajno odabrani izazov. Izlaz PUF-a trebalo bi biti teško predvidjeti, a gotovo nemoguće bi trebalo biti napraviti identičan PUF čak i ako se zna njegova struktura i proces izrade.

2.2. Vrste PUF-ova

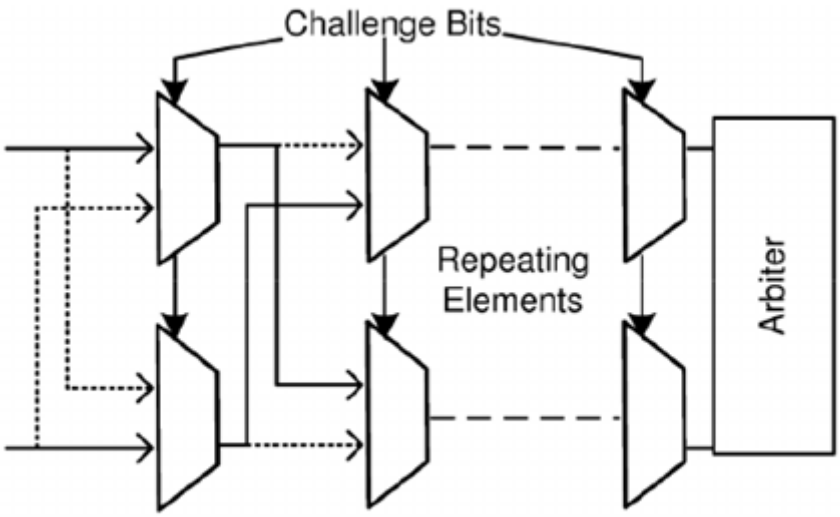
Postoje različite vrste PUF-ova s obzirom na način njihove izrade. Silicijski PUF-ovi iskorištavaju varijabilnost proizvodnje kako bi generirali jedinstveno ulazno / izlazno mapiranje za svaki integrirani krug. Silicijski PUF-ovi temeljeni na kašnjenju (*Delay-based*) koriste varijacije kašnjenja CMOS logičke komponente kako bi se stvorili jedinstveni odgovori. Odgovori se generiraju usporedbom analogne vremenske razlike između dva puta kašnjenja koji moraju biti ekvivalentni konstrukcijom na razini logike, ali su različiti zbog varijabilnosti proizvodnje. Strukture temeljene na kašnjenju koriste digitalnu komponentu - arbitar, koji prevodi analognu razlučivost vremena u digitalnu vrijednost. Arbitar je sekvencijalna komponenta s dva ulaza i jednim izlazom.



Slika 2.1: Osnovni implementacijski blokovi PUF-ova [7]

Izlaz iz arbitra je 1 ako rastući brid signala stiže na svoj prvi ulaz ranije u usporedbi sa signalom koji dolazi na drugi ulaz barem za vrijednost praga. U suprotnom, izlaz

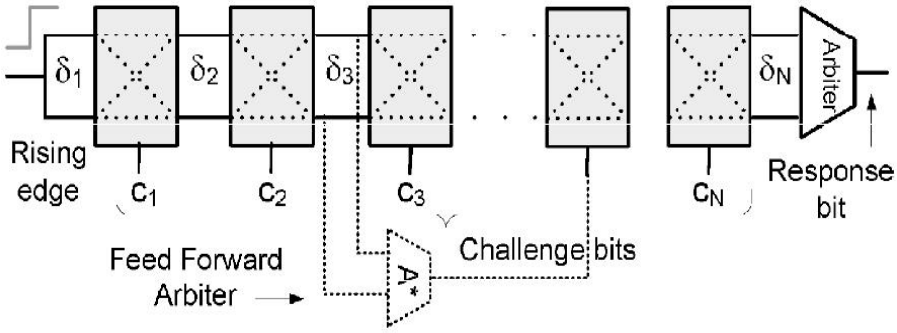
iz arbitra je 0. Slika 2.1.a prikazuje arbitar izveden pomoću bridom okidanog zapornog sklopa. Ako je vremenska razlika između dolaznih signala manja od vremena postavljanja i zadržavanja (*setup and hold time*) zapornog sklopa, arbitar može postati metastabilan i ne može proizvesti precizan deterministički izlaz. Shematski prikaz PUF-a takve izvedbe (*Arbiter PUF-a*) prikazan je slikom 2.2.



Slika 2.2: Arbiter PUF [7]

Arbiter PUF je model PUF-a korišten u simulaciji pa će stoga biti detaljnije objašnjen. *Arbiter PUF* ima gornji i donji signal koji prolazi kroz k razina kašnjenja (*delay stages*). Svaka od tih razina sastoji se od dva identična dvobitna multipleksora koji primaju gornji i donji signal kao ulaz. Svaki tranzistor u multipleksorima ima malo drukčija kašnjenja zbog varijacija u procesu proizvodnje i zbog toga je kašnjenje drugačije za gornji i donji signal. Te razlike u kašnjenjima su jedinstvene za svaki čip. Za ulaz (*challenge* bit) $c_i = 1$, gdje je i razina (*stage*) i, multipleksori zamjenjuju gornji i donji signal, a ako je $c_i = 0$, nema zamjene. U takvom načinu izvedbe, signal može proći različitim putevima – *Arbiter PUF* za k razina ima 2^k različitih puteva. Arbitar (sudac) na kraju svih razina kašnjenja odlučuje koji je signal bio brži, gornji ili donji. Ako je gornji signal bio brži, izlaz je 1, a ako je donji signal brži, onda je izlaz 0.

Problem s ovakvim modelom PUF-a je da se može simulirati programski, na računalu. Parametri za programski model mogu se aproksimirati korištenjem tehnika strojnog učenja. Ideja kojom bi se povećala otpornost *Arbiter PUF-a* od napada strojnim učenjem je dodavanje nelinearnih elemenata u dizajn. Uobičajeno je dodavanje XOR elemenata, pa tako nastaje XOR PUF. U n-XOR PUF-u, n *Arbiter PUF-ova* postavljeno je na čip. Svaki *Arbiter PUF* prima iste ulazne bitove (*challenge*), a XOR izračunamo na izlazima (*responses*) od n PUF-ova te tako dobijemo završni izlaz. Povećavanjem broja PUF-ova koji ulaze u XOR povećava se otpornost od napada, no, time opada pouzadnost XOR PUF-a. Iz navedenih razloga u praksi je broj PUF-ova koji ulaze u XOR ograničen. Izlaz k-razinskog *Arbiter PUF-a* je određen razlikom kašnjenja između gornjeg i donjeg signala. Razlika kašnjenja je suma razlika kašnjenja u svim razinama (k). Razlika kašnjenja pojedine razine ovisi o odgovarajućem *challenge* bitu. Dakle, postoje dvije razlike kašnjenja za svaku razinu i, $\delta_{1,i}$ razlika je kašnjenja za *challenge* bit 1, a $\delta_{0,i}$ za *challenge* bit 0. [6]



Slika 2.3: Paralelna PUF struktura [7]

U paralelnoj strukturi PUF-a s k razina generiranje jednog izlaznog signala zahtijeva da signal prođe kroz dva paralelna puta s više segmenata koji su povezani nizom sklopki s dva ulaza i dva izlaza, kako je prikazano na slici 2.3. Ovaj uvjet simetrije dvaju putova određuje implementacijsku složenost PUF-a na FPGA pločicama. Slika 2.4 prikazuje uzorak FPGA simetričnog usmjeravanja za implementaciju *Arbiter*

PUF-a. *Arbiter PUF* teško je implementirati na FPGA pločicama zbog različitih kašnjenja prisutnih između parova elemenata krugova za koje je potrebno da budu simetrični [7].

Slika 2.4: Arbiter PUF implementiran na FPGA [7]

Programski model k-razinskog *Arbiter PUF-a* koristi sljedeće formule za izračun: za vektor kašnjenja $\vec{\omega} = (\omega_1, \ldots, \omega_{k+1})$ vrijedi:

$$\omega_1 = \delta_{0,1} - \delta_{1,1} \tag{1}$$

$$\omega_i = \delta_{0,i-1} + \delta_{1,i-1} + \delta_{0,i} - \delta_{1,i}, 2 \leq i \leq k \tag{2}$$

$$\omega_{k+1} = \delta_{0,k} + \delta_{1,k} \, . \tag{3}$$

δ je oznaka za ranije spomenutu razliku kašnjenja pojedinih razina.

Za razliku kašnjenja koristi se formula skalarnog produkta transponiranog vektora kašnjenja i svojstvenog vektora:

$$\Delta D = \vec{\omega}^T \vec{\Phi} \tag{4}$$

$$r = 1, \Delta D < 0$$

$$r = 0, \Delta D > 0\ . \tag{5}$$

Svojstveni vektor izračunava se iz ulaznog vektora (*challenge vector*):

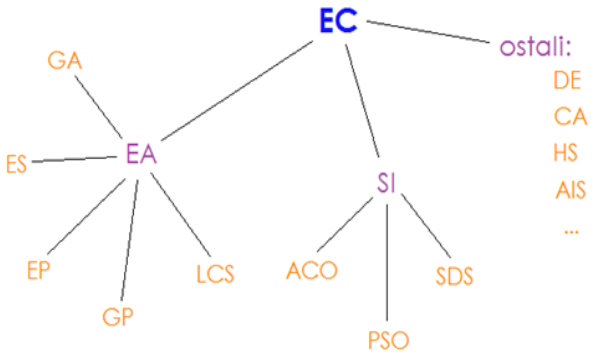
$$\Phi_i = \sum_{l=i}^{k} (-1)^{c_l} \quad \text{za } 1 \leq i \leq k \tag{6}$$

$$\Phi_{k+1} = 1\ . \tag{7}$$

3. Evolucijski algoritmi

Evolucijski algoritmi (EA - *evolutionary algorithms*) dio su područja evolucijskog računanja (EC - *evolutionary computation*) koje se proučava u širem području umjetne inteligencije, zajedno sa skupinom algoritama temeljenih na inteligenciji rojeva (SI - *swarm intelligence*) i ostalim algoritmima. Evolucijsko računanje pripada heuristikama i metahuristikama, što znači da ove tehnike ne garantiraju pronalazak pravog globalnog optimuma, već dovoljno dobrog rješenja. Takve metode su korisne kada je optimalno rješenje nemoguće pronaći jer pronalaze zadovoljavajuće rješenje u konačnom vremenu. Te metode su iterativne, a tijekom svake iteracije cilj je približiti se globalnom optimumu. Evolucijsko računanje temeljeno je na radu s populacijom nad kojom se izvršavaju određene operacije, poput križanja jedinki i mutacije jedinki. Te operacije inspirirane su evolucijskim procesima iz prirode, što znači da je cilj da bolja rješenja preživljavaju, a loša rješenja se odbace. Odbacivanje loših rješenja provodi se na temelju izračunate dobrote (*fitness*) jedinke. Korištenjem tih operacija početni skup nasumično odabranih rješenja kroz određeni broj iteracija evoluira i algoritam pronalazi sve bolja i bolja rješenja.

Osnovna prednost ovih algoritama nad drugima je preslikavanje rješenja problema faktorijelne ili eksponencijalne složenosti u probleme polinomijalne složenosti. Budući da se ne pretražuje čitavi problemski prostor, optimalno rješenje nije garantirano. Međutim, pronađena rješenja su u većini slučajeva optimalna ili vrlo blizu optimalnih. Metode koje spadaju u područje evolucijski algoritama su genetski algoritmi (GA - *genetic algorithm*), evolucijske strategije (ES - *evolution strategy*), genetsko programiranje (GP - *genetic programming*), evolucijsko programiranje (*evolutionary programming*) i klasifikatorski sustavi sa sposobnošću učenja (LCS – *learning classifier systems*). U ovo radu korišteno je nekoliko evolucijskih algoritama, uključujući algoritam evolucijske strategije CMA-ES (*Covariance Matrix Adaptation Evolution Strategy*).



Slika 3.1: Podjela algoritama evolucijskog računanja [8]

3.1. Povezivanje evolucijskih algoritama s problemom optimizacije PUF-ova

Isprobani su modeli PUF-ova temeljeni na postojećim implementacijama algoritama *Genetic Hooke-Jeeves* (*GenHookeJeeves*), *Clonal selection algorithm* (*ClonAlg*), *Steady state tournament* (SST, *SteadyState*), *Optimization Immune algorithm* (*OptIA*). Dodatno, implementiran je algoritam evolucijske strategije CMA-ES (*Covariance Matrix Adaptation Evolution Strategy*) i detaljnije je objašnjen u četvrtom poglavlju.

Navedeni optimizacijski algoritmi korišteni su u sklopu sustava ECF - *Evolutionary Computation Framework* i služe za optimiziranje zadanog PUF-a. U jednom Visual Studio projektu nalazi se program za evaluaciju PUF-ova koji povezuje ECF i njegove algoritme s problemom optimizacije PUF-ova. Pokretanjem tog programa sa željenima parametrima (koji su opisani u petom poglavlju zajedno s tijekom ispitivanja programa) optimizira se vektor kašnjenja za k-razinski *Arbiter PUF*. Svi izvorni kodovi napisani su u programskom jeziku C++.

Pseudokod za evaluaciju PUF-a [6]:

$error = 0$

za svaki response bit

$delay = skalarniProdukt(featureVektor, jedinka)$

ako $delay < 0$

$r = 1$

inače

$r = -1$

ako $r! = trenutniResponseBit$

$error++$

Razlika između objašnjenog programskog modela *Arbiter PUF-a* i programske implementacije je u tome što se umjesto binarnih vrijednosti 1 i 0 koriste vrijednosti 1 i -1.

Vektor kašnjenja prikazuje se strukturom *FloatingPoint* genotipa preuzetog iz ECF-a. Taj genotip predstavljen je vektorom realnih vrijednosti. Gornja i donja granica postavljene su u datoteci s parametrima na vrijednosti -1 i 1. One predstavljaju interval u kojem se nalaze dopuštene vrijednosti vektora. Parametri genotipa i željenog algoritma zadaju se u zasebnoj datoteci (najčešće nazvanoj "parameters.txt") koja ima strukturu XML datoteke. Postavljanje bitnih parametara na određene vrijednosti pojašnjeno je u petom poglavlju, u kojem je detaljno opisan cjelokupan tijek ispitivanja programa s različitim vrijednostima parametara.

4. Algoritam CMA-ES

CMA-ES skraćenica je za algoritam naziva *Covariance Matrix Adaptation Evolution Strategy*. Pripada skupini evolucijskih algoritama. Evolucijske strategije su optimizacijske tehnike bazirane na prilagođavanju i evoluciji. Njihova središnja ideja je stvaranje jednog ili λ potomaka operacijom mutacije iz jednog ili μ roditelja. U sljedeću se generaciju prenosi ili μ roditelja i λ potomaka ili samo λ potomaka. Roditelji i potomci predstavljaju potencijalno rješenje optimizacijskog problema. Potencijalna rješenja prikazuju se pomoću vektora realnih brojeva. Broj na određenom mjestu unutar vektora opisuje neku karakteristiku samog rješenja. Kako bi se došlo do rješenja određenog problema koriste se operatori mutacije i križanja (rekombinacije). U sljedeću generaciju prenose se samo najbolje jedinke koje se odabiru procesom selekcije.

4.1. Klasifikacija evolucijskih strategija

Označimo broj roditelja u nekoj generaciji x sa μ, a broj potomaka u generaciji x sa λ. Postoji sedam različitih tipova evolucijskih procesa koje koriste evolucijske strategije i s obzirom na te tipove evolucijskih procesa, evolucijske strategije možemo podijeliti u sedam skupina. [9]

1.) (1+1)-ES
U populaciji postoje samo dvije jedinke. Jedna jedinka je roditelj iz kojeg nakon reprodukcije procesom mutacije nastaje potomak. Proces selekcije se obavlja između te dvije jedinke, a u sljedeću generaciju ide ona jedinka koja je bolja, tj. koja ima bolji faktor dobrote.

2.) (μ +1)-ES Populacija se sastoji od μ jedinki roditelja. Mutacijom jedne jedinke nastaje jedan potomak koji se konstantno reproducira. Iz spojenog seta potomaka izabrane jedinke i trenutne populacije odbacuje se jedinka koja ima najmanji faktor dobrote.

3.) (μ + λ)-ES
U ovom slučaju iz μ jedinki roditelja nastaje λ potomaka procesom mutacije. Uvjet za ovaj proces je da je nastalo više djece nego što je roditelja, dakle λ>μ. Svaki od λ potomaka ima svoj faktor dobrote, kao što imaju i svi roditelji. Najboljih μ jedinki iz skupa roditelja i potomaka zajedno prelazi u sljedeću generaciju.

4.) (μ , λ)-ES
Populacija se sastoji od μ jedinki roditelja, koji procesom mutacije daju λ potomaka, s time da vrijedi λ>μ. Svaki od λ potomaka ima svoj faktor dobrote. Za razliku od prethodnog slučaja, ovdje se za prijelaz u novu generaciju promatraju samo potomci, dakle roditelji ne ulaze u izbor. Iz λ potomaka izabire se μ najboljih jedinki koje prelaze u sljedeću generaciju.

5.) (μ/ρ , λ)-ES
Ova strategija je (μ, λ) strategija uz dodatak parametra ρ. Ovaj parametar označava broj jedinki roditelja koji se upotrebljava u procesu reprodukcije. Ukoliko je ρ=1, ova strategija je analogna strategiji (μ, λ)-ES, tj. prilikom reprodukcije koristi se samo jedna jedinka iz populacije roditelja, što znači da se upotrebljava operator mutacije. Ukoliko je ρ=2, prilikom reprodukcije se koriste dvije jedinke iz populacije roditelja, što znači da se upotrebljava operator rekombinacije. Selekcija je analogna selekciji kod (μ , λ)-ES.

6.) (μ/ρ + λ)-ES
Ova strategija je (μ + λ) strategija uz ponovni dodatak parametra ρ. Za reprodukciju vrijede ista pravila kao i kod (μ/ρ , λ) strategije, a selekcija je analogna selekciji kod (μ + λ)-ES.

7.) (μ' , λ'(μ , λ)x)-ES
Kod ove strategije se iz populacije roditelja veličine μ' kreira λ' potomaka i izolira na x generacija. U svakoj od x generacija stvara se λ potomaka od kojih samo μ najboljih prelazi u sljedeću generaciju. Nakon x generacija izabiru se najbolje jedinke

od x izoliranih populacija i krug kreće ponovno sa λ' novih jedinki potomaka. Ova strategija nije implementirana u ECF-u.

4.2. Opis tijeka algoritma evolucijske strategije

Slikom 4.1. prikazan je tijek algoritma evolucijske strategije u pseudokodu.

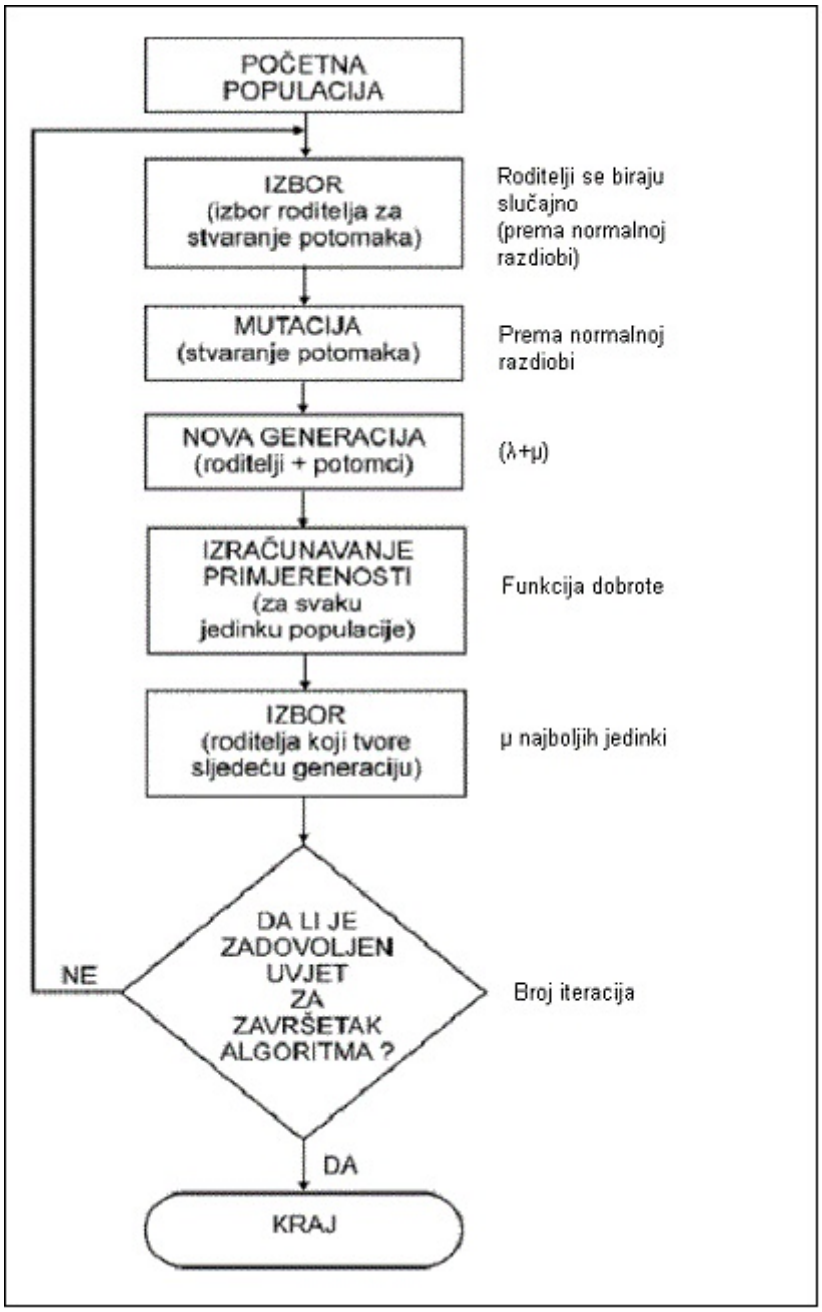
```
Evolucijska strategija{
      t = 0;
      generiraj početnu populaciju P(0);
      evaluiraj P(0); //provjeri dobrotu inicijalne populacije
      sve dok nije zadovoljen uvjet završetka evolucijskog procesa {
            izaberi najboljih μ roditelja iz P(t) i stavi ih u Pp(t);
            iz Pp(t) reproduciraj λ potomaka i stavi ih u Pt(t);
            mutiraj Pt(t);
            evaluiraj Pt(t);
            ako se koristi plus strategija P(t+1) = Pt(t) ∪ P(t);
                  inače P(t+1) = Pt(t);
            t=t+1;
      }
}
```

Slika 4.1: Pseudokod evolucijske strategije [10]

Proces stvaranja nove generacije može se opisati kroz sljedeće korake:

1.) stvori λ novih jedinki:
   a) odaberi slučajnih μ roditelja, tako da je ρ manje ili jednako μ
   b) križaj ρ roditelja kako bi dobio potomka
   c) odredi novu vrijednost parametra ρ prema normalnoj razdiobi
   d) mutiraj dijete pomoću nove vrijednosti parametra ρ
2.) odaberi novu populaciju na temelju funkcije dobrote:
   a) iz populacije djece ukoliko se radi o (λ , μ)-ES
   b) iz populacije djece i roditelja ukoliko se radi o $(\lambda + \mu)$-ES.

Dijagram tijeka evolucijske strategije prikazan je na Slici 4.2. [10]



Slika 4.2: Dijagram tijeka evolucijske strategije [10]

4.3. Opis implementacije algoritma CMA-ES

Za pomoć pri implementaciji korišten je već postojeći kod iz završnog rada [11]. Taj kod izmjenjen je tako da mu više nisu potrebne dodatne biblioteke za računanje različitih (većinom prilično složenih) operacija s matricama, koje su sastavni dio algoritma CMA-ES (na primjer rastavljanje matrica, računanje svojstvenih vrijednosti i svojstvenih vektora, traženje inverza matrice). Te operacije su u novoj implementaciji napisane u kodu samog algoritma.

CMA-ES je evolucijska strategija s adaptacijom kovarijantne matrice, bazirana na normalnoj razdiobi pomoću koje postupno prilagođava očekivanje, standardnu devijaciju i kovarijancu, odnosno kovarijantnu matricu distribucije. Glavne osobine ovog algoritma su postupno povećavanje uspješnosti generacije na način da se povećava vjerojatnost odabira uspješne jedinke i pamćenje evolucijskog puta strategije. Pamćenje evolucijskog puta omogućava da se pretraživanje s ciljem pronalaženja optimalnog rješenja kreće u pravome smjeru - prema globalnom optimumu. Postupno povećavanje uspješnosti generacije ogleda se u tome da se, korak po korak, populacija sve više približava optimalnoj vrijednosti, a kovarijanca i disperzija se smanjuju.

Jedinke koje prelaze u novu populaciju biraju se po multivarijantnoj normalnoj distribuciji (ako se radi o višedimenzionalnom problemu). Pri tome je vektor X vektor sa vrijednostima jedinke i zapisuje se kao:

$$X \sim \mathcal{N}(\mu, C), \tag{8}$$

gdje je μ vektor očekivane vrijednosti razdiobe, a C nenegativna, simetrična kovarijantna matrica razdiobe.
Matrica C rastavlja se na umnožak:

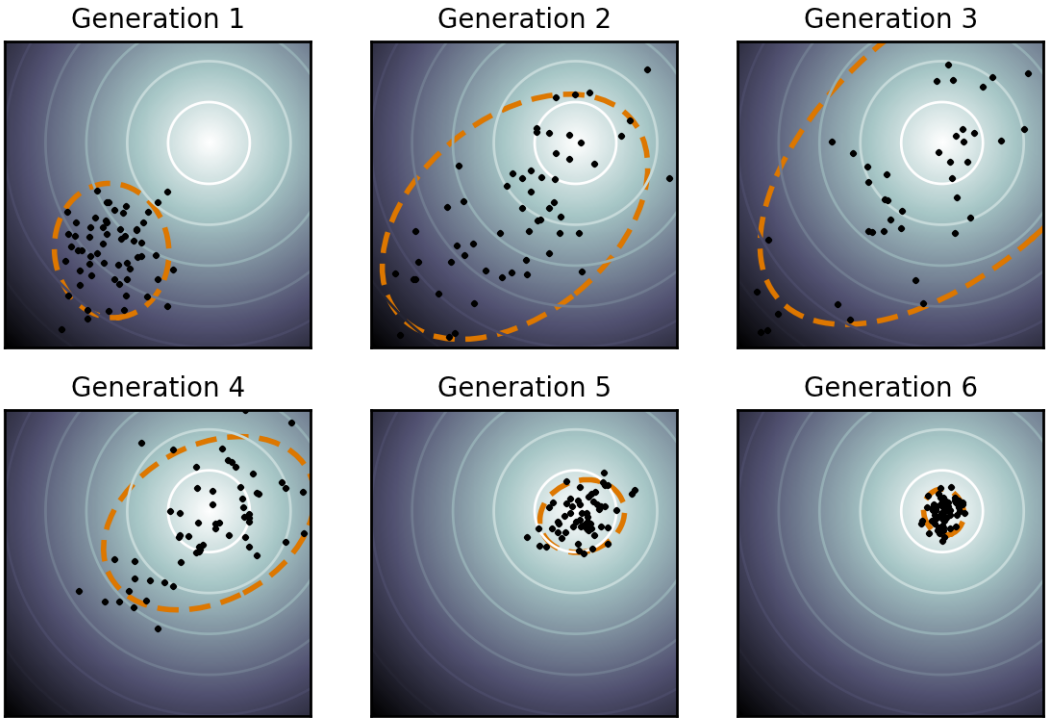
$$C = BDB^{-1}, \tag{9}$$

gdje je B matrica svojstvenih vektora (jedan stupac matrice - jedan vektor), a D dijagonalna matrica sa svojstvenim vrijednostima (jedan element dijagonale - jedna svojstvena vrijednost).
Izraz za uzorkovanje vrijednosti po normalnoj razdiobi dobiva se korištenjem nekoliko matematičkih teorema i prikazanog rastava matrice:

$$X = \mu + BD^{1/2}Y, \tag{10}$$

gdje je Y vektor nasumičnih vrijednosti distribuiranih po jediničnoj normaloj razdiobi ($\mathcal{N}(0, 1)$).

Postoje različite varijante ovog algoritma s obzirom na ranije navedene tipove selekcije. Najčešće odabirana implementacija je $(\mu/\mu_w, \lambda)$ - CMA-ES, gdje μ_w označava da se kod izbora roditelja koristi težinska funkcija, koja ima formulu po kojoj se određuje važnost jedinke, odnosno vjerojatnost odabira te jedinke. Također, varijante algoritma se razlikuju ovisno o tome koji parametri se mijenjaju, je li omogućeno ponovno pokretanje, povećavanje populacije ili kombiniranje s nekim drugim strategijama i proširivanje (dodavanje) mogućnosti.



Slika 4.3: Koncept smjera optimizacije u CMA-ES algoritmu [12]

Na slici 4.3 ilustriran je koncept pokretanja optimizacije s prilagodbom kovarijantne matrice na jednostavnom dvodimenzionalnom problemu. Sferni optimizacijski prostor prikazan je neprekinutim linijama jednakih vrijednosti funkcije cilja. Populacija (točkice) je mnogo veća nego što je potrebno, ali jasno pokazuje kako se raspodjela populacije (točkasta crta) mijenja tijekom optimizacije. Na ovom jednostavnom problemu, populacija se koncentrira na prostor globalnog optimuma unutar nekoliko

generacija.

Implementirana su tri različita tipa evolucijske strategije s adaptacijom kovarijantne matrice. Jedan je $(\mu/\mu, \lambda)$ - CMA-ES. Njegovo svojstvo je da sve roditelje smatra jednako bitnima. Srednja vrijednost roditelja bit će aritmetička sredina jedinki roditelja. To znači da težinska funkcija ima formulu $w_i = 1/\mu$. Ako se želi koristiti taj tip, potrebno je u datoteci s parametrima navesti parametar $type = 0$. Taj tip je u implementaciji označen enumeracijom pod brojem 0 ("NORMAL").

Drugi je tip IPOP-CMA-ES (increasing population size CMA-ES), odnosno $(\mu/\mu_w, \lambda)$ - IPOP- CMA-ES. Ima mogućnost ponovnog pokretanja s povećanom populacijom. Njegova težinska funkcija, za razliku od prethodnog tipa algoritma, nije jednaka za sve jedinke, već daje veću prednost jedinkama koje su bile uspješnije. Odabire ga se postavljanjem parametra "type" na vrijednost 1 ("IPOP").

Treća varijanta je $(\mu/\mu_w, \lambda)$ - CMA-ES A-CMA-ES (active CMA-ES, modifikacija CMA-ES tako da ne koristi samo podatke o najboljim potomcima, nego i podatke o najlošijim potomcima jer se tako razdioba uz pomicanje prema najboljem rješenju također i odmiče od najlošijeg rješenja). U toj varijanti algoritma, obavlja se i promjena kovarijantne matrice s negativnom matricom, što znači da se u obzir uzimaju najgore jedinke populacije i oduzimaju se od kovarijantne matrice, čime se smanjuje njihov utjecaj. Ova varijanta algoritma odabire se postavljanjem parametra "type" na vrijednost 2 ("ACTIVE").

4.3.1. CMA-ES parametri

U nastavku su objašnjeni parametri algoritma CMA-ES.
Osim broja roditelja (μ) i broja potomaka(λ), koji su karakteristični parametri za evolucijske strategije, CMA-ES koristi puno svojih internih parametara koji su neovisni o optimizacijskom problemu. Većina njih nije dostupna korisniku za mijenjanje i njihova vrijednost zapravo ovisi o implementaciji algoritma. [11]

Dimenzija problema (N) je dimenzija koordinatnog sustava nad kojim se radi. Povećavanjem dimenzije povećava se složenost i vrijeme izračuna.

Očekivana vrijednost (engl. *mean*), oznaka m, opisuje položaj u prostoru u kojem se nalazi sredina oko koje se biraju nove jedinke prilikom stvaranja populacije. Očekivana vrijednost je vektor čija je veličina određena dimenzijom problema. Srednja vrijednost se tijekom trajanja algoritma računa tako da se najbolje jedinke skaliraju s težnskom funkcijom ili se računa njihova srednja vrijednost. Očekivana vrijednost trebala bi biti jednaka ili približna optimalnoj vrijednosti funkcije cilja.

Veličina koraka (engl. *step-size*), oznaka σ, skalarna je vrijednost koja kaže koliko se u širinu uzorkuje, odnosno kolika je disperzija za sve dimenzije. To je promjenjiva vrijednost koja se prilagođava s obzirom na uspješnost generacije. Koliko će se promijeniti veličina koraka ovisi o evolucijskom putu (engl. *evolution path*). Evolucijski put je zbroj prijeđenih koraka do promatranog trenutka, pri čemu korak označava razliku između očekivane vrijednosti trenutne generacije i očekivane vrijednosti prijašnje generacije. Ako je duljina puta velika, onda i veličina koraka mora narasti da može obuhvatiti optimalno rješenje. No, ako je duljina puta mala, treba smanjiti veličinu koraka da se unutar područja uzorkovanja može konvergirati k optimalnom rješenju. [11]

Kovarijantna matrica (engl. *covariance matrix*), oznaka C, ponaša se kao parametar disperzije, ali za više dimenzija [11]. Ona određuje kolika će biti disperzija za svaku dimenziju. Kovarijantna matrica mijenja se tako da pomiče prostor pretraživanja prema vrijednostima jedinki koje su u prošlom koraku bile uspješne. Unutar kovarijantne matrice se tako pamti informacija o uspješnosti populacija. Postoje tri

načina prilagodbe kovarijantne matrice:
1. Ažuriranje matricom ranga jedan - prilagodba ovisi samo o evolucijskom putu kovarijantne matrice, slijedeći pritom gradijent funkcije dobrote.
2. Ažuriranje matricom ranga μ - prilagodba na temelju ponderirane srednje vrijednosti boljeg dijela populacije, što je korisno kod velikih populacija jer direktno uzima u obzir sve vrijednosti roditelja.
3. Ažuriranje negativnom matricom - prilagodba s obzirom na najlošiji dio populacije.

Stope učenja (engl. *learning rate*) su konstante koje određuju kojom brzinom i kojim intenzitetom će se događati promjene. Intuitivno je jasno da bi premala stopa učenja bila jako neefikasna, no neefikasna bi bila i prevelika stopa učenja jer može „pregaziti“ optimalno rješenje i teško će konvergirati. [11]
CMA-ES koristi četiri stope učenja:
1. za ažuriranje evolucijskog puta veličine koraka,
2. za ažuriranje evolucijskog puta kovarijantne matrice,
3. za ažuriranje kovarijantne matrice za matricu ranga jedan,
4. za ažuriranje kovarijantne matrice za matricu ranga μ.

4.3.2. Pseudokod s objašnjenjem za tip $(\mu/\mu_w, \lambda)$-CMA-ES

1. Unos i inicijalizacija parametara

Kovarijantna matrica C i njene komponente (matrica svojstvenih vektora B i matrica svojstvenih vrijednosti D) se inicijaliziraju na matricu identiteta I (elementi na dijagonali imaju vrijednost jedan, a svi ostali elementi su nula):

$$C = I, B = I, D = I. \tag{11}$$

Evolucijski putovi za kovarijantnu matricu i veličinu koraka se postave na nul-vektore veličine N:

$$p_c = \vec{0}, p_\sigma = \vec{0}. \tag{12}$$

Vektor težine w može se odabrati na tri načina tako da bude neutralan (sve jedinke su jednako važne), linearan ili logaritamski. Odabir se postiže upisivanjem parametra *weightType* i njegovim postavljanjem na 0 (neutralan), 1 (linearan) ili 2 (logaritamski). Općenito, za vektor težine vrijede uvjeti:

$$w_1 \geq w_2 \geq ... \geq w_\mu, \tag{13}$$

$$\sum_{i=1}^{\mu} w_i = 1. \tag{14}$$

Za efektivnu vrijednost populacije μ_w vrijedi formula:

$$\mu_w = \frac{1}{\sum_{i=1}^{\mu} w_i^2}. \tag{14}$$

Za stope učenja preporučene su vrijednosti:
za kovarijantnu matricu: $c_c \approx \frac{4}{N}$
za veličinu koraka: $c_\sigma \approx \frac{4}{N}$
za rang-jedan matricu: $c_1 \approx \frac{2}{N^2}$
za rang-μ matricu: $c_\mu \approx \frac{\mu_w}{N^2}$.

U implementaciji su za ovaj rad korištene formule preporučene od strane autora ovog algoritma (Hansen, 2001.) [11]. Te su formule:

$$c_c = \frac{4+\frac{\mu_w}{N}}{N+4+\frac{2\mu_w}{N}} \tag{15}$$

$$c_\sigma = \frac{\mu_w+2}{N+\mu_w+5} \tag{16}$$

$$c_1 = \frac{2}{(N+1.3)^2+\mu_w} \tag{17}$$

$$c_\mu = min(1 - c_1, 2\frac{\mu_w-2+\frac{1}{\mu_w}}{(N+2)^2+\mu_w}). \tag{18}$$

2. Glavna generacijska petlja:

a) uzorkovanje vrijednosti (potomci)
- izvlačenje potomaka iz prostora pretraživanja po normalnoj razdiobi

b) evaluacija i rekombinacija potomaka (izbor budućih roditelja)
- svaki potomak se evaluira, a zatim se bira najboljih μ potomaka
-najbolje jedinke se usrednjuju pomoću težinske funkcije i njihova sredina se postavlja kao nova srednja vrijednost

c) promjena ostalih parametara s obzirom na najbolje jedinke iz skupa potomaka
- evolucijski put veličine koraka mijenja se prema formuli:

$$p_\sigma = (1 - c_\sigma)p_\sigma + \sqrt{1 - (1 - c_\sigma)^2}\sqrt{\mu_w}C^{-\frac{1}{2}}\frac{m_{novi} - m_{stari}}{\sigma_{stari}} \tag{19}$$

- promjena veličine koraka ovisi o tome kakav je njen evolucijski put (ako je duljina evolucijskog puta velika, povećat će se i veličina koraka, a ako je duljina mala, smanjiti će se i veličina koraka) i računa se po formuli:

$$\sigma_{novi} = \sigma_{stari} exp(\frac{c_\sigma}{d_\sigma}(\frac{||p_\sigma||}{E||N(0,1)||} - 1)), \tag{20}$$

pri čemu je d_σ faktor prigušenja i vrijednost mu je približna nuli, a očekivanje normalne razdiobe može se računati kao:

$$E||N(0,1)|| \approx \sqrt{n}(1 - \frac{1}{4n} + \frac{1}{21n^2}) \tag{21}$$

- evolucijski put kovarijantne matrice pamti promjene, tj. smjer putovanja kovarijantne matrice, a promjenu tog evolucijskog puta računamo prema formuli:

$$p_c = (1 - c_c)p_c + \sqrt{1 - (1 - c_c)^2}\sqrt{\mu_w}\frac{m_{novi} - m_{stari}}{\sigma_{stari}} \tag{22}$$

- promjena kovarijantne matrice se može napraviti dodavanjem triju različitih matrica: matricom ranga jedan, matricom ranga μ te negativnom matricom; kada se koristi samo matica ranga jedan, stara vrijednost kovarijantne matrice se malo smanji, da bi se za tu smanjenu vrijednost mogla dodati nova matrica $(p_c p_c^T)$ [11]:

$$C = (1 - c_1)C + c_1 p_c p_c^T \tag{23}$$

- kod velikih populacija se olakšava konvergencija ako se koristi i matrica ranga μ:

$$C = (1 - c_1 - c_\mu)C + c_1 p_c p_c^T + c_\mu \sum_{i=1}^{\mu} w_i y_{i:\lambda} y_{i:\lambda}^T, \tag{24}$$

gdje je $y_{i:\lambda} = \frac{x_{i:\lambda} - m_{stari}}{\sigma_{stari}}$ (25)

- kod tipa A-CMA-ES, formula za računanje kovarijantne matrice se dodatno proširuje tako da uzima u obzir loše jedinke:

$$C = (1 - c_1 - c_\mu + \alpha c_{neg})C + c_1 p_c p_c^T + (c_\mu - (1-\alpha)c_{neg}) \sum_{i=1}^{\mu} w_i y_{i:\lambda} y_{i:\lambda}^T - c_{neg} \sum_{i=\lambda-\mu+1}^{\lambda} w_i z_{i:\lambda} z_{i:\lambda}^T, \quad (26)$$

pri čemu α označava faktor kojim se nadoknađuje gubitak varijance, c_{neg} jačinu utjecaja negativne matrice na kovarijantnu matricu, a vektor z_i je vektor nasumičnih vrijednosti distribuiranih po jediničnoj normalnoj razdiobi

3. Povrat najbolje jedinke (vrijednosti)

- nakon što se zadovolji postavljeni uvjet prekida, CMA-ES vraća najbolju jedinku kao pronađeni optimum funkcije

5. Provedena ispitivanja

5.1. Ispitivanje utjecaja preciznosti

Funkcija cilja je broj krivih odgovora ($response$ bitova). Stoga, optimum je nula, a takvo rješenje postigne se kad nema pogrešno dobivenih odgovora. Radi se o problemu minimizacije (traži se optimizacija PUF-a u smislu da se minimizira broj pogrešno dobivenih $response$ bitova, dakle najbolje rješenje je upravo nula).
Utjecaj preciznosti prikaza na učinkovitost pronalaženja rješenja ispitan je algoritmom *GenHookeJeeves* i PUF-om dimenzije 16. *GenHookeJevees* algoritam ima parametar $precision$ čijim smanjivanjem povećavamo razlučivost algoritma. Za svaku različitu konfiguraciju parametara algoritam je pokrenut deset puta. Početno zadana preciznost je iznosa 1, a smanjuje kada niti jedno od deset pokretanja ne dođe do nule.
Parametri algoritma koji se ne mijenjaju su "delta"=1, "localonly"=0, a "precison" poprima redom vrijednosti 1, 0.5, 0.2, 0.1, 0.05, 0.02, 0.01, 0.005 itd. U $FloatingPoint$ genotipu postavljene su donja i gornja granica na -1 i 1, a dimenzija na 17. Razlog tome je što je veličina vektora kašnjenja jednaka broju razina uvećanoj za jedan. Dakle, ako želimo broj razina 16 (što znači da je PUF veličine 1x16), postavit ćemo parametar "dimension" u $FloatingPoint$ genotipu na 17.

Dio datoteke s parametrima izgleda ovako:
<Registry>
<Entry key="randomizer.seed">0</Entry>
<Entry key="puf.debug">1</Entry>
<Entry key="puf.chains">1</Entry>
<Entry key="puf.responses">128</Entry>
<Entry key="puf.combinations">0</Entry>
<Entry key="randomizer.seed">0</Entry>
<Entry key="population.size">50</Entry>
<Entry key="mutation.indprob">0.5</Entry>

<Entry key="term.eval">2500000</Entry>
<Entry key="term.stagnation">500000</Entry>
<Entry key="term.fitnessval">0</Entry>
<Entry key="log.frequency">50</Entry>
<Entry key="log.filename">log.txt</Entry>
<Entry key="batch.repeats">10</Entry>
<Entry key="batch.statsfile">stats.txt</Entry>
</Registry>

Parametar $puf.responses$ se povećava tako dugo dok algoritam uspijeva doći do nule. Kad algoritam to ne uspije (u deset pokretanja), onda mijenjamo parametre tako da fiksiramo zadnji broj na koji je bio postavljen $puf.responses$, a preciznost smanjimo i ponovno pokrenemo program.
Opisanim postupkom željelo se utvrditi kakav učinak ima preciznost s obzirom na broj $challenge - response$ parova. Dobiveni rezultati prikazani su na grafu. Priložena je i tablica koja povezuje postavljenu preciznost s vrijednostima parametra $puf.response$ na kojima je testirana i za koje je algoritam uspio doći do nule.

Slika 5.1: Preciznost na PUF-u veličine 1x16

Tablica 5.1: Učinkovitost preciznosti za PUF 1x16

preciznost	puf.responses
1	1, 2, 4, 8, 16, 32, 48, 64, 96, 128, 160, 192, 256, 336, 424, 468, 490, 512, 768
0.5	896, 1024, 1536, 2048
0.005	4096

Na y-osi prikazana je preciznost, a na x-osi broj *challenge-response* parova. Vrijednost funkcije koju očitavamo na y-osi prikazuje najveću potrebnu preciznost (najmanju potrebnu razlučivost) za pripadnu veličinu $puf.responses$ parametra. Na primjer, možemo očitati da je za zadanu preciznost 1 moguće doći do nule za otprilike maksimalno 768 PUF odgovora ($responses$). Za 896 odgovora, preciznost 1 više nije dovoljna. Ako preciznost postavimo na 0.5, moguće je doći do nule za veći broj odgovora (do otprilike 2048), uključujući i 896 za koji preciznost iznosa 1 nije bila dovoljna. Za vrijednosti parametra $puf.responses$ veće od toga, nije dovoljna ni preciznost 0.5.

Tablica 5.2: Učinkovitost preciznosti za PUF 1x8 i 1x12

preciznost	puf.responses
1	1024, 2048, 4096, 8192

Za PUF veličine 1x8 i za PUF veličine 1x12, dobiveno je rješenje nula za broj odgovora do 8192. Jedina razlika je što konfiguracija s veličinom 1x12 nije za svako pokretanje došla do nule.

5.2. Ispitni primjer: PUF veličine 1x64

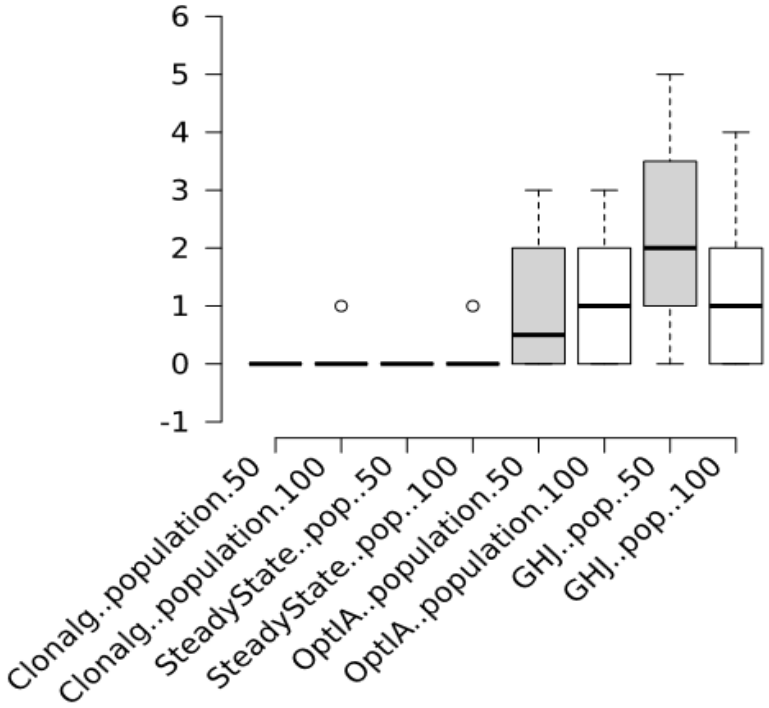
Na primjeru PUF-a veličine 1x64 isprobana je učinkovitost algoritama *Clonalg, SteadyState, OptIA, GenHookeJevees* i CMA-ES. Algoritam CMA-ES isproban je na sve tri implementirane inačice i dobiveni rezultati za CMA-ES algoritam prikazani su posebno u poglavlju 5.3. U ovom poglavlju obrađeni su rezultati dobiveni za preostala četiri algoritma.

Svaka konfiguracija pokrenuta je dvadeset puta. Kao uvjet prekida postavljen je maksimalan broj evaluacija (parametar $term.eval$), za većinu algoritama na pet milijuna. Parametri koji su se mijenjali za svaku konfiguraciju su $puf.responses$ i veličina populacije ($population.size$).

Algoritmi su najprije ispitani za fiksirani parametar *puf.responses*=128, a veličina populacije postavljena je na 50 za prvih dvadeset pokretanja, a zatim na 100 za drugih dvadeset pokretanja. Zatim su odabrane najbolje konfiguracije iz prošlog ispitivanja i parametar $puf.responses$ fiksiran je na iznos 256. Cilj je pronaći najbolju konfiguraciju pojedinog algoritma za zadani problem, a zatim najbolje konfiguracije za svaki algoritam međusobno usporediti.

Bitno je napomenuti koji su parametri pojedinih algoritama fiksirani u svim testiranjima:
za *Clonalg: n=50, beta=1.0, c=0.2, d=0.0, cloningVersion=proportional, selectionScheme=CLONALG1*
za *SteadyState: tsize=3*
za *OptIA: dup=5, c=0.2, tauB=100*
za *GenHookeJevees: precision=0.01, delta=0.3, localonly=0.*

Prvo su prikazani rezultati za *puf.responses*=128 (na slici 5.2) i usporedba u ovisnosti o veličini populacije za navedene algoritme.



Slika 5.2: Usporedba za $puf.responses = 128$

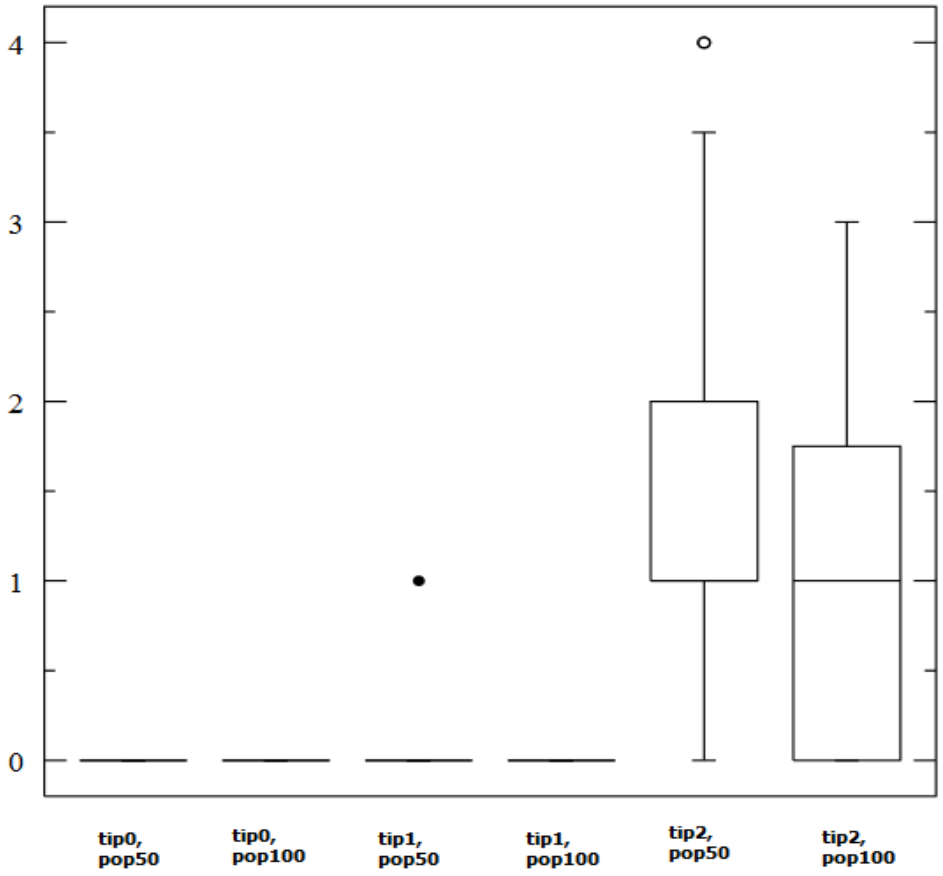
Iz danog prikaza može se vidjeti da su algoritmi *Clonalg* i *SteadyState* za veličinu populacije 50 svaki puta uspjeli doći do nule. Za populaciju veličine 100 jednom nisu uspjeli doći do nule i u tom slučaju dobrota najbolje jednike bila je 1 za oba algoritma. Algoritam *OptIA* bolje radi za veličinu populacije 50 (vrijednost medijana bolje odgovara funkciji minimizacije). Za *GenHookeJevees* bolje rješenje je veličina populacije 100.

Usporedba algoritama za veličinu *puf.responses*=256 napravljena je u sljedećem potpoglavlju, zajedno s algoritmom CMA-ES.

5.3. Varijacije CMA-ES algoritma na problemu PUF-a veličine 1x64

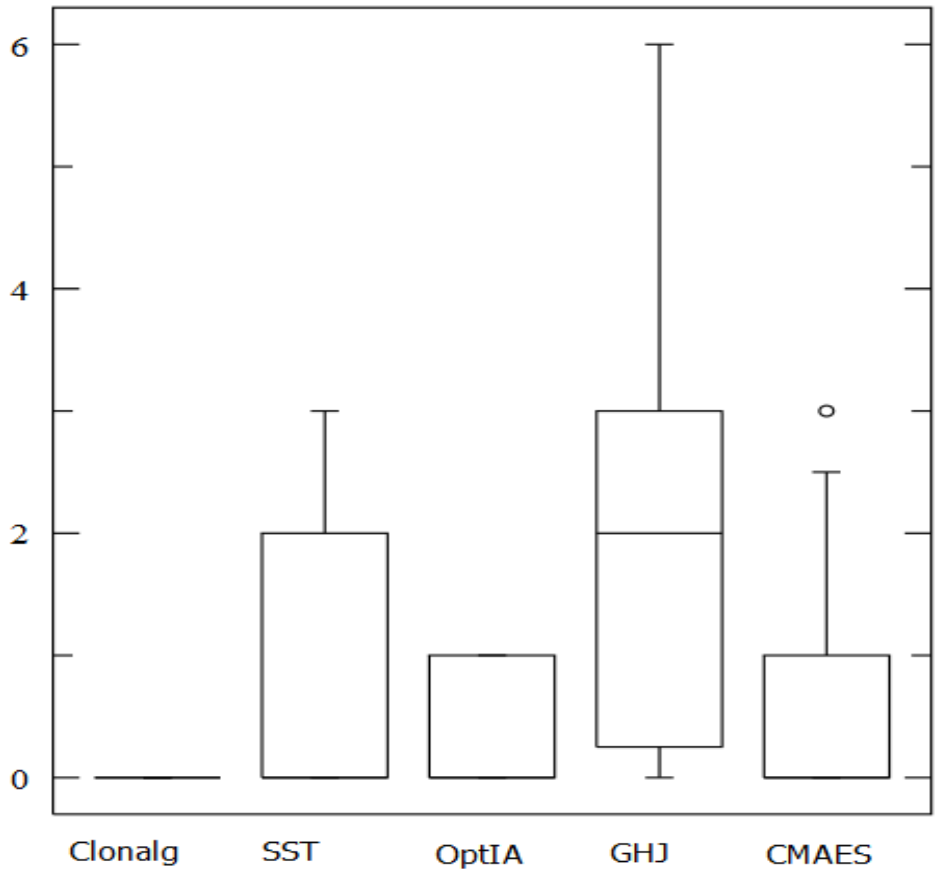
Provedena su testiranja na tri inačice CMA-ES algoritma, čija je implementacija objašnjena u prethodnom poglavlju. Kao što je već rečeno, tip algoritma bira se parametrom $type$ koji može biti postavljen na "0" za tip $NORMAL$, "1" za tip $IPOP$ i "2" za tip $ACTIVE$. Ostali parametri algoritma postavljeni su na *weightType=2* i *mu=3*.

Algoritam je testiran na PUF-u veličine 1x64 na isti način kao i ostali algoritmi, dakle mijenjajući veličinu populacije i broj $response$ bitova. Graf dobivenih rezultata za *puf.responses*=128 prikazan je na slici 5.3.



Slika 5.3: CMA-ES za puf.responses=128

Rezultati za *puf.responses*=256 mogu se vidjeti na slici 5.4. Slika 5.4 uključuje usporedbu najboljih konfiguracija isprobanih algoritama s obzirom na veličinu populacije. Za *Clonalg, SteadyState, OptIA* to je veličina populacije 50, a za *GenHookeJevees* veličina populacije 100. Za CMA-ES tip odabran je $IPOP$ s veličinom populacije 50.

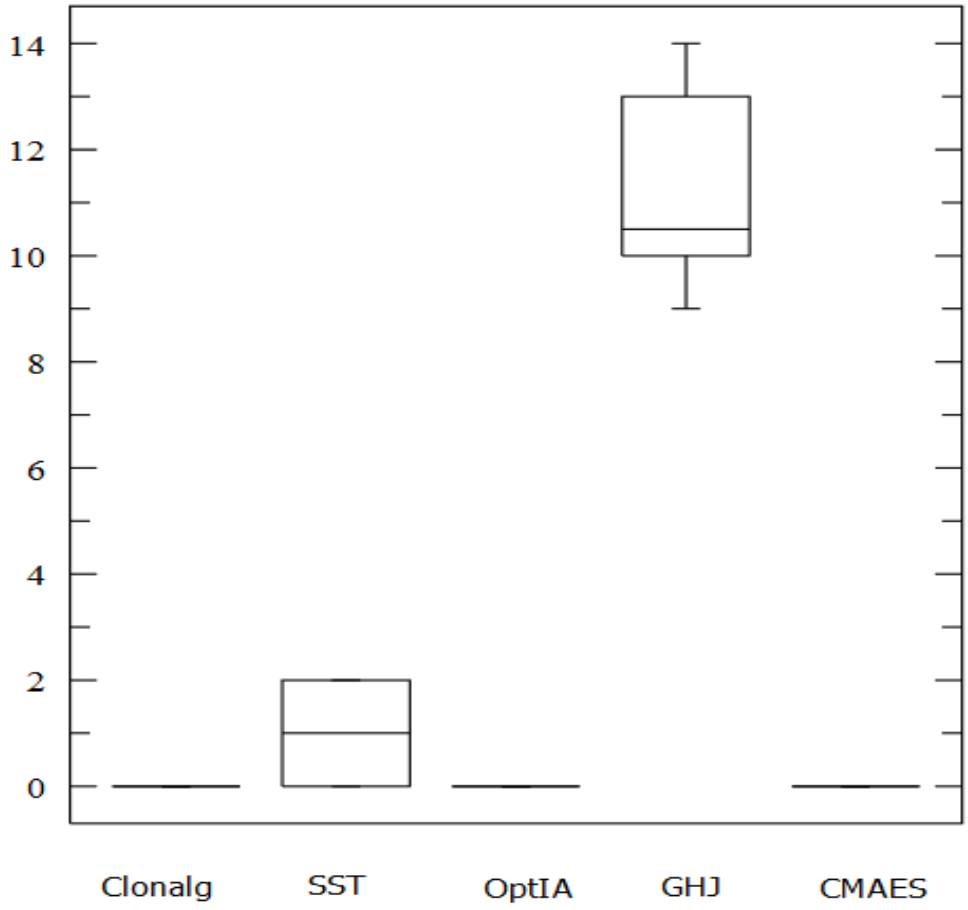


Slika 5.4: Usporedba za $puf.responses = 256$

Najveća promjena može se zapaziti na algoritmu *SteadyState* koji se vidljivo pogoršao zbog povećanja broja *puf.responses*.

5.4. Ispitni primjer: PUF veličine 4x64

Najbolja rješenja iz prethodnih testiranja, koja su se provodila na PUF-u veličine 1x64, odabrana su za testiranje na PUF-ovima većih dimenzija. Dimenzija problema se povećala tako da se povećao broj lanaca, što znači da se mijenjao parametar $puf.chains$. Postavljen je na 4 (za dimenziju 4x64). Dimenzija u $FloatingPoint$ genotipu postavljena je na umnožak broja lanaca i broja 65 (zbog zadane veličine PUF-a).
Svaki algoritam pokretao se deset puta s parametrom $puf.responses$ postavljenim na 128. Rezultati su prikazani na slici 5.5.



Slika 5.5: Usporedba za PUF 4x64

GenHookeJevees se pokazao daleko najlošijim algorimom na ovom problemu. *Clonalg, OptIA, CMAES* su svaki puta došli do nule, što je malo bolji rezultat nego na ispitivanju s $puf.responses$ postavljenim na 256. *SteadyState* pokazuje podjednako dobre rezultate na ovom ispitnom primjeru i na primjeru manje dimenzije s većim brojem $puf.responses$ (pokazano u prošlom poglavlju na dimenziji PUF 1x64).

6. Zaključak

Ispitivanje učinkovitosti zadanih algoritama (*Clonalg, SteadyState, OptIA, GenHookeJevees*, CMA-ES) provodilo se na nekoliko razina. Prva od razina ispitivanja sadržavala je PUF dimenzije 1x16 kao ispitni primjer, koji je služio da se utvrdi utjecaj preciznosti algoritma *GenHookeJevees* na učinkovitost optimizacije ispitnog primjera. Pokazalo se do kojih se dimenzija može povećavati broj $reponse$ bitova za određenu preciznost, a da se pronađe zadovoljavajuće rješenje. Rezultati tih testova objašnjeni su detaljnije u prethodnom poglavlju o ispitivanju.

PUF dimenzije 1x64 poslužio je kao sljedeći ispitni primjer. Isprobani su svi navedeni algoritmi, i to tako da se mijenjala veličina populacije (50, 100), a zatim i broj $response$ bitova (sa 128 na 256). Usporedba svih ispitnih konfiguracija prikazana je u prethodnom poglavlju. Za algoritam *Clonalg* i *SteadyState* nema velike razlike u ovisnosti o veličini populacije (oba algoritma u većini slučajeva dolaze do nule), ali ipak se kao malo bolje rješenje pokazalo veličina populacije 50 i broj $response$ bitova 128. Za algoritam *OptIA* vidljivo je najbolje rješenje ono s veličinom populacije 50 i brojem $response$ bitova 128. Kao najbolja konfiguracija za algoritam *GenHookeJevees* odabrana je ona s veličinom populacije 100 i brojem $response$ bitova 128.
U međusobnoj usporedbi algoritama može se vidjeti da su se na ovom problemu najboljima pokazali *Clonalg, SteadyState* i prva dva tipa CMA-ES algoritma, a dosta lošijima u odnosu na njih *OptIA* i *GenHookeJevees*.
Najbolja rješenja prethodnog testiranja za pojedine algoritme odabrana su kako bi se ispitala njihova učinkovitost na ispitnom primjeru PUF-a dimenzije 4x64. Rezultati su prikazani grafički i komentirani u poglavlju o provedenim ispitivanjima. Usporedba međusobnog odnosa pojedinih algoritama ostala je slična kao na problemu dimenzije 1x64, osim što je algoritam *OptIA* dao bolje rezultate u odnosu na dimenziju 1x64.

Svi algoritmi na problemima relativno većih dimenzija pokazuju znatno usporenje u radu i manju učinkovitost u odnosu na probleme malih dimenzija. Veličina

populacije, kad se fiksira na vrijednost 50 ili 100, na neke algoritme ima veći utjecaj (*OptIA, GenHookeJevees*) u odnosu na ostale. Općenito, značajniji je utjecaj parametra $puf.responses$ jer za povećanje vrijednosti tog parametra s 128 na 256 algoritmi imaju lošiju učinkovitost i rjeđe dođu do optimalnog rješenja. Do istog zaključka može se doći ako se promatra prvo testiranje - preciznost na dimenziji 16. Kako se povećava broj $response$ bitova, tako se smanjuje učinkovitost.

7. Literatura

[1] Maes, R., Physically Unclonable Functions: A Study on the State of the Art and Future Research Directions, Part of the series Information Security and Cryptography pp 3-37, 2010

[2] Maes, R., Physically Unclonable Functions: Constructions, Properties and Applications, 2013

[3] Bolotnyy, L., Robins, G., Physically Unclonable Function-Based Security and Privacy in RFID Systems; Fifth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom'07)

[4] Suh, G.E., Physical unclonable functions for device authentication and secret key generation; DAC '07 Proceedings of the 44th annual Design Automation Conference, San Diego, California, Pages 9-14

[5] Beckmann, N., Potkonjak, M., Hardware-Based Public-Key Cryptography with Public Physically Unclonable Functions; Lecture Notes in Computer Science, vol 5806. Springer, Berlin, Heidelberg, 2009.

[6] Krček, M., Optimizacija PUF-ova, seminar, FER, 2016.

[7] Ayat, M., Atani, R.E., Mirzakuchaki, S., On design of PUF-based random number generators; International Journal of Network Security and Its Applications (IJNSA), Vol.3, No.3, May 2011

[8] Golub, M., prezentacija Evolucijsko računarstvo u okviru predmeta Neizrazito, evolucijsko i neuro-računarstvo, 2014./15.

[9] Thomas, W., Global Optimization Algorithms,
http://www.it-weise.de/projects/book.pdf, 24.10.2007.

[10] Čeri, M., Malović, I., Evolucijske strategije, Projekt, FER, 2007.

[11] Ćosić, M., Stohastička numerička optimizacija u okruženju za evolucijsko računanje, Završni rad, FER, 2012.

[12] Concept of directional optimization in CMA-ES:
https://en.wikipedia.org/wiki/CMA-ES

Modeliranje logičkih funkcija bez mogućnosti kopiranja evolucijskim algoritmima

Sažetak

Modeliranje logičkih funkcija bez mogućnosti kopiranja koristi se u uređajima za sigurnu komunikaciju kao metoda generiranja jedinstvenih kriptografskih ključeva. Za potrebe ovog rada ostvaren je programski sustav za optimizaciju modela zadanih funkcija temeljen na evolucijskim algoritmima i algoritmu evolucijske strategije. Proučavani su povezani optimizacijski problemi i testirana učinkovitost nekoliko evolucijskih algoritama i algoritma evolucijske strategije na zadanim modelima. Ispitivana je učinkovitost preciznosti dobivenih modela s obzirom na veličinu polazne funkcije, preciznost prikaza i parametre optimizacijskih algoritama. Rezultati ispitivanja su analizirani i prikazani dijagramima i tablicama.

Ključne riječi: logičke funkcije bez mogućnosti kopiranja, evolucijski algoritmi, evolucijska strategija, optimizacija, preciznost

Modelling physically unclonable functions with evolutionary algorithms

Abstract

Modelling physically unclonable functions is used in safety communication devices as a method of generating unique cryptographic keys. For the purpose of this paper, a program system for optimization of the given function model based on evolution algorithms and evolution strategy algorithm has been implemented. Related optimization problems were studied and effectiveness of several evolutionary algorithms and evolution strategy algorithm was tested on given models. The efficiency of the precision of the obtained models was studied with regard to the size of the starting function, the precision of the display and parameters of the optimization algorithms. The test results are analyzed and presented in diagrams and tables.

Keywords: physically unclonable functions, evolutionary algorithms, evolution strategy, optimization, precision