Univerzitet Crne Gore
Elektrotehnički fakultet

Diplomski rad:

Genetički algoritmi

i njihova praktična primjena

Kandidat
Edin Salković
9016/01

Podgorica, septembar 2005.

UNIVERZITET CRNE GORE
ELEKTROTEHNIČKI FAKULTET
Broj _____
Podgorica, _____ 2005. godine

Genetički algoritmi

i njihova praktična primjena

- Diplomski rad -

Datum izdavanja rada:
Podgorica, _____ 2005. godine

Mentor

_____
Doc. dr Igor Đurović

Datum predaje rada:
Podgorica, _____ 2005. godine

Kandidat

_____
BSc. Edin Salković

IZJAVA O SAMOSTALNOM RADU

Ovim izjavljujem da sam diplomski rad uradio samostalno, uz pomoć mentora i navedene literature.

Kandidat

_____

Edin Salković

Diplomski rad je odbranjen dana _____ 2005. god. pred komisijom u sastavu:

1. _____ (predsjednik)
2. _____ (mentor)
3. _____ (član)

Sa ocjenom _____ (_____)

Sadržaj

Poglavlje 1

O genetičkim algoritmima

U ovom poglavlju se daje kratak uvod u genetičke algoritme, terminologiju vezanu za njih i opseg primjene. Kao izvor informacija korišteno je [1] i [2]. Trebalo bi da nakon ovog poglavlja čitalac stekne uvid u suštinu genetičkih algoritama i da može da počne da ih koristi u praksi. Zato je ovo poglavlje „inženjerske" prirode i šturo teorijom.

1.1 Uvod

Genetički algoritmi (eng. *Genetic Algorithms*) predstavljaju porodicu algoritama koji se služe nekim od genetičkih principa koji su prisutni u prirodi, a da bi riješili određene računske probleme. Ti prirodni principi su: nasljeđivanje, ukrštanje, mutacije, zakon jačeg (eng. *survival of the fittest*), migracije itd.

Ovi algoritmi se mogu koristiti za rješavanje raznoraznih klasa problema, jer su prilično opšte prirode. Načelno, oni se koriste kod problema optimizacije — nalaženja optimalnih parametara nekog sistema. Kod „klasičnih" adaptivnih algoritama sve ide dobro ako imamo sistem sa jednim optimumom (tj. jednim ekstremumom — maksimumom ili minimumom — parametra koji želimo da optimizujemo). Problemi se javljaju kod sistema sa više lokalnih ekstremuma, gdje se može desiti da algoritam „zakuje" oko nekog lokalnog ekstremuma, koji može biti prilično daleko od globalnog ekstremuma. Genetički algoritam pretražuje stohastički razne dijelove prostora pretrage i stoga je veća vjerovatnoća da će „ubosti" globalni ekstremum. Prostor pretrage može biti n-dimenzioni.

U užem smislu, pojam genetički algoritam se odnosi samo na model koji je uveo John Holland i njegovi studenti (DeJong itd.). Veliki dio teorije genetičkih algoritama se odnosi upravo na taj model. Mi ćemo taj model zvati **kanonički genetički algoritam**. Nešto malo teorije vezane za ovaj model je dato u poglavlju 2. U širem smislu, genetički algoritam je svaki algoritam koji je zasnovan na nekoj populaciji i operatorima selekcije, rekombinacije i mutacije, koji služe za dobijanje novih tačaka u prostoru pretrage.

Veliki problem genetičkih algoritama je teorijske prirode — nedostatak matematički rigoroznih dokaza da genetički algoritmi zaista rade ono što rade. U

stvari, nove „varijacije" genetičkih algoritama se javljaju vrlo često, te je skoro nemoguće za sve njih postaviti teorijsku osnovu, a izgleda da nema puno ni potrebe; naime, obično se kvalitet nekog novog tipa genetičkog algoritma ocjenjuje empirijski, tako što se eksperimentalno testira na nekom problemu optimizacije.

1.2 Jednostavni genetički algoritam i terminologija

Terminologija koja se koristi kod genetičkih algoritama se generalno zasniva na onoj koja se koristi u biologiji, mada se u skorije vrijeme možda i više koriste „tehnički" termini.

Individue, koje predstavljaju trenutne aproksimacije rješenja, se kodiraju kao **hromozomi** (**stringovi**), ili obrnuto: hromozom je kodirana predstava individue. Od **roditelja** (eng. *parents*) reprodukcijom nastaje **potomstvo** (eng. *offspring*). Još jedan naziv za stringove je **genotip**. Dakle, svakom genotipu odgovara jedinstveni **fenotip**, što je samo drugi naziv za individuu. Hromozom se sastoji iz više **gena**. Gen je kodirana varijabla našeg konkretnog problema. Stringovi predstavljaju niz karaktera (**alela**) nekog **alfabeta**. Često je taj alfabet binarni — $\{0, 1\}$ —, ali se mogu koristiti predstave koje koriste prirodne, realne ili neke druge alfabete. Na primjer, ako imamo problem sa dvije promjenljive x_1 i x_2, možemo napraviti strukturu hromozoma na način prikazan sljedećim primjerom:

$$\underbrace{1011010011}_{x_1}\underbrace{010111010100101}_{x_2}$$

Varijable x_1 i x_2 su predstavljene kao dva gena. x_1 je kodirano sa 10 bita, a x_2 sa 15 bita. Razlika u dužini bita može biti uslijed toga što x_2 uzima vrijednosti iz nekog većeg skupa, ili ih uzima iz istog skupa kao i x_1, ali sa većom preciznošću.

Interesantno je da se termin **populacija** (ili **kolonija**) ne koristi za skup individua, već za skup njihovih kodiranih predstava — hromozoma. Ovo je stoga što se sve operacije genetičkog algoritma sprovode nad hromozomima, osim **procjene** (**evaluacije**) njihovog kvaliteta (**fitnesa**), koja se mora obaviti nad dekodiranim genotipom. Fitnes (eng. *fitness*) se dobija pogodnom transformacijom rezultata evaluacije.

Na slici 1.1 je data šema tipičnog genetičkog algoritma. Mogući pseudo-kod koji bi odgovarao dotičnoj šemi je dat na slici 1.2, [1].

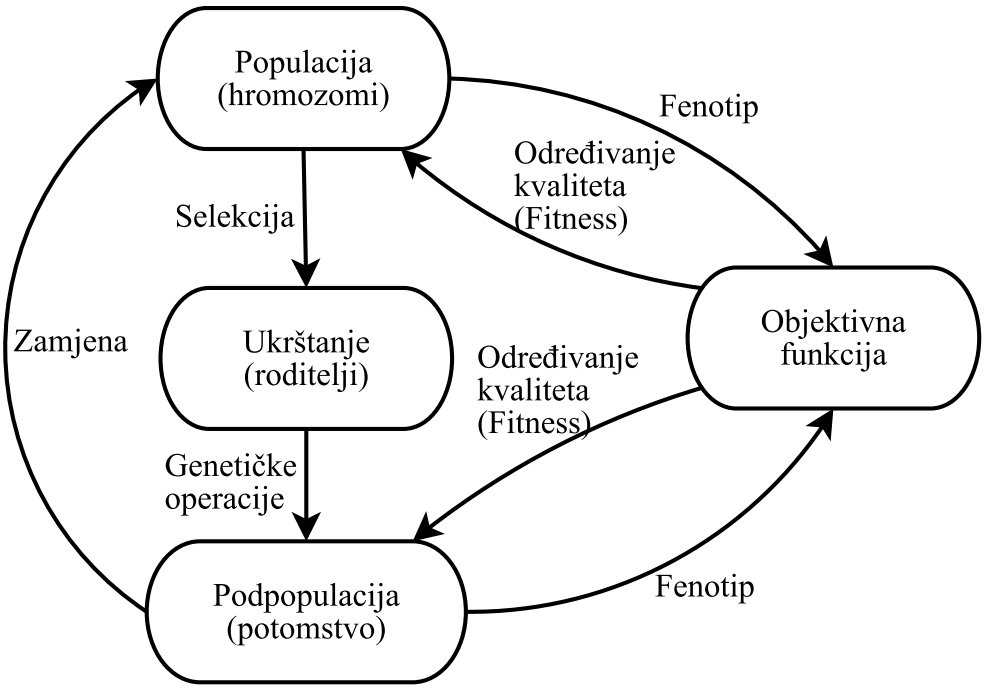
Slijedi kratak opis svih faza prikazanih pseudo-kodom:

1. Slučajno izabrati inicijalnu populaciju $X(0) = (x_1, x_2, x_3, \ldots, x_N)$, gdje je N početni broj hromozoma.

2. Izračunati fitnes $F(x_i)$ za svaki hromozom x_i u tekućoj populaciji $X(t)$.

3. Kreirati nove hromozome $X_r(t)$ ukrštanjem tekućih (roditeljskih) hromozoma primjenjujući **mutacije** i **rekombinacije**.

Slika 1.1: Šema tipičnog genetičkog algoritma



Slika 1.2: Pseudo-kod za tipični genetički algoritam

```
procedure GA
begin
    t = 0;
    inicijalizuj X(t);
    ocijeni X(t);
    radi do nekog uslova
    begin
        t = t + 1;
        odaberi X(t) iz X(t-1);
        reprodukuj parove u X(t);
        ocijeni X(t);
    end
end
```

4. Izbrisati određen broj hromozoma iz populacije da bi se napravilo mjesto za nove (**umiranje hromozoma**).

5. Izračunati fitnes novih hromozoma u populaciji $X_r(t)$.

6. Postaviti $t = t + 1$. Ako nije zadovoljen odgovarajući kriterijum ići na korak 3, a ako jeste obustaviti petlju i uzeti najbolji hromozom. Obustavljanje petlje može biti nakon postizanja zadovoljavajućeg kvaliteta ili broja iteracija.

Jedna iteracija algoritma se naziva **epohom**. Obično se u jednoj epohi vrši samo jedna reprodukcija.

1.3 Kodiranje i problemi optimizacije

Obično postoje samo dvije komponente genetičkog algoritma koje zavise od konkretnog problema: **kodiranje problema** i **funkcija procjene** (**evaluacije**) koja se još naziva i **objektivnom**, **kriterijumskom**.

Kao što je već navedeno, mi obično želimo pomoću genetičkog algoritma da izvršimo optimizaciju nekih parametara. Posmatramo neku funkciju sa M parametara:

$$f(X_1, X_2, \ldots, X_M)$$

za koju želimo da nađemo ekstremum. Parametri mogu biti raznovrsni: temperatura, pritisak, geografska koordinata, brzina, ili pak razni ekonomski parametri itd. Na primjer, kada bi u pitanju bili neki ekonomski parametri rezultujuća funkcija bi mogla da bude profit, pa bi nam genetički algoritam poslužio za maksimizovanje profita.

Obično se genetički algoritmi koriste kod nelinearnih problema, a to često znači da će parametri biti i međusobno zavisni, a to dalje znači da se i algoritam mora prilagoditi konkretnom problemu, na primjer implementacijom nekog nestandardnog načina ukrštanja. Interakcija između parametara se naziva **epistazom**.

Parametri se obično kodiraju pomoću binarnih brojeva, mada je ponekad ipak bolje i jednostavnije koristiti realne ili prirodne brojeve. Način kodiranja i nadovezivanja varijabli je prikazan na strani 3. Zanimljivo je da se kodiranjem lakše prikazuju kontinualne nego diskretne vrijednosti. Kako? Recimo da hoćemo da kodiramo skup realnih brojeva od 0 do 1. Ovo možemo odraditi sa proizvoljnom preciznošću, uzimajući da oblast diskretizacije bude neka oblast od 0 do 2^K, gdje je K broj bita koji smo dodijelili varijabli koju kodiramo. Ako nam je dovoljna preciznost od 1024 tačke uzećemo $K = 10$. Definišemo korak:

$$P = \frac{U_{max} - U_{min}}{2^K - 1}$$

Vrijednost U_{min} se kodira kao $00\ldots00$ (K nula), dok se $U_{min} + P$ kodira kao $00\ldots01$.

Međutim, ako kodiramo diskretnu promjenljivu koja uzima vrijednosti od 0 do 800, praktično moramo uzeti $K = 10$, ali kako, prilikom evaluacije, protumačiti hromozome između 800 i 1024? Rješavanje ovog pitanja je stvar dizajna funkcije procjene koja može hromozome iz tog raspona protumačiti na više načina:

- Mogu biti eliminisani.
- Mogu im biti određeni veoma slabi fitnesi (tako da ipak utiču na pretraživanje prostora rješenja).
- Nekim vrijednostima parametara se mogu pripisati po dva hromozoma umjesto jednog.

Takođe, postoji standardno binarno kodiranje i Gray-ovo kodiranje. Ovo drugo je često pogodno jer smanjuje Hamming-ovo rastojanje između susjednih vrijednosti, odnosno čini ga konstantnim. Prevelika Hamming-ova distanca može rezultirati slabom, pristrasnom konvergencijom, što dalje može rezultovati nenalaženjem globalnog ekstremuma.

Funkcija evaluacije je najčešće data samim problemom i treba da bude relativno brza jer se provodi nad svakim članom populacije, i u svakoj generaciji. Na primjer, ako je potreban 1 sat za evaluaciju jednog hromozoma, onda treba više od jedne godine za 10000 evaluacija. To bi bilo oko 50 generacija za populaciju od samo 200 stringova. Kao primjer funkcije evaluacije često se navodi suma kvadrata, odnosno minimizacija te sume, koja služi i kao (jedna od) test funkcija za provjeru ispravnosti nekog genetičkog algoritma:

$$f(x) = \sum_{i=1}^{N} x_i^2 \qquad -100 \leq x_i \leq 100 \qquad \min(f) = f(0, 0, \ldots, 0) = 0$$

N je proizvoljan broj, ali radi jednostavnosti i brzine računanja ne treba uzimati prevelike vrijednosti.

1.4 Pojedine etape genetičkih algoritama

1.4.1 Odabir (selekcija)

U prirodi bolji primjerci imaju veće šanse za reprodukcijom nego oni „manje dobri". Kako to primijeniti na stringove, tj. kako odabrati bolje? Preko funkcije evaluacije. Stringovi koji imaju veći fitnes dobijaju veću šansu za ukrštanjem. Ranije smo naveli da se fitnes dobija transformacijom rezultata funkcije evaluacije.

Ponekad se ova dva pojma, fitnes i evaluacija, koriste sa istim značenjem. Međutim, treba praviti razliku, jer najčešće „sirove" vrijednosti funkcije evaluacije ne mogu biti primijenjene direktno u genetičkom algoritmu. Kod kanoničkog genetičkog algoritma, fitnes se definiše kao:

$$f(x_i) = \frac{u(x_i)}{u}$$

i predstavlja odnos sirove vrijednosti performanse pojedinca i performanse cjelokupne populacije. $u(x_i)$ je funkcija evaluacije i-tog hromozoma.

Međutim, ovakva procjena fitnesa pada u vodu kada funkcija evaluacije daje i negativne vrijednosti. Stoga se često koristi **linearno skaliranje i pomjeranje** (pomoću **offset**-a). Tada imamo:

$$f(x_i) = au(x_i) + b$$

gdje je a faktor skaliranja koji je pozitivan ako treba maksimizovati rezultat, a negativan ako ga treba minimizovati. b je offset (pomjeraj), i on se postavlja tako da vrijednosti fitnesa budu pozitivne.

Posmatrajmo sljedeći primjer koji ilustruje pomenutu tehniku. Neka je data funkcija evaluacije:

$$u(x, y) = (x - 7)^2 + (y - 3)^2$$

i neka su x i y cijeli brojevi u intervalu od 1 do 7. U tabeli 1.1 su dati fitnesi za 4 proizvoljno izabrana inicijalna hromozoma sa odgovarajućim trobitnim kodiranjem svake od osobina. Uzeli smo da je $a = 1$, a $b = 0$, jer je suma kvadrata uvijek

Tabela 1.1: Primjer linearnog skaliranja i pomjeranja

Br. hromozoma	String	(x, y)	$f(x, y)$
1	100001	(4,1)	13
2	001100	(1, 4)	37
3	110010	(6, 2)	2
4	000100	(0, 4)	50

pozitivna, pa nema potrebe za ofsetom.

Postoji i druga tehnika koje se naziva **linearnom normalizacijom** gdje se hromozomi sortiraju od najgoreg ka najboljem i to tako da najgori ima vrijednost 1 (ili 0), sljedeći 2, sljedeći 3 itd. Kada to primijenimo na slučaj iz prethodnog primjera dobićemo rezultate kao u tabeli 1.2.

Tabela 1.2: Primjer linearne normalizacije

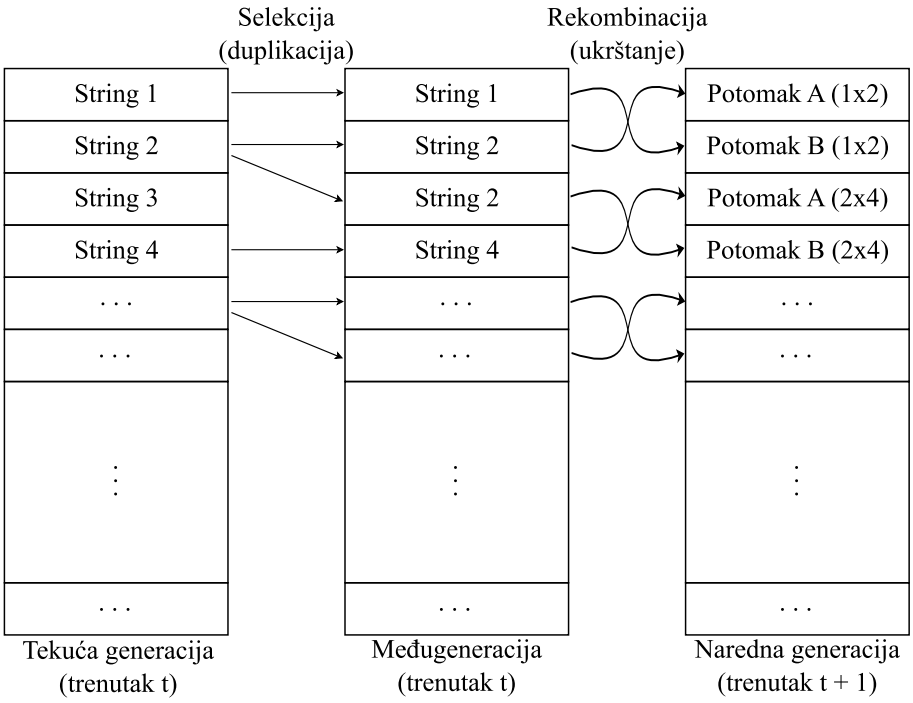
Br. hromozoma	String	(x, y)	$f(x, y)$
1	100001	(4,1)	2
2	001100	(1, 4)	3
3	110010	(6, 2)	1
4	000100	(0, 4)	4

Gore pomenuti postupci linearizacije su, međutim, podložni brzoj, bolje reći preranoj konvergenciji. Naime, broj potomaka u narednim generacijama je otprilike proporcionalan fitnesu pojedinca. Pošto ne postoji ograničenje fitnesa

pojedinca u tekućoj generaciji, postoji velika vjerovatnoća da jedinke sa visokim fitnesom u prvim generacijama prosto zadominiraju reprodukcijom i time izazovu preranu konvergenciju prema, najvjerovatnije, neoptimalnim rješenjima.

U fazi selekcije, neposredno pred rekombinaciju, nastaje svojevrsna **međugeneracija** (eng. *intermediate generation*). To je, praktično, pomoćna generacija u jednoj iteraciji genetičkog algoritma. Ovo je ilustrovano slikom 1.3, [2].

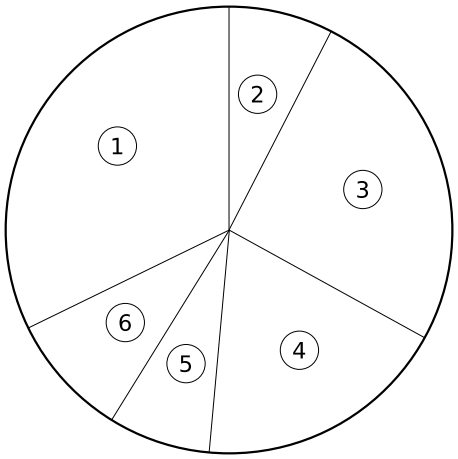
Slika 1.3: Prikaz generacija i etapa u jednoj iteraciji



Prvo ćemo objasniti kako se dobija međugeneracija iz **tekuće generacije** (eng. *current generation*). Nakon računanja fitnesa za sve jedinke na neki od pomenutih načina, izvršava se selekcija. Kod kanoničkog genetičkog algoritma, vjerovatnoća da neki string iz tekuće generacije bude umnožen u više primjeraka i smješten u međugeneraciju je proporcionalna njegovom fitnesu. Postoje mnogobrojni načini za obavljanje selekcije.

Obično se kod objašnjavanja selekcije koristi pojam ruleta — odnosno **rulet-točka**. Populacija se posmatra kao da je mapirana na rulet-točak, otprilike kao na slici 1.4. Veći dijelovi „kolača" pripadaju onim stringovima koji imaju veći fitnes. Kada se formira rulet-točak, obavlja se njegovo „obrtanje" onoliko puta koliko će biti jedinki u međugeneraciji. Pri svakom obrtanju, odabira se jedan hromozom pomoću „pokazivača" koji je fiksiran na rulet-točku. Ovaj način selekcije se zove **stohastičko odabiranje sa zamjenom** (eng. *stochastic sampling with replacement*), i njegov glavni problem je to što, teoretski, jedan hromozom (bilo koji sa fitnesom većim od nule) može popuniti čitavu međugeneraciju. **Stohastičko**

Slika 1.4: Rulet-točak sa šest jedinki



odabiranje sa djelimičnom zamjenom (eng. *stochastic sampling with partial replacement*) ovo prevazilazi tako što svaki put kada neka jedinka bude odabrana smanjuje njen udio u „kolaču" za 1.0.

Postoje i metodi selekcije koji implementiraju **stohastičko odabiranje ostatka** (eng. *remainder stochastic sampling*). Kod ovog metoda za svaku individuu čiji je fitnes veći od 1.0, cjelobrojni dio fitnesa ukazuje na to koliko će se njenih kopija direktno ubaciti u međugeneraciju. Nakon toga, svi stringovi (uključujući i one kojima je fitnes bio manji od 1.0) ubacuju dodatne kopije u međugeneraciju, sa vjerovatnoćom koja je proporcionalna vrijednosti ostatka fitnesa (tj. necjelobrojnom dijelu fitnesa). Znači string sa fitnesom 2.26 ubacuje dvije kopije direktno u međugeneraciju, a nakon toga ima vjerovatnoću koja je proporcionalna 0.26 (ne jednaka!) da ubaci novu kopiju u međugeneraciju.

Kod **stohastičkog odabiranja ostatka sa zamjenom** (eng. *remainder stochastic sampling with replacement*), pravi se rulet-točak nad ostacima. Opet, da bi se prevazišla teorijska mogućnost da jedan hromozom dopuni ostatak međugeneracije koristi se **stohastičko odabiranje ostatka bez zamjene** (eng. *remainder stochastic sampling without replacement*), gdje se u drugoj fazi (sa ostacima), hromozom izbacuje sa rulet-točka, nakon što mu bude dodijeljena jedna kopija na osnovu njegovog ostatka.

Stohastičko univerzalno odabiranje (eng. *stochastic universal sampling*) je algoritam koji ima samo jednu fazu i koji se najčešće koristi. Umjesto odabiranja samo jednog hromozoma prilikom svakog okretanja rulet-točka, formira se N pokazivača koji su ravnomjerno raspoređeni po obodu točka. N je broj potrebnih odabiranja (broj jedinki u međugeneraciji). Ovako, dovoljno je samo jednom okrenuti točak da bi dobili sve jedinke koje su nam potrebne. Štaviše, ovim će se postići i najbolje, tj. najpoštenije, odabiranje. Interesantno je napomenuti da ovakvo odabiranje zahtijeva broj operacija reda N, dok svi ostali metodi zahtije-

vaju $N \log N$.

1.4.2 Ukrštanje — rekombinacija

Ukrštanje (eng. *crossover*), tj. rekombinacija (eng. *recombination*), je osnovni operator za stvaranje novih hromozoma (nove generacije) iz međugeneracije (slika 1.3). U prirodi, ukrštanjem, potomak dobija dio osobina i jednog i drugog roditelja. Postoji više načina ukrštanja, od kojih su neki obrađeni u ovom odjeljku.

Ukrštanje u jednoj tački

Ukrštanje u jednoj tački (eng. *single-point crossover*) je najprostiji način ukrštanja. Posmatraju se dva stringa dužine l. Slučajno se bira cijeli broj i, koji je iz intervala $[1, l-1]$, a zatim se razmijene genetičke informacije između dva stringa od i-te tačke do kraja stringova. Na primjer, neka su data dva stringa roditelja:

$$P_1 = 10010110$$

$$P_2 = \underline{10111000}$$

Dakle, l je 8. Uzmimo da je $i = 5$. Dobijamo potomke:

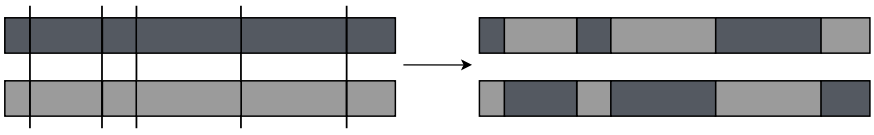
$$O_1 = 10010\underline{000}$$

$$O_2 = \underline{10111}110$$

Ukrštanje u više tačaka

Kod **ukrštanja u više tačaka** (eng. *multi-point crossover*), bira se m tačaka u kojima se vrši ukrštanje. Ove tačke uzimaju vrijednosti iz skupa $\{1, 2, \ldots, l-1\}$, gdje je l dužina hromozoma. Svaka od m tačaka mora uzeti jedinstvenu vrijednost. Obično se tačke biraju slučajno. Na slici 1.5 je dat primjer ukrštanja za $m = 5$, [1].

Slika 1.5: Ukrštanje u više tačaka ($m = 5$)



Povod za uvođenje višestrukog ukrštanja, a i ostalih varijanti operatora ukrštanja je ta da dijelovi hromozoma koji najviše utiču na performanse genetičkog algoritma ne moraju biti smješteni u susjednim podstringovima. Nadalje, „razdvajajuća" (eng. *disruptive*) priroda ukrštanja u više tačaka izgleda podstiče dublje

pretraživanje prostora pretrage, namjesto rane konvergencije, čime se povećava robusnost algoritma. Do skora je najpopularnije ukrštanje bilo ukrštanje u dvije tačke.

Uniformno (ujednačeno) ukrštanje

Prethodne dvije tehnike ukrštanja precizno definišu određeni broj mjesta na kojima će se izvršiti ukrštanje. **Uniformno ukrštanje** (eng. *uniform crossover*) ovo generalizuje, tako da svako razdjelno mjesto postaje potencijalna tačka ukrštanja. Ovo se najbolje objašnjava uvođenjem pojma **maska ukrštanja**. Razmotrićemo naredni primjer:

$$\begin{aligned} P_1 &= 1011000111 \\ P_2 &= \underline{0001111000} \\ mask &= 0011001100 \\ O_1 &= \underline{00}11\underline{11}01\underline{00} \\ O_2 &= 10\underline{01}00\underline{1}011 \end{aligned}$$

Maska (eng. *mask*) je iste dužine kao i roditeljski hromozomi (u primjeru P_1 i P_2) a sastoji se iz slučajno generisanih jedinica i nula. Potomak O_1 se dobija uzimanjem bita od roditelja P_1 na pozicijama gdje je bit maske jedinica, i uzimanjem bita od P_2 ako je odgovarajući bit maske 0. Potomak O_2 se dobija na suprotan način.

Slično ukrštanju u više tačaka smatra se da i uniformno ukrštanje smanjuje pristrasnost koja je u vezi sa dužinom binarne reprezentacije i specifičnim kodiranjem za dati skup parametara. Ovo pomaže prevazilaženju ograničenja ukrštanja u jednoj tački.

Kada se uniformno kodiranje koristi kod alela sa realnim vrijednostima, naziva se **diskretnom rekombinacijom**.

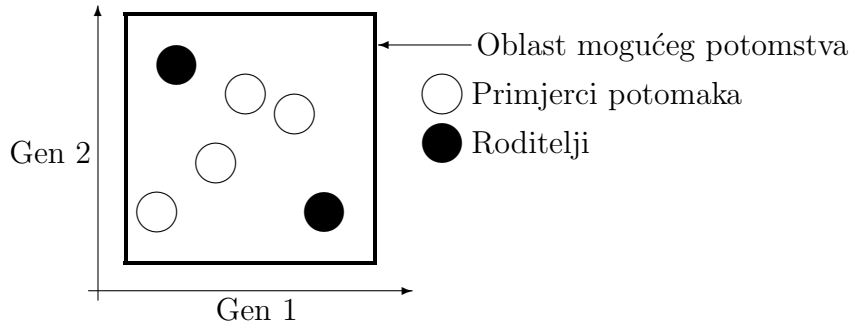
Ostali operatori ukrštanja

Treba pomenuti **operator miješanja** (eng. *shuffle*). Bira se samo jedna tačka ukrštanja, ali prije negoli se bitovi razmijene, roditelji „promiješaju" svoje bite (razmjeste ih). Nakon rekombinacije, biti potomaka se vraćaju na stara mjesta. Ovo takođe uklanja pozicionu naklonost.

Operator redukovanog surogata (eng. *reduced surrogate*) uslovljava da ukrštanje mora, kada je god to moguće, dati nove potomke. Obično se ovo implementira tako što se lokacije ukrštanja ograničavaju na one kod kojih se vrijednosti gena razlikuju.

Postoji i tehnika **ukrštanja sa maticom**. Ona simulira razmnožavanje insekata (pčela) kod kojih jedna jedinka (matica) sa najboljim fitnesom učestvuje u svim ukrštanjima sa ostalim jedinkama (trutovima). U nekim aplikacijama je ovaj tip izbora hromozoma za ukrštanje postigao značajno bolje rezultate nego druge tehnike.

Slika 1.6: Intermedijarna rekombinacija



Intermedijarna rekombinacija

Ovaj vid rekombinacije (eng. *intermediate recombination*) se koristi kod hromozoma sa realnim vrijednostima alela i omogućava dobijanje novih fenotipova oko i između vrijednosti roditeljskih fenotipova. Potomstvo se dobija saglasno pravilu:

$$O_1 = P_1 \times \alpha(P_1 - P_2)$$

gdje je α faktor skaliranja koji se bira uniformno i slučajno iz nekog intervala, obično $[-0.25, 1.25]$, a P_1 i P_2 su roditeljski hromozomi. Svaka promjenljiva iz potomstva je rezultat kombinovanja promjenljivih od roditelja na način koji je dat u formuli, i to tako što se novo α bira za svaki par roditeljskih gena. Geometrijski prikaz je dat na slici 1.6, [1].

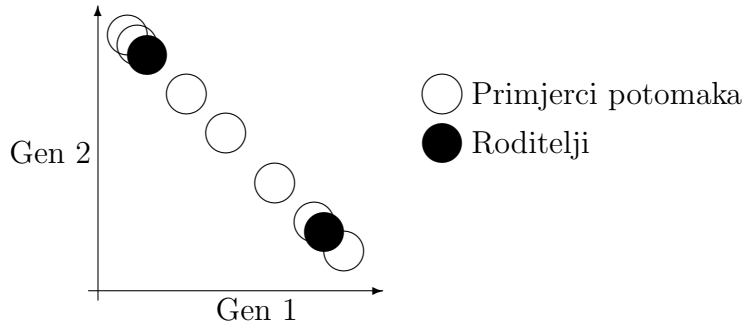
Linearna rekombinacija

Ovaj vid rekombinacije (eng. *linear recombination*) je sličan intermedijarnoj, osim što se koristi samo jedna vrijednost za α u procesu rekombinacije. Slika 1.7 [1] prikazuje kako linijska rekombinacija može generisati ma koju tačku na liniji definisanoj roditeljima unutar granica definisanih „poremećajem" α. Rekombinacija je data za dvije varijable.

1.4.3 Mutacija

U prirodi, mutacija (eng. *mutation*) je proces gdje se jedna alela gena mijenja drugom, što rezultuje novom genetičkom strukturom. Kod genetičkih algoritama, mutacija se primjenjuje slučajno sa niskom vjerovatnoćom, obično u rangu od 0.001 do 0.01, i modifikuje elemente hromozoma. Obično se smatra da je uloga mutacije pozadinska, tj. na mutaciju se gleda kao na nešto što garantuje da vjerovatnoća da se ispita bilo koji string nikada neće biti nula. Takođe, mutacija

Slika 1.7: Linearna rekombinacija



odigrava ulogu sigurnosne mreže koja omogućava da se povrati koristan genetski materijal, koji se mogao izgubiti u selekciji i rekombinaciji.

Efekat mutacije je prikazan tabelom 1.3 [2] za 10-bitnu hromozomsku reprezentaciju realnog broja iz intervala $[0, 10]$ koristeći i standardno i Gray kodiranje, i 3 kao tačku mutiranja binarnog stringa. Ovdje binarna mutacija mijenja (eng.

Tabela 1.3: Binarno i Gray-ovo kodiranje stringova

		Binarno	Gray
Originalni string	0001100010	0.9659	0.6634
Mutirani string	0011100010	2.2146	1.8439

flip) vrijednost bita na poziciji koja odgovara tački mutiranja. Pošto se mutacija generalno primjenjuje uniformno na cjelokupnu populaciju stringova, moguće je da će jedan binarni string biti mutiran na više mjesta.

Kod ne-binarnih reprezentacija, mutacija se postiže ili perturbacijom vrijednosti gena ili slučajnim odabirom novih vrijednosti unutar dozvoljenog intervala. Izgleda da realno-kodirani genetički algoritmi mogu iskoristiti veće frekvencije mutacije, za razliku od binarno-kodiranih genetičkih algoritama, čime se povećava nivo moguće pretrage prostora rješenja, a da se ne utiče negativno na karakteristike konvergencije.

Postoje mnoge varijacije operatora mutacije. Na primjer, naklonjenost mutacije individuama sa manjim fitnesom da bi se proširilo polje pretrage, a istovremeno očuvanje informacija od onih sa većim fitnesom. Ili, parametrizacija mutacije tako da frekvencija mutacije opada sa konvergencijom populacije.

Treba pomenuti i **mutaciju razmjene** (eng. *trade mutation*), gdje se vrijednosti pojedinih gena u hromozomu koriste za usmjeravanje mutacije ka slabijim vrijednostima, i **mutaciju pregrupisavanja** (eng. *reorder mutation*), koja mijenja pozicije bita ili gena da bi povećala raznovrsnost u prostoru promjenljivih.

1.4.4 Zamjena generacije

Kada se, napokon, proizvede međugeneracija odabiranjem, rekombinacijom i mutacijom individua iz tekuće generacije, pristupa se proračunu fitnesa novih individua. Ako se rekombinacijom proizvede manje individua nego što ih već ima u originalnoj (tekućoj) generaciji, onda se razlika u broju individua međugeneracije i tekuće generacije naziva generacijskim jazom (eng. *generation gap*). Za slučaj kada se u svakoj proizvede jedan ili dva primjerka, kažemo da je genetički algoritam u stabilnom stanju (eng. *steady-state*) ili sa priraštajem (eng. *incremental*). Ako se jednom ili većem broju primjeraka sa najvećim fitnesom dozvoli da deterministički propagira (tj. opstane) kroz više uzastopnih generacija, tada se kaže da genetički algoritam koristi elitističku strategiju (eng. *elitist strategy*) ili **elitizam**.

Da bi se održala veličina originalne populacije, nove jedinke moraju da zamijene određeni broj starih primjeraka. Takođe, ako ne želimo da ubacimo u populaciju sve nove jedinke, ili ako proizvedemo prevelik broj novih jedinki (veći od ukupnog dozvoljenog broja jedinki u populaciji), onda moramo definisati strategiju zamjene (eng. *reinsertion*) da bi precizno utvrdili koje će jedinke biti ubačene u populaciju, a koje ne. Međutim, prednost algoritma kod kojeg ne pravimo prevelik broj novih jedinki je u tome što se smanjuje vrijeme generisanja novih jedinki, što je najuočljivije kod genetičkog algoritma u stabilnom stanju, uz smanjenje memorijskih zahtjeva jer je potrebno skladištiti manji broj individua.

Kada se postavi pitanje koji će predstavnici stare generacije biti zamijenjeni, nameće se kao logičan odgovor da to treba da budu oni sa najmanjim fitnesom. Ipak, to ne mora da bude optimalno rješenje, i time u suštini implementiramo elitizam, jer se povećava vjerovatnoća da će članovi sa najvećim fitnesom propagirati kroz veliki broj generacija. U stvari, najbolja tehnika zamjene je da se izbace najstariji primjerci, pa čak i oni koji su u nekim ranijim generacijama imali odličan fitnes. Stoga, da bi genetički materijal neke jedinke propagirao (preko potomstva) kroz niz generacija, onda ona mora imati zaista odličan fitnes.

Postoji mnoštvo strategija u zamjeni prethodne generacije novom, a sve proizilaze iz prethodne priče. Mi ćemo pomenuti tri:

Zamjena kompletne generacije. Izvrši se ukrštanje roditelja i svi roditelji se zamijene novim primjercima, čiji broj (nakon rekombinacije) mora biti veći ili jednak broju starih primjeraka. Ova strategija je dobra kada smo daleko od minimuma.

Zamjena u stabilnom stanju. Svi primjerci (najčešće jedan ili dva — kako je već navedeno) bivaju generisani i umetnuti u staru generaciju.

Ubijanje hromozoma. U svakoj generaciji se ubija po nekoliko hromozoma sa najgorim performansama kao i oni hromozomi koji su u populaciji zadati broj epoha (kaže se da su „umrli od starosti"). Ipak, danas se često u ovoj metodi implementira elitizam.

1.4.5 Prekidanje genetičkog algoritma

Pošto je genetički algoritam stohastički metod pretraživanja, teško je formalno precizirati kriterijume konvergencije. Kako fitnes cjelokupne populacije može ostati nepromijenjen kroz niz generacija, sve dok se ne naiđe na superiornog pojedinca, prekid (eng. *termination*) algoritma na klasičan način (zadovoljenjem uslova) postaje problematičan. Najčešće se u praksi genetički algoritam prekida nakon određenog broja generacija, a zatim se provjerava kvalitet najboljih jedinki. Ako rezultat nije prihvatljiv možemo opet pokrenuti pretragu — iznova.

Poglavlje 2

Teorija iza genetičkih algoritama

Ovo poglavlje daje osnovne informacije vezane za teorijsku osnovu genetičkih algoritama, ali neko opširnije izlaganje prevazilazi okvire jednog diplomskog rada. Kao izvor informacija korišteno je [2]. Taj rad ulazi u detaljniju analizu i preporučuje se čitaocu koga više interesuje oblast.

2.1 Zašto genetički algoritmi rade? Hiperkocke, sheme

Kako to da genetički algoritmi daju nešto korisno kao rezultat, ili preciznije, kako rade? Kakva se to skrivena logika krije iza njih? Odgovor leži u tome da genetički algoritam implicitno odabira[1] **hiperprostor**[2] (eng. *hyperspace*). Mi ćemo uvesti i pojam **dio hiperprostora**, i pod njime ćemo podrazumijevati „podprostor", tj. particiju (eng. *partition*) ili podskup hiperprostora. Takođe ćemo, u istom značenju, koristiti i termin **hiperravan** (eng. *hyperplane*). Ako je (hiper)prostor n-dimenzioni, onda hiperravan može imati najviše $n-1$ dimenzija. Da bi shvatili kako genetički algoritam može da pokrije (pretraži) razne dijelove hiperprostora posmatraćemo jednostavni trodimenzioni prostor.

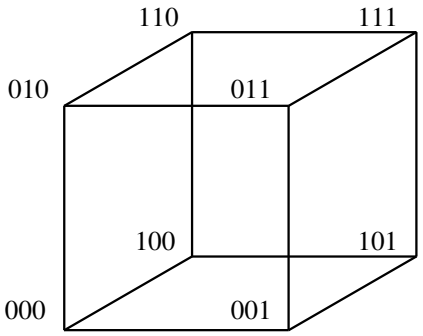
Pretpostavimo da imamo problem koji je kodiran sa samo 3 bita; ovo može biti predstavljeno kao jednostavna kocka sa stringom 000 kao „prvim" ćoškom. Ćoškovi kocke su numerisani stringovima bita, s tim što se svi susjedni stringovi razlikuju za tačno jedan bit.

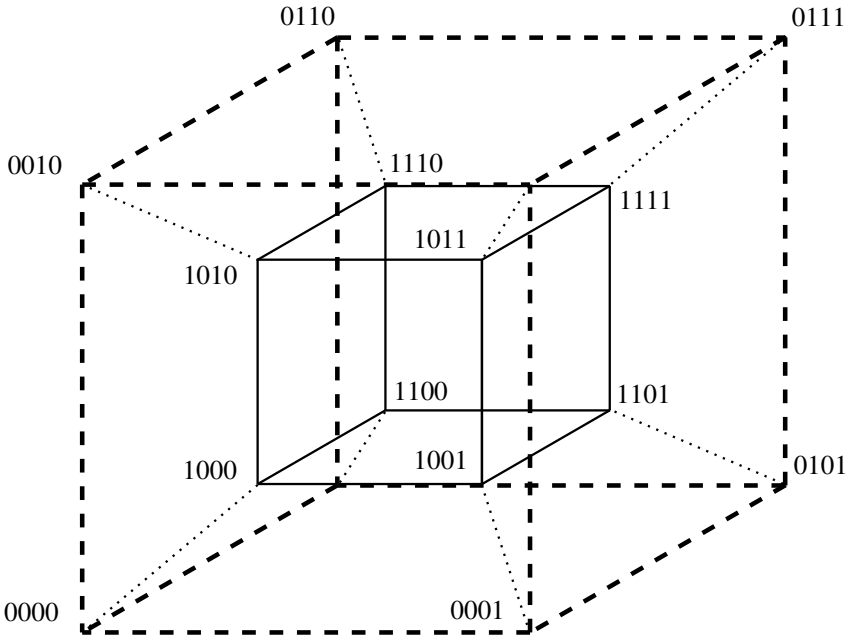
Primjer ovoga je slika 2.1 [2], i to gornja kocka. Prednja stranica kocke sadrži sve tačke (ukupno 4) koje počinju sa 0. Ako * iskoristimo kao znak koji označava bilo 1, bilo 0, onda se ova stranica može predstaviti specijalnim stringom 0**. Stringovi koji u sebi sadrže makar i jednu „zvjezdicu" (eng. *asterisk*) se nazivaju **shemama** (*schema, pl. schemata*) ili **šemama**; svaka shema odgovara jednoj hiperravni u prostoru pretrage. Red hiperravni se odnosi na broj „stvarnih" bita koji se pojavljuju u shemi. Tako, 1** je reda 1 dok bi hiperravan 1**1******0**

[1] „Sempluje" (eng. *sample*)

[2] Hiperprostor je n-dimenzioni prostor

Slika 2.1: 3-dimenziona i 4-dimenziona hiperkocka. Ćoškovi unutrašnje kocke i spoljašnje kocke u donjem 4-D primjeru su numerisani na isti način kao i gornja 3-D kocka, samo što je 1 dodato kao prefiks oznakama unutrašnje kocke, a 0 oznakama spoljašnje. Samo odabrane tačke su obilježene na 4-D kocki





bila reda 3.

Dno slike 2.1 ilustruje 4-dimenzioni prostor predstavljen kockom koja „visi" unutar druge kocke. Obilježimo tačke i unutrašnje i spoljašnje kocke identično kao na gornjoj , čisto 3-D kocki. Sada, dodajmo prefiks 1 svakoj oznaci na unutrašnjoj kocki, a 0 svakoj oznaci na unutrašnjoj. Ovo daje takav raspored tačaka u hiperprostoru da su, slično kao na gornjoj kocki, sve tačke koje se razlikuju za tačno jedan bit susjedne. Unutrašnja kocka sada odgovara hiperravni 1***, a spoljašnja hiperravni 0***. Takođe, lako je uvidjeti da *0** odgovara podskupu tačaka na prednjim stranama obije kocke. Hiperravan drugog reda 10** odgovara prednjoj strani unutrašnje kocke.

String pripada određenoj shemi ako se može dobiti iz nje zamjenom svih simbola * odgovarajućim vrijednostima bita (1 ili 0). Uopšte, svi stringovi koji pripadaju određenoj shemi se nalaze na hiperravni koju predstavlja ta shema. Svaki binarni niz je „hromozom" koji odgovara nekom od ćoškova hiperkocke i koji je član $2^L - 1$ različitih hiperravni, gdje je L dužina binarnog niza. Kako se dolazi do ovog broja? Malo elementarne kombinatorike će nam dobro doći. Svaka od L pozicija tačno određenog stringa (niza) može biti ili * ili stvarni bit tog stringa (0 ili 1, ali ne oboje). Te je to 2^L (varijacije sa ponavljanjem). Ono -1 je uslijed toga što ne ubrajamo hromozom kod koga su sve alele * (***...*)[3].

Takođe je vrlo lako pokazati da je moguće definisati tačno $3^L - 1$ hiperravanskih particija za čitav prostor pretrage. Na svakoj od L pozicija stringa može biti ili * ili 1 ili 0. To je 3^L.

Podatak da je svaki string član $2^L - 1$ hiperravanskih particija ne daje puno informacija o tome da li svaku tačku prostora pretrage ispitujemo zasebno. Odatle je populacijski zasnovana pretraga kritična za genetičke algoritme. Populacija odabranih tačaka daje informacije o brojnim hiperravnima; nadalje, hiperravni nižeg reda bi trebalo da budu odabrane sa brojnim tačkama populacije. Ključni dio inherentnog ili implicitnog paralelizma (eng. *intrinsic, implicit parallelism*) genetičkog algoritma se izvlači iz činjenice da se mnoge hiperravni odabiraju kada ispitujemo populaciju stringova; štaviše, može se pokazati da se mnogo više hiperravni odabira nego što ima stringova u cjelokupnoj populaciji ($3^L \gg 2^L$). Mnogo različitih hiperravni se ispituje na jedan implicitno paralelan način svaki put kada se samo jedan string ispituje; međutim, tek kumulativni efekat ispitivanja čitave populacije daje statističke podatke o bilo kojem, unaprijed definisanom podskupu hiperravni. Implicitni paralelizam podrazumijeva to da se mnoge usporedbe (kao prava takmičenja!) hiperravni simultano obavljaju u paraleli. Teorija nam sugeriše da kroz proces reprodukcije i rekombinacije, sheme „suparničkih" hiperravni povećavaju ili smanjuju svoj udio (tj. prisustvo) u populaciji shodno relativnom fitnesu stringova koji leže u tim hiperravanskim particijama. Iz razloga što genetički algoritmi djeluju nad populacijama stringova, može se pratiti proporcionalna zastupljenost jedne jedine sheme koja predstavlja određenu hiperravan u populaciji i na osnovu toga procijeniti da li će se ta zastupljenost povećati ili smanjiti u populaciji tokom vremena kada se odabir zasnovan na fitnesu upari sa

[3]String koji se sastoji samo od znakova * označava prostor i ne smatra se particijom prostora

ukrštanjem da bi se dobilo potomstvo od postojećih stringova u populaciji.

2.2 Dva prikaza odabiranja hiperravni

Još jedan način da se gleda na hiperravanske particije je dat na slici 2.2, [2]. Posmatra se funkcija jedne promjenljive, a cilj je njena maksimizacija. Hiperravan 0****...** pokriva prvu polovinu prostora, a 1****...** drugu. Pošto su stringovi u 0****...** particiji u prosjeku bolji od onih u 1****...** particiji, mi bi željeli da pretraga bude proporcionalno naklonjena njoj. Na drugom grafiku šrafiran je i dio prostora koji odgovara particiji **1**...**, pa se jasno vidi i presjek ove particije sa 0****...**, koji se može označiti kao 0*1**...** particija. Konačno, na trećem grafiku je prikazana hiperravan 0*10*...**.

Jedan od zaključaka koji se može izvući sa ova slike je da odabiranje hiperravanskih particija nije podložno uticaju lokalnih optimuma (ekstremuma). U isto vrijeme, povećanje učestanosti odabiranja particija koje su iznad prosjeka u odnosu na druge particije ne garantuje konvergenciju ka globalnom optimumu. Globalni optimum može biti jedan relativno izolovani vrh (pik), na primjer. Ipak, dovoljno dobra rješenja bi trebala biti nađena.

Isto tako, korisno je za vježbu pogledati primjer jednog jednostavnog genetičkog algoritma u akciji (opet, izvor je [2]). U tabeli 2.1 prva tri bita svakog stringa su data, dok ostali biti nijesu precizirani. Cilj je posmatrati samo one hiperravni koje su definisane pozicijama prva tri bita da bi se vidjelo šta se stvarno dešava tokom faze odabiranja, kada se stringovi umnožavaju shodno svom fitnesu. Teorija iza genetičkih algoritama sugeriše da bi se nova raspodjela tačaka u svakoj hiperravni trebala odviti shodno srednjem fitnesu stringova populacije koji pripadaju odgovarajućoj hiperravanskoj particiji. Prema tome, iako genetički algoritam nikada ne ispituje direktno nijednu hiperravansku particiju, ipak bi trebao da mijenja raspored kopija stringova kao da to zaista radi.

Primjer iz tabele 2.1 sadrži samo 21 string. Pošto nas tačna evaluacija ovih stringova puno i ne interesuje, date su samo vrijednosti fitnesa. U tabeli se nalaze informacije o fitnesu svakog stringa, i o broju kopija koji će se smjestiti u međugeneraciju. U ovom primjeru, broj kopija proizvedenih tokom selekcije je određen cjelobrojnom vrijednošću fitnesa i zbirom decimalnog ostatka fitnesa i broja „Random", koji je u intervalu od 0.0 do 1.0. Ako je ovaj zbir veći od 1, stringu se dodjeljuje (još) jedna kopija.

Izgleda da genetički algoritam ispituje više hiperravni implicitno u paraleli kada djeluje operator selekcije. Tabela 2.2 prikazuje 27 hiperravni (3^3) koje mogu biti definisane za prva tri bita stringova u populaciji i *eksplicitno* računa fitnes koji je pridružen odgovarajućoj hiperravanskoj particiji. Istinski fitnes hiperravanske particije odgovara prosječnom fitnesu svih stringova koji leže u njoj. Genetički algoritam koristi populaciju kao svojevrstan odbirak za procjenu fitnesa hiperravanske particije. Naravno, jedini put kada je odabiranje slučajno je tokom prve generacije. Nakon ovoga, odabiranje novih stringova bi trebalo biti naklonjeno regijama koje su u ranijim iteracijama sadržavale stringove koji su bili

Slika 2.2: Proizvoljna funkcija i razne particije hiperprostora. Fitnes je skaliran u opsegu od 0 do 1

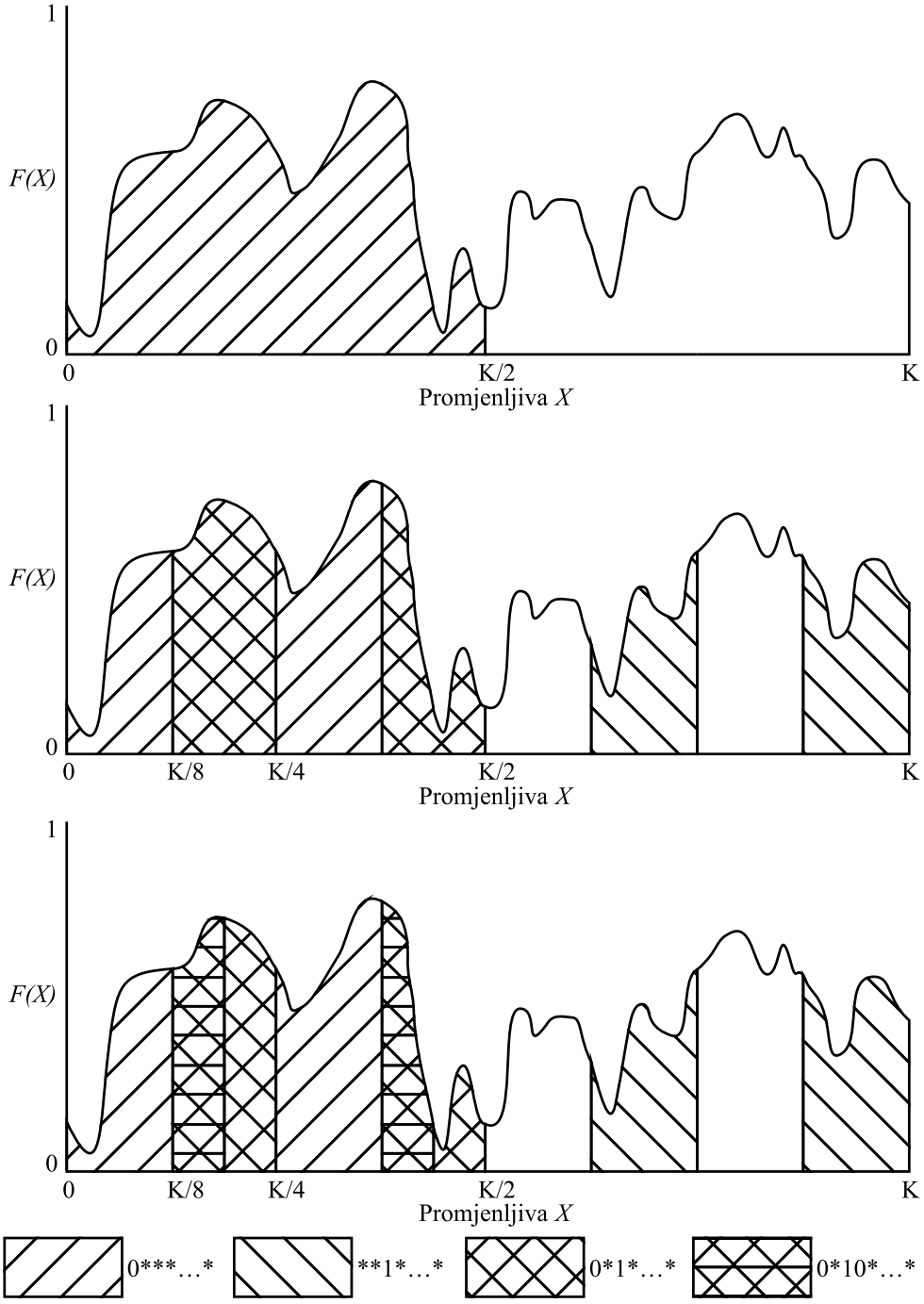


Tabela 2.1: Data je populacija sa dodijeljenim fitnesima. „Random" je slučajan broj koji određuje da li će se dodijeliti kopija stringu za decimalni ostatak njegovog fitnesa

String	Fitnes	Random	Br. kopija
$001b_{1,4}\ldots b_{1,L}$	2.0	-	2
$101b_{2,4}\ldots b_{2,L}$	1.9	0.93	2
$111b_{3,4}\ldots b_{3,L}$	1.8	0.65	2
$010b_{4,4}\ldots b_{4,L}$	1.7	0.02	1
$111b_{5,4}\ldots b_{5,L}$	1.6	0.51	2
$101b_{6,4}\ldots b_{6,L}$	1.5	0.20	1
$011b_{7,4}\ldots b_{7,L}$	1.4	0.93	2
$001b_{8,4}\ldots b_{8,L}$	1.3	0.20	1
$000b_{9,4}\ldots b_{9,L}$	1.2	0.37	1
$100b_{10,4}\ldots b_{10,L}$	1.1	0.79	1
$010b_{11,4}\ldots b_{11,L}$	1.0	-	1
$011b_{12,4}\ldots b_{12,L}$	0.9	0.28	1
$000b_{13,4}\ldots b_{13,L}$	0.8	0.13	0
$110b_{14,4}\ldots b_{14,L}$	0.7	0.70	1
$110b_{15,4}\ldots b_{15,L}$	0.6	0.80	1
$100b_{16,4}\ldots b_{16,L}$	0.5	0.51	1
$011b_{17,4}\ldots b_{17,L}$	0.4	0.76	1
$000b_{18,4}\ldots b_{18,L}$	0.3	0.45	0
$001b_{19,4}\ldots b_{19,L}$	0.2	0.61	0
$100b_{20,4}\ldots b_{20,L}$	0.1	0.07	0
$010b_{21,4}\ldots b_{21,L}$	0.0	-	0

iznad prosjeka.

Ako genetički algoritam radi kao što smo razmotrili, onda bi broj kopija stringova koji pripadaju određenoj hiperravanskoj particiji nakon selekcije trebao biti približan očekivanom broju kopija za tu particiju.

U tabeli 2.2, *očekivani* broj stringova koji odabiraju neku hiperravansku particiju može biti izračunat tako što ćemo pomnožiti broj odbiraka hiperravni u tekućoj populaciji prije selekcije sa prosječnim fitnesom stringova populacije koji pripadaju toj particiji. *Posmatrani* (ostvareni) broj kopija je takođe dat. U većini slučajeva je greška vrlo mala i uzrokovana je malom populacijom.

Sada ćemo formalizovati ideju praćenja potencijalne frekvencije odabiranja za neku hiperravan H. Neka je $M(H,t)$ broj stringova koji odabiraju H u tekućoj generaciji t neke populacije. Neka $(t + intermed)$ označava generaciju t nakon selekcije (ali prije ukrštanja i mutacije), i neka je $f(H,t)$ prosječna evaluacija odabranih stringova u particiji H tekuće populacije. Formalno, promjena u reprezentaciji prouzrokovana fitnesom stringova koji su uzeti iz hiperravni H

Tabela 2.2: Eksplicitno su izračunate prosječna (srednja) vrijednost za odgovarajuće odbirke iz 27 hiperravni definisanih za prve tri pozicije bita. Prikazane su očekivana i ostvarena reprezentacija. „Prije sel." se odnosi na broj stringova u hiperravni prije selekcije

Shema	Srednja vr.	Prije sel.	Očekivani	Posmatrani
101*...*	1.70	2	3.4	3
111*...*	1.70	2	3.4	4
1*1*...*	1.70	4	6.8	7
01...*	1.38	5	6.9	6
**1*...*	1.30	10	13.0	14
11...*	1.22	5	6.1	8
11**...*	1.175	4	4.7	6
001*...*	1.166	3	3.5	3
1***...*	1.089	9	9.8	11
0*1*...*	1.033	6	6.2	7
10**...*	1.020	5	5.1	5
*1**...*	1.010	10	10.1	12
****...*	1.000	21	21.0	21
*0**...*	0.991	11	10.9	9
00**...*	0.967	6	5.8	4
0***...*	0.933	12	11.2	10
011*...*	0.900	3	2.7	4
010*...*	0.900	3	2.7	2
01**...*	0.900	6	5.4	6
0*0*...*	0.833	6	5.0	3
10...*	0.800	5	4.0	4
000*...*	0.767	3	2.3	1
**0*...*	0.727	11	8.0	7
00...*	0.667	6	4.0	3
110*...*	0.650	2	1.3	2
1*0*...*	0.600	5	3.0	4
100*...*	0.566	3	1.70	2

se može izraziti kao:

$$M(H, t + intermed) = M(H, t)\frac{f(H, t)}{\bar{f}}$$

gdje je $\bar{f}$ srednja vrijednost fitnesa za čitavu generaciju. Dakle, ovo je promjena u reprezentaciji za već postojeće stringove, i time još nijesmo dobili suštinski nove stringove, odnosno nove tačke odabiranja u hiperravni. Nove stringove dobijamo tek nakon ove faze, kada primijenimo rekombinaciju i mutaciju, a ti novodobijeni stringovi bi trebalo da budu u korelaciji sa gore pomenutom promjenom u reprezentaciji.

2.3 Operatori ukrštanja — uticaj na sheme

Posmatrana zastupljenost hiperravni u tabeli 2.2 odgovara zastupljenosti u međugeneraciji nakon selekcije, ali prije rekombinacije. Kakav je uticaj rekombinacije na posmatranu raspodjelu stringova? Očigledno, hiperravni prvog reda nijesu pogođene rekombinacijom, zato što kritični bit uvijek nasljeđuje jedan od potomaka. Međutim, hiperravanske particije drugog ili višeg reda mogu biti pogođene ukrštanjem. Nadalje, čak ni sve hiperravni istog reda nijesu, u principu, pogođene ukrštanjem u istoj mjeri. Što se tiče ukrštanja u jednoj tački, ono je pogodno za proučavanje jer je relativno lako kvantifikovati njegove efekte na različite sheme koje predstavljaju hiperravni. Uzećemo, radi jednostavnosti, a ne utičući na opštost, kao primjer string od 12 bita. Neka su date sljedeće dvije sheme:

$$11********** \qquad \text{i} \qquad 1**********1$$

Vjerovatnoća da će kritični biti prve sheme biti **razdvojeni** (eng. *disrupted*) tokom ukrštanja u jednoj tački je tek $\frac{1}{(L-1)}$ pošto postoji $L-1$ tačaka ukrštanja u stringu dužine L, a samo jedna tačka razdvajanja. Vjerovatnoća da će se biti u desnoj shemi razdvojiti je, nasuprot tome, $\frac{L-1}{L-1}$, ili 1.0, zato što svaka od $L-1$ tačaka ukrštanja razdvaja bite iz sheme. Možemo zaključiti sljedeće: kada koristimo ukrštanje u jednoj tački pozicije bita u shemi su važne, jer utiču na vjerovatnoću da će biti ostati zajedno nakon ukrštanja.

2.3.1 Ukrštanje u dvije tačke

Šta će se desiti ako primijenimo ukrštanje u dvije tačke? Ovaj operator, kao što znamo, koristi dvije nasumično odabrane tačke; stringovi razmjenjuju segment koji se nalazi između njih. Može se reći da se operator ukrštanja u dvije tačke odnosi prema stringu (shemi) kao prema prstenu. Primjer za ovo je:

$$\begin{array}{cccccccccccc}
 & & b_4 & b_3 & & & & & * & * & & \\
 & b_5 & & & & b_2 & & & * & & & * \\
b_6 & & & & & & b_1 & & * & & & 1 \\
b_7 & & & & & & b_{12} & & * & & & 1 \\
 & b_8 & & & & b_{11} & & & * & & & * \\
 & & b_9 & b_{10} & & & & & & * & * &
\end{array}$$

gdje su $(b_1, b_2, \ldots, b_{12})$ biti. Dakle, što su biti bliži u prstenu, manja je vjerovatnoća razdvajanja. Maksimalno razdvajanje dvobitne sheme se dešava kada su biti na komplementarnim pozicijama (npr. kada bi b_1 i b_7 bili jedinice). Očigledno da su neke sheme kompaktnije od drugih (za definisano ukrštanje i red sheme), i da će one biti vjerovatno odabirane frekvencijom koja je bliža onoj ostvarenoj bez ukrštanja. Možemo reći da je **kompaktna reprezentacija**, u odnosu na

shemu, ona koja minimizuje vjerovatnoću razdvajanja tokom ukrštanja. Ova definicija je zavisna od operatora ukrštanja, jer su obije sheme iz prvog primjera u odjeljku 2.3 jednako i maksimalno kompaktne u odnosu na ukrštanje u dvije tačke, ali maksimalno različite u odnosu na ukrštanje u jednoj tački.

2.3.2 Povezanost i definišuća dužina

Povezanost (eng. *linkage*) se odnosi na fenomen gdje se skupina bita ponaša kao više „međusobno prilagođenih alela" (eng. *coadapted alleles*) koje se obično nasljeđuju u grupi. Naravno, povezanost se može posmatrati kao generalizacija sintagme „kompaktna reprezentacija u odnosu na shemu". Međutim, najčešće se definiše kao prosta fizička bliskost (bez prstenova i sličnih stvari) bita u stringu, koja se mjeri **definišućom dužinom** (eng. *defining length*).

Definišuća dužina je restojanje između dva bita u shemi koji su najviše udaljeni jedan od drugog. Na primjer, u shemi ***0**1***0* četvrti bit i pretposljednji (jedanaesti) bit određuju definišuću dužinu, i ona je $11 - 4 = 7$. Normalno, govorimo o najudaljenijim „pravim" bitima, ne zvjedicama. Svrha uvođenja definišuće dužine je u tome što nam ona daje precizan podatak o tome koliko ima tačaka u kojima će ukrštanje imati efekta. Mi ćemo definišuću dužinu sheme koja odgovara nekoj hiperravni H obilježavati sa ΔH.

2.4 Teorema sheme

Teorema sheme (eng. *The schema theorem*) je istorijski prva teorema koja je pokušala da objasni fenomen genetičkih algoritama i danas je, možemo slobodno reći, prevaziđena. Neki od problema u vezi sa njom su to što je ona nejednakost, što uvodi neka pregruba zaokruživanja, a što sve zajedno rezultuje time da ona važi samo za jednu generaciju u budućnosti, što nam nikako ne omogućava da pratimo populaciju na duže staze. Ipak ona je sasvim dovoljna da bi se „pohvatali" neki suštinski koncepti.

Na strani 22 smo dobili sljedeći izraz:

$$M(H, t + intermed) = M(H, t)\frac{f(H, t)}{\bar{f}}$$

Da bi našli $M(H, t+1)$, moramo uzeti u obzir i efekat koji prouzrokuje ukrštanje. Iz tog razloga uvodimo pojmove **gubici** i **dobici**, pa dobijamo sljedeću relaciju:

$$M(H, t+1) = (1-p_c)M(H, t)\frac{f(H, t)}{\bar{f}} + p_c\left[M(H, t)\frac{f(H, t)}{\bar{f}}(1 - gubici) + dobici\right]$$

Gubici i dobici se odnose na reprezentaciju hiperravni H, a p_c je vjerovatnoća ukrštanja. Sada se obično uvodi jedna gruba aproksimacija, tako što zanemarujemo dobitke, a gubitke, opet grubo, predstavljamo brojem razdvajanja.

$$M(H, t+1) \geq (1 - p_c)M(H, t)\frac{f(H, t)}{\bar{f}} + p_c\left[M(H, t)\frac{f(H, t)}{\bar{f}}(1 - razdvajanja)\right]$$

Međutim, kada se dva stringa koji odabiraju hiperravan H ukrste, onda se ne dešava razdvajanje, i to moramo uzeti u obzir. Sada dobijamo da je broj razdvajanja dat kao:

$$\frac{\Delta(H)}{L-1}(1 - P(H,t))$$

$P(H,t)$ predstavlja proporcionalnu zastupljenost hiperravni H u populaciji i dobija se kao količnik $M(H,t)$ i ukupne veličine populacije (tj. broja svih stringova u populaciji).

Sada ćemo malo preurediti našu nejednakost, tako što ćemo obije strane podijeliti sa ukupnom veličinom populacije i izvršiti neka pregrupisavanja:

$$P(H,t+1) \geq P(H,t)\frac{f(H,t)}{\bar{f}}\left[1 - p_c\frac{\Delta(H)}{L-1}(1 - P(H,t))\right]$$

U ovaj izraz možemo unijeti i činjenicu da se često oba roditelja biraju na osnovu fitnesa. Ovo se implementira tako što ćemo naznačiti da se alternativni roditelj bira iz međugeneracije, nakon selekcije. Slijedi:

$$P(H,t+1) \geq P(H,t)\frac{f(H,t)}{\bar{f}}\left[1 - p_c\frac{\Delta(H)}{L-1}(1 - P(H,t)\frac{f(H,t)}{\bar{f}})\right]$$

Treba još ubaciti i mutaciju u dobijenu formulu. Ako je p_m vjerovatnoća mutiranja i $o(H)$ red (eng. *order*) H dobijamo:

$$P(H,t+1) \geq P(H,t)\frac{f(H,t)}{\bar{f}}\left[1 - p_c\frac{\Delta(H)}{L-1}(1 - P(H,t)\frac{f(H,t)}{\bar{f}})\right](1 - p_m)^{o(H)}$$

Ovdje smo pretpostavili da mutacija uvijek mijenja bit.

Napominjemo da je ovo izvođenje dato samo radi kompletnosti rada, da su neke stvari preskočene, pojednostavljene i da se detaljniji (i savremeniji) uvid u teoriju može pronaći u [2].

Poglavlje 3

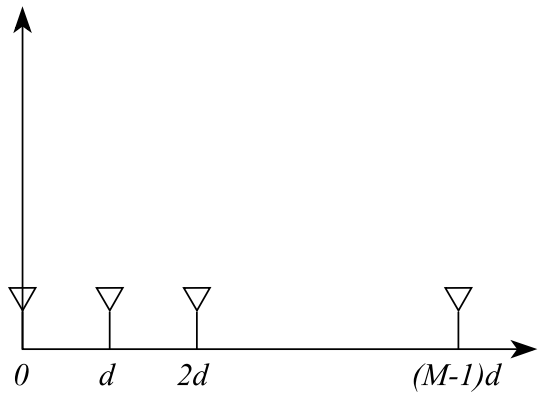
Jedan praktičan primjer

U ovom poglavlju je data simulacija genetičkog algoritma koji služi za računanje smjera dolaska (eng. *Direction Of Arrival — DOA*) signala na uniformni linearni niz senzora (eng. *Uniform Linear Array — ULA*). Kao polazna referenca korišteno je [3], zatim [4], [5], [6].

3.1 Uvod

Primjer jednog ULA je dat na slici 3.1. d je razdaljina između susjednih senzora,

Slika 3.1: Šema jednog ULA



M je broj senzora. Neka na ULA stiže L različitih signala. Opšti izraz za izlaz iz ULA je tada dat kao:

$$\boldsymbol{x}(t) = \boldsymbol{A}(t)\boldsymbol{s}(t) + \boldsymbol{n}(t), \qquad t = 0, 1, \ldots, N-1$$

gdje je:

$\boldsymbol{A}(t)$	$M \times L$ matrica usmjeravanja (eng. *direction matrix*);
$\boldsymbol{s}(t)$	$L \times 1$ vektor svih signala koji stižu na ULA;
$\boldsymbol{n}(t)$	$M \times 1$ vektor bijelog, izotropnog gausovskog šuma koji je prisutan u ULA;
N	broj odbiraka signala.

No, mi ćemo pojednostaviti stvari. Posmatraćemo samo jedan linearni FM signal, tj. svešćemo vektor $\boldsymbol{s}(t)$ na samo jedan signal $s(t)$:

$$s(t) = Se^{(jat^2/2+jbt+jc)}$$

gdje je S amplituda. Sada je signal koji se dobija od m-tog senzora jednak:

$$x_m(t) = \widetilde{S}e^{ja(t-m\tau)^2+jb(t-m\tau)+jc} + n_m(t)$$

Naravno $m = 0, 1, \ldots, M-1$. Takođe, $\tau = d \sin\theta/\lambda$, gdje je θ DOA. Za šum važi $E\{n_m(t_1)n_k^*(t_2)\} = \sigma^2\delta(l-k)\delta(t_1-t_2)$, gdje je σ varijansa šuma. U parametru b je sadržana informacija o frekvenciji nosioca, $b = 2\pi(f_c - B/2)$. Talasna dužina signala je $\lambda = c/f_c$, c je brzina prostiranja, a f_c frekvencija nosioca. Cilj je procjena parametara $(a, b, c, \widetilde{S}, \theta)$ na osnovu signala $x_m(t)$, $m = 0, 1, \ldots, M-1$. Mi dalje uproštavamo problem tako što ćemo estimirati samo parametre (a, b, θ), jer je estimacija ostala dva trivijalna.

Za estimaciju je korišten **chirp beamformer**, [4]. On se svodi na maksimizaciju sume:

$$f(\alpha, \beta, \eta) = \frac{1}{N}\left|\sum_{t=0}^{N-1}\sum_{m=0}^{M-1} x_m(t)e^{-j\alpha t^2/2-j\beta t+j\alpha tm\eta+j\beta m\eta}\right|^2$$

po (α, β, η). Procijenjeni parametri su tada:

$$(\hat{a}, \hat{b}, \hat{\tau}) = \arg\max_{(\alpha,\beta,\eta)} f(\alpha, \beta, \eta)$$

Rješavanje gornje jednačine zahtijeva robusnu trodimenzionalnu pretragu i mi smo pokušali da implementiramo jednu takvu pretragu korišćenjem genetičkog algoritma.

3.2 Implementacija u MATLAB-u

Za implementaciju genetičkog algoritma korišten je *Genetic Algorithm TOOLBOX* za MATLAB. Iako vremešan[1], *TOOLBOX* je odlično poslužio svrsi. Uz male izmjene proradio je i sa MATLAB-om 7. Treba napomenuti da je ovaj *toolbox* besplatan i da dolazi sa odličnom dokumentacijom, [1]. MATLAB skripte koje su napisane za potrebe ovog rada su date u dodatku A.

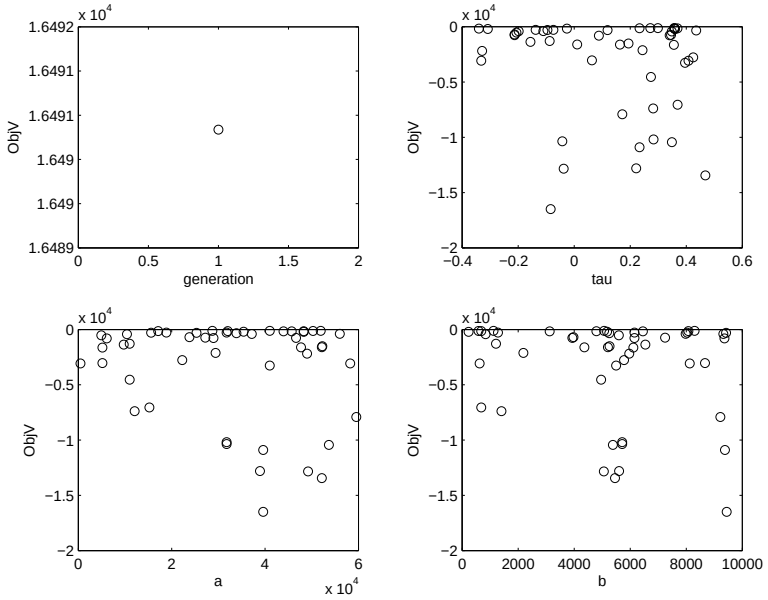
[1] Pisan za MATLAB v4.2 1994-te godine

Odabrano je da ULA ima $M = 8$ senzora. Frekvencija nosioca je $f_c = 1000$Hz. Brzina prostiranja je $c = 1500$m/s. Rastojanje između senzora je $d = \lambda/2$. Maksimalna frekvencija signala je $f_{max} = 2000$Hz. Uzeto je $N = 128$ odbiraka sa periodom odabiranja $\Delta t = \frac{1}{2f_{max}}$. Uzeta je širina opsega signala $B = 200$Hz, zatim $a = 2\pi\frac{B}{N\Delta t}$ i $b = 2\pi(f_c - B/2)$. DOA je odabran da bude $\theta = 25°$.

Što se samog genetičkog algoritma tiče, podijeljen je u dvije faze[2]. U prvoj fazi su granice prostora pretrage široke, da bi se u drugoj fazi suzile. Druga faza je, tako, svojevrsno „fino štelovanje".

U prvoj fazi primijenjen je prosti genetički algoritam[3] sa 300 generacija i 50 individua. Uzet je generacijski jaz (vidjeti 1.4.4) od 200%, tj. u svakoj generaciji se stvara 2 puta više potomaka nego što ima roditelja. Takođe, uzete su nešto uže granice pretrage za τ, i to $\tau \in [0, 0.5]$[4]. U prvim generacijama individue su razbacane svuda po prostoru pretrage. Ovo se vidi na slici 3.2. `ObjV` predstavlja estimaciju naše sume, odnosno, to je najbolji fitnes u generaciji. `a`, `b` i `tau` su estimacije parametara, tj. oni predstavljaju kodirane varijable (gene).

Slika 3.2: Prva generacija



Nakon određenog broja generacija počinje konvergencija ka tačnim vrijednostima. Najbrže konvergira β, potom α, dok τ najsporije konvergira. To se vidi na slici 3.3 kao grupisanje kružića oko određenih vrijednosti.

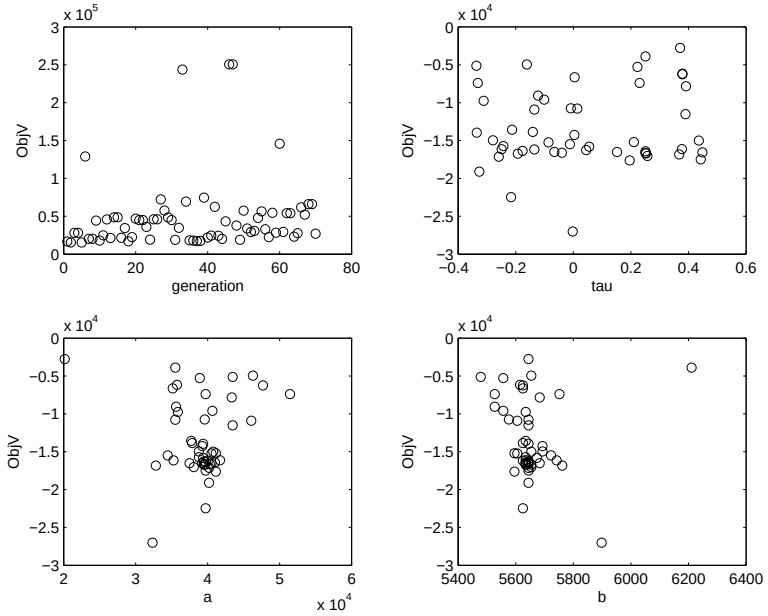
Rezultate dobijene prvim genetičkim algoritmom koristimo kao ulazne podatke druge faze, i to tako što sužavamo prostor pretrage. Broj generacija u drugoj fazi je 150, a generacijski jaz je povećan na 300%.

[2]Tačnije, odrađena su dva genetička algoritma, jedan za drugim

[3]Deriviran iz fajla `SGA.M` koji dolazi sa *TOOLBOX*-om

[4]Ovo se svodi na ugao od 0° do 90°

Slika 3.3: Nakon 70-tak generacija



Algoritam je testiran za četiri SNR-a (odnos signal-šum): $[-5, -2, 2, 5]$. Za svaki SNR je urađeno po 10 simulacija i na kraju je dobijena prosječna vrijednost srednje kvadratne greške (MSE), što je dato u tabeli 3.1.

Tabela 3.1: Vrijednosti MSE dobijene simulacijom

SNR	$a(\cdot 10^7)$	b	τ
-5	1.7876	2.4290	0.0048
-2	0.9044	0.2118	0.0029
2	0.7627	2.7196	0.0003
5	1.1484	1.5633	0.0008

Poglavlje 4

Zaključak

4.1 Oblast primjene

Očigledno je da je oblast primjene genetičkih algoritama vrlo široka. Zbog svojih svojstava, genetički algoritmi se mogu primjenjivati u problemima gdje postoji potreba za robusnom, višedimenzionom pretragom prostora rješenja. To su, na primjer, problemi vezani za vještačku inteligenciju (AI), i to u prvom redu za robote i sl., zatim ekonomski problemi, digitalna obrada signala, koriste se u sprezi sa neuronskim mrežama, u raznim tehnološkim procesima itd. Nadalje, oni se mogu koristiti u sprezi sa klasičnim optimizacionim algoritmima, tako što se prvo primijeni genetički algoritam, a nakon njega npr. LMS algoritam. Treba napomenuti da je danas popularna i još jedna oblast tzv. **Evolutivnog računarstva**, a to je **Genetičko programiranje**, gdje se principi slični onim izloženim u ovom radu koriste za rješavanja nekih problema u programiranju. Nadalje, genetički algoritmi su pogodni za paralelnu, decentralizovanu obradu podataka, što omogućava implementaciju na višeprocesorskim sistemima.

4.2 Rezultati dobijeni MATLAB-om

Primjer dat u poglavlju 3 pokazuje jedan genetički algoritam na djelu. Iako bi se algoritam mogao optimizovati, korišćenjem, na primjer, multipopulacijskog genetičkog algoritma umjesto klasičnog, cilj je postignut — pokazano je kako se genetički algoritam vrlo dobro „bori" sa jednim problemom, koji, kada se rješava na klasičan način, nije jednostavan. Inače, prilikom simulacije su mijenjani određeni parametri, ali to, radi nedostatka prostora, nije uvršteno u ovaj rad.

Bibliografija

[1] A. Chipperfield, P. Fleming, H. Pohlheim, and C. Fonseca, „Genetic Algorithm TOOLBOX For Use with MATLAB". [Online]. Available: citeseer.ist.psu.edu/502345.html

[2] D. Whitley, „A genetic algorithm tutorial," *Statistics and Computing*, vol. 4, pp. 65–85, 1994. [Online]. Available: citeseer.ist.psu.edu/177719.html

[3] I. Djurović, A. Ohsumi, H. Ijima, „Estimation of the parameters and angle-of-arrival of wideband signals on sensor arrays based on the phase differentiation algorithm," in *IEEE OCEANS 2004*, vol. 2, Kobe, Japan, 2004, pp. 631–634. [Online]. Available: www.etfprog.cg.ac.yu/D43.pdf

[4] A. B. Gershman, M. Pesavento, M. G. Amin, „Estimating parameters of multiple wideband polynomial-phase sources in sensor arrays," *IEEE Trans. SP*, vol. 49, no. 12, pp. 2924–2934, Dec. 2001.

[5] I. Djurović, Lj. Stanković, „Non-parametric IF and DOA estimation," in *ISSPA*, vol. 1, Paris, France, 2003, pp. 149–152. [Online]. Available: www.etfprog.cg.ac.yu/D32.pdf

[6] D. Yang and S. J. Flockton, „An evolutionary algorithm for parametric array signal processing," in *Evolutionary Computing, AISB Workshop*, 1995, pp. 191–199. [Online]. Available: citeseer.ist.psu.edu/yang95evolutionary.html

Dodatak A

MATLAB skripte

Za potrebe ovog rada su napisane tri matlab skripte. Fajl `main.m` je glavni, u njemu se podešavaju parametri. Druga dva fajla su funkcijski m-fajlovi. `doaGA.m` implementira genetički algoritam, dok `objfundoa.m` služi za proračun fitnesa individua.

Listing A.1: Glavna skripta (main.m)

```
clear all
close all
%Adding directory '/genetic' to PATH
path(path, './genetic')
tic
%Initialization of signal parameters
%Amplitude
A=1;
%Bandwidth
B = 200;
%Carrier frequency
fc = 1000;
%Max frequency
fmax = 2000;
%The propagation speed
c = 1500;
%Wavelength
lambda = c/fc;
%Number of snapshots (samples)
N = 128;
%sampling rate
dt = 1/(2*fmax);
%Time axis
t = 0 :dt: (N-1)*dt;
%Signal parameters. The parameters are selected to satisfy
```

```
%the bandwidth and the fact that carrier frequency is in
%the middle of the frequency domain
a = 2*pi*B/(N*dt)
b = 2*pi*(fc-B/2)

%Initialization of the sensor array parameters

%Number of sensors
M = 8;
%The distance between sensors in the array
d = lambda/2;
%Destination of arrival
theta = 25*(pi/180);
%theta=linspace(20*(pi/180),22*(pi/180),N);
tau = d*sin(theta)/lambda

%Original signal (not used for anything)
y = A*exp(j*a*t.^2/2+j*b*t);

first = [];
second = [];
MSE=[];
TRIAL = 10;
for SNR = [-5,-2,2,5]
    %variance
    sigm = sqrt(10^(-SNR/10));
    mserr = zeros(1,3);
for trial = 1:TRIAL
        %The signals that arrive at sensors of the ULA
xULA = [];
for m = 0:M - 1
            temp = A*exp(j*a*(t - m*tau).^2/2+j*b*...
            (t - m*tau))+ sigm*(rand(1,N)+j*rand(1,N))/sqrt(2);
            xULA = [xULA;temp];
end
        exact_sum = -objfundoa([a, b, tau], xULA, t)
NIND = 50;          % Number of individuals
MAXGEN = 100;       % maximum Number of generations
GGAP = 2;           % Generation gap
NVAR = 3;           % Number of variables
MUTRATE = 0.1;      % Percent of mutations
% PRECI = 9;        % Precision of binary representation
PRECI = [9,10,11];  % Precision of binary representation
a_range = [0;60000];
```

```
b_range = [0;10000];
tau_range = [0;0.5];
%The first stage GA
result = doaGA(NIND,MAXGEN,GGAP,NVAR,MUTRATE,PRECI...
            ,a_range,b_range,tau_range,xULA,t);
first = [first,result];
%First results
a_range = result(1:2,1)
b_range = result(1:2,2)
tau_range = result(1:2,3)
a_mean = result(3,1)
b_mean = result(3,2)
tau_mean = result(3,3)
%Calling GA function
NIND=50;
MAXGEN = 150;
%PRECI = [10,10,10];
GGAP = 3;
%The second stage GA
result = doaGA(NIND,MAXGEN,GGAP,NVAR,MUTRATE,PRECI...
            ,a_range,b_range,tau_range,xULA,t);
second = [second,result];
%Second results
a_range = result(1:2,1)
b_range = result(1:2,2)
tau_range = result(1:2,3)
a_mean = result(3,1)
b_mean = result(3,2)
tau_mean = result(3,3)
        mserr(1) = mserr(1) + (a_mean-a).^2/TRIAL;
        mserr(2) = mserr(2) + (b_mean-b).^2/TRIAL;
        mserr(3) = mserr(3) + (tau_mean-tau).^2/TRIAL;
        [SNR,trial]
        toc
end
    MSE = [MSE;mserr];
    toc
end
toc
%save main
```

Listing A.2: doaGA.m

```
function result = doaGA(NIND,MAXGEN,GGAP,NVAR,MUTRATE...
```

```
    ,PRECI,a_range,b_range,tau_range,xULA,t)
%Derived from sga.m from the GA toolbox
% Build field descriptor
FieldD = [rep([PRECI],[1, 1]);...
         [a_range,b_range,tau_range];...
         rep([1;0;1;0],[1,NVAR])];
%           1 1 1;
%           0 0 0;
%           1 1 1;
%           0 0 0];

% Initialise population
Chrom = crtbp(NIND, sum(PRECI));

% Reset counters
%Best = NaN*ones(MAXGEN,1); % best in current population
Best = [];
a_est_v = [];
b_est_v = [];
tau_est_v = [];
gen = 0; % generational counter
%figure;
% Evaluate initial population
ObjV = objfundoa(bs2rv(Chrom,FieldD), xULA, t);
% Generational loop
while gen < MAXGEN
    % Assign fitness-value to entire population
    FitnV = ranking(ObjV);
    % Select individuals for breeding
    SelCh = select('sus', Chrom, FitnV, GGAP);
    % Recombine selected individuals (crossover)
    SelCh = recombin('xovsp',SelCh,0.7);
    % Perform mutation on offspring
    SelCh = mut(SelCh, MUTRATE);
    % Evaluate offspring, call objective function
    phen = bs2rv(SelCh,FieldD);
    ObjVSel = objfundoa(phen, xULA, t);
    % Reinsert offspring into current population
    [Chrom ObjV]=reins(Chrom,SelCh,1,1,ObjV,ObjVSel);
    phen = bs2rv(Chrom,FieldD);
    computed_sum = max(-ObjV);
    ind = find(ObjV == min(ObjV));
    ind=ind(1,1);
    a_est = phen(ind,1);
```

```
    a_est_v = [a_est_v,a_est];
    b_est = phen(ind,2);
    b_est_v = [b_est_v,b_est];
    tau_est = phen(ind,3);
    tau_est_v = [tau_est_v,tau_est];
    %size(phen)
    %Ploting the dots
    Best(gen+1) = max(-ObjV);
%     subplot(2,2,1),plot(Best,'ko');
%     xlabel('generation'); ylabel('ObjV');
%     subplot(2,2,2),plot(phen(:,3),ObjV,'ko');
%     xlabel('tau'); ylabel('ObjV');
%     subplot(2,2,3),plot(phen(:,1),ObjV,'ko')
%     ;xlabel('a'); ylabel('ObjV');
%     subplot(2,2,4),plot(phen(:,2),ObjV,'ko');
%     xlabel('b'); ylabel('ObjV');
%     drawnow;
%     Increment generation counter
    gen = gen+1;
end
% End of GA
% figure;
% subplot(2,2,1),plot(sort(Best),'ko');
% xlabel('generation'); ylabel('ObjV');
% subplot(2,2,2),plot(phen(:,3),ObjV,'ko');
% xlabel('tau'); ylabel('ObjV');
% subplot(2,2,3),plot(phen(:,1),ObjV,'ko');
% xlabel('a'); ylabel('ObjV');
% subplot(2,2,4),plot(phen(:,2),ObjV,'ko');
% xlabel('b'); ylabel('ObjV');
Bsort = sort(Best);
ind_v=[];
for ind = MAXGEN-5:MAXGEN
    ind_v = [ind_v,find(Best == Bsort(ind))];
end
a_best = a_est_v(ind_v)
b_best = b_est_v(ind_v)
tau_best = tau_est_v(ind_v)
a_min = min(a_best);
a_max = max(a_best);
a_mean = mean(a_best);
a_dist = (a_max-a_min)/2;
a_range = [a_mean-a_dist;a_mean+a_dist];
b_min = min(b_best);
```

```
b_max = max(b_best);
b_mean = mean(b_best);
b_dist = (b_max-b_min)/2;
b_range = [b_mean-b_dist;b_mean+b_dist];
tau_min = min(tau_best);
tau_max = max(tau_best);
tau_mean = mean(tau_best);
tau_dist = (tau_max-tau_min);
tau_range = [max(tau_range(1,1),tau_mean-tau_dist);...
    min(tau_range(2,1),tau_mean+tau_dist)];
result = [a_range, b_range, tau_range;[a_mean,b_mean,tau_mean]];
```

Listing A.3: objfundoa.m

```
function ObjV = objfundoa(phen, xULA, t)
% These are the estimations!
% But for simplicity are shortened.
a = phen(1:size(phen,1),1);
b = phen(1:size(phen,1),2);
tau = phen(1:size(phen,1),3);

%Generating the objective values
ObjV = [];
for i = 1:size(phen,1)
    %a(i), b(i), tau(i)
    mult2 = [];
    for m = 0:size(xULA,1) - 1
        temp = exp(-j*a(i)*t.^2/2 - j*b(i)*t...
            + j*a(i)*t*m*tau(i) + j*b(i)*m*tau(i));
        mult2 = [mult2;temp];
    end
    %size(mult2)
    %size(xULA)
    inner_sum = xULA.*mult2;
    %size(inner_sum)
    inner_sum = sum(inner_sum);
    %size(inner_sum)
    ObjV = [ObjV; -abs(sum(inner_sum)).^2];
end
%size(ObjV)
```