Genetski algoritmi i primjene

Droždjek, Tomislav

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Tomislav Droždjek

GENETSKI ALGORITMI I PRIMJENE

Diplomski rad

Voditelj rada:
doc. dr. sc. Nela Bosner

Zagreb, Veljača, 2015

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Uvod

Ovaj rad može se podijeliti na 2 dijela. U prvom dijelu opisat ćemo teorijsku stranu evolucijskih algoritama, s naglaskom na genetskim algoritmima. Posebno ćemo se fokusirati na ključne elemente genetskih algoritama i razne implementacije tih elemenata, nakon čega ćemo detaljno opisati za kakav je tip problema koja od implementacija pogodna. U drugom dijelu na primjeru biomorfa demonstrirat ćemo rad genetskih algoritama, varirati njihove parametre te usporediti izvedbu algoritama u ovisnosti o složenosti problema na koji smo ih primijenili.

0.1 Uvod, ideja i povijest evolucijskih algoritama

U ovom poglavlju ukratko ćemo opisati osnovne ideje i pojmove koji su nam potrebni za razmatranje genetskih algoritama. Također ćemo, u kratkim crtama, opisati razvoj ideje genetskih algoritama kroz povijest te biološke procese koji su inspirirali njihov razvoj.

Evolucijski algoritmi, kao što i samo ime govori, posebna su vrsta algoritama inspirirana procesom evolucije. Glavna ideja evolucijskih algoritama je, koristeći metodu pokušaja i pogrešaka, simulirati proces evolucije te ga primijeniti na rješavanje problema.

0.1.1 Osnovni pojmovi

Promotrimo sada podrobnije kako je pojam evolucije povezan s evolucijskim algoritmima. U teoriji evolucije, neku okolinu nastanjuje populacija jedinki kojima je "cilj" preživjeti i razmnožavati se. Podobnost (eng. fitness) tih jedinki govori nam koliko je pojedina jedinka uspješna u ispunjavanju tih ciljeva, dakle reprezentira šansu jedinke da preživi dovoljno dugo kako bi se razmnožavala. U kontekstu rješavanja problema jedinke izjednačavamo s kandidatima za rješenje. Kvalitetu tih potencijalnih rješenja, dakle koliko dobro ona aproksimiraju rješenje problema, možemo iskoristiti kako bi odlučili s kolikom će vjerojatnošću određeni kandidat za rješenje sudjelovati u konstrukciji sljedećih kandidata (intuitivno, što kandidat za rješenje bolje aproksimira rješenje ta bi vjerojatnost trebala biti veća). Sljedeća

tablica prikazuje vezu pojmova iz teorije evolucije i evolucijskih algoritama.

Evolucija	↔	Algoritam
Okolina	↔	Problem
Jedinka	↔	Kandidat za rješenje
Podobnost	↔	Kvaliteta rješenja

0.1.2 Kratak povijesni pregled

Ideja primjene Darwinovih principa evolucije na računalno rješavanje problema datira još iz 1940-ih godina, dakle čak i prije razvoja modernih računala. Već 1948. godine Turing je predložio "genetsko ili evolucijsko pretraživanje", a 1962. Bremermann je provodio eksperimente na "optimizaciji uz evoluciju i rekombinaciju". 60-ih godina razvijene su 3 različite implementacije osnovne ideje evolucijskih algoritama. U SAD-u, Fogel, Owens i Walsh razvili su evolucijsko programiranje (eng. evolutionary programming)[7], dok je Holland svoju metodu nazvao genetskim algoritmom[10]. Istovremeno, u Njemačkoj, Rechenberg i Schwefel razvili su evolucijske strategije (eng. evolution strategies)[13][15]. Neko vrijeme ta su se područja razvijala odvojeno, no od ranih 90-ih godina ona se smatraju različitim predstavnicima jedne grane. U isto vrijeme pojavila se i četvrta podvrsta, genetičko programiranje[11]. Danas se svi algoritmi iz tog područja nazivaju zajedničkim terminom, evolucijski algoritmi. Stari nazivi ostali su kao nazivi podvrsta algoritama.

0.1.3 Inspiracija u biologiji

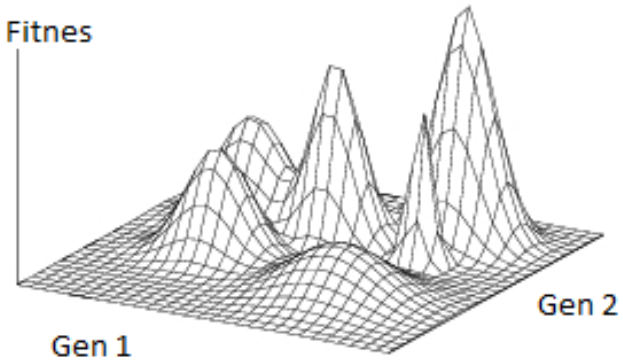
Darwinova teorija evolucije daje nam objašnjenje bioraznolikosti i mehanizama kojima se postiže bioraznolikost. U makroskopskom pogledu na evoluciju, glavnu ulogu igra proces prirodne selekcije. U okolini u kojoj može živjeti samo određen broj jedinki, očito je da je potreban neki oblik selekcije, ako se želi izbjeći eksponencijalan rast populacije. Prirodna selekcija favorizira one jedinke koje su najbolje prilagođene uvjetima okoline, tj. koje najbolje mogu iskoristiti dostupne resurse.

Ovako opisana prirodna selekcija je jedna od dvije osnovne ideje teorije evolucije. Druga osnovna ideja rezultat je fenotipskih varijacija unutar populacije. Fenotip jedinke su karakteristike jedinke (fizičke ili biheviorističke) koje imaju direktan utjecaj na interakciju jednike s okolinom, dakle koje utječu na njenu podobnost, a preko toga i na vjerojatnost preživljavanja. Svaka jedinka jedinstvena je kombinacija fenotipskih karakteristika. Ako okolina te fenotipske karakteristike ocijeni povoljno, onda se one zadržavaju u populaciji kroz potomstvo te jedinke, dok se negativno ocijenjene karakteristike gube jer negativno ocijenjene jedinke češće umiru bez potomstva. Darwin je uočio da se male nasumične varijacije, mutacije, u fenotipu događaju tijekom reprodukcije iz generacije u generaciju.

Kao rezultat tih varijacija, pojavljuju se i ocijenjuju nove kombinacije svojstava. Najbolje među njima prežive i razmnožavaju se, time omogućujući evoluciju.

Ukratko, populacija se sastoji od određenog broja jedinki. Vjerojatnost da se te jedinke razmnožavaju direktno ovisi o tome koliko su one dobro prilagođene okolini. Razmnožavanjem uspješnijih jedinki, uz povremene mutacije, pojavljuju se nove jedinke. Time se, uz dovoljno vremena, mijenja cijela populacija, dakle ona evoluira.

Grafički se taj proces može prikazati kao na grafu 1. Z-os grafa prikazuje podobnost (fitnes), dakle veći z pridružujemo boljoj podobnosti i obrnuto. X i Y os pridružujemo nekim karakteristikama jedinke. Očito, na X-Y ravnini sada su sadržane sve moguće kombinacije karakteristika, dok se na Z-osi može očitati podobnost jedinke s tim karakteristikama. Neku populaciju sad možemo zamisliti kao skup točaka u prostoru, gdje svaka točka predstavlja jednu jedinku. Evoluciju tada možemo zamisliti kao proces postupnog pomaka populacije na veću visinu. Valja spomenuti da je, zbog konačnog broja jedinki te zbog nasumičnosti u cijelom procesu, moguć gubitak dobro prilagođenih jedinki iz populacije. To nam, za razliku od procesa optimizacije, omogućuje i "pomicanje nizbrdo" te nam ništa ne jamči da će se populacija vratiti istim putem. Iz toga slijedi da je moguće pobjeći iz lokalnih optimuma te postići globalni optimum.



Slika 1: Primjer ovisnosti vrijednosti podobnosti o kombinaciji 2 gena.

Poglavlje 1

Evolucijski algoritmi

U ovom poglavlju pozabavit ćemo se glavnim pojmovima koji su nužni kod proučavanja bilo kakvih evolucijskih algoritama. Svi pojmovi opisani u ovom poglavlju zajednički su za sve 4 varijante evolucijskih algoritma [1], iako njihova važnost varira. Opisat ćemo glavne komponente evolucijskih algoritama, objasniti njihovu ulogu te pridruženu terminologiju. Također ćemo usporediti evolucijske algoritme s ostalim algoritmima za optimizaciju.

1.1 Što je evolucijski algoritam?

Kao što smo već spomenuli u prethodnom poglavlju, ideja na kojoj se baziraju evolucijski algoritmi je da će se opća podobnost populacije podići kao rezultat prirodne selekcije. Za danu funkciju podobnosti koju trebamo maksimizirati, nasumično kreiramo skup kandidata za rješenje te na njih primijenimo funkciju. Na temelju rezultata, možemo odabrati bolje kandidate kao roditelje sljedeće generacije i na njih primijeniti rekombinaciju i/ili mutaciju. Operatore mutacije i rekombinacije detaljnije ćemo objasniti kasnije. Primjenom tih operatora dobijemo novi skup kandidata za rješenje za koje opet izračunamo njihovu podobnost. Nakon toga, novi zajedno sa starim kandidatima ulaze u izbor za kandidate koji će se prenijeti u sljedeću generaciju. Taj izbor najčešće se zasniva na temelju podobnosti (ulogu može igrati i starost kandidata). Opisani proces ponavljamo sve dok ne pronađemo dovoljno dobrog kandidata za rješenje.

Dakle, gore navedeni proces zasniva se na dvije stvari:

- Operatori mutacije i rekombinacije stvaraju potrebnu raznolikost i time omogućavaju varijacije u populaciji
- Funkcija podobnosti omogućuje nam selekciju rješenja.

[1] Nabrojene u 0.1.2

Kombinacija te dvije pokretačke sile vodi do porasta podobnosti kandidata kroz generacije.

Primijetimo da je evolucijski proces stohastički. Iako je vjerojatnost odabira jedinki s višom podobnosti veća od vjerojatnosti odabira jedinki s manjom podobnosti, čak i te manje kvalitetne jedinke imaju pozitivnu vjerojatnost preživljavanja, kao i vjerojatnost da će postati roditelj. Naime, i kod odabira roditelja kod rekombinacije, i kod odabira jedinke za mutaciju, taj odabir se vrši nasumično. Algoritam 1 prikazuje pseudokod evolucijskog algoritma[2], dok slika 1.1 prikazuje shemu.

Algorithm 1: Evolucijski algoritam

```
INICIJALIZIRAJ populaciju sa nasumičnim kandidatima
IZRAČUNAJ podobnost za svakog kandidata
while nije zadovoljen UVJET ZAUSTAVLJANJA do
    Odaberi roditelje
    Izvrši REKOMBINACIJU roditelja
    MUTIRAJ potomstvo
    EVALUIRAJ nove kandidate
    ODABERI jedinke koje će sačinjavati sljedeću generaciju
end
```

Primijetimo tri bitne činjenice kod evolucijskih algoritama:

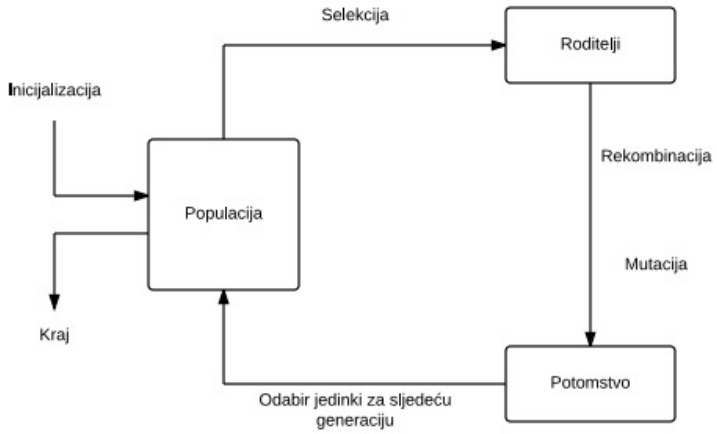
- Evolucijski algoritmi istovremeno procesuiraju cijelu populaciju, tj. sve kandidate za rješenja.
- Evolucijski algoritmi koriste rekombinaciju kako bi informacije iz više kandidata bile zastupljene u jednom, novom kandidatu.
- Evolucijski algoritmi su stohastički

1.2 Komponente evolucijskih algoritama

U ovom poglavlju detaljno ćemo opisati sve komponente, procedure i operatore koji moraju biti zadani kako bi EA bio dobro definiran. Posebno ističemo najvažnije među njima (također istaknuti tiskanim slovima u algoritmu 1):

- Reprezentacija jedinki

[2] U daljnjem tekstu EA



Slika 1.1: Shema evolucijskog algoritma

- Funkcija podobnosti
- Populacija
- Mehanizam odabira roditelja
- Operatori mutacije i rekombinacije
- Mehanizam odabira jedinki koje će preživjeti do sljedeće generacije.
- Način inicijalizacije populacije
- Uvjet zaustavljanja

Svaka od tih komponenti je nužna kako bi se definirao neki EA.

1.2.1 Reprezentacija jedinki

Kao prvi korak u definiranju EA moramo definirati vezu između prostora u kojem se nalazi problem i prostora u kojem će EA tražiti rješenje. Kandidate za rješenje unutar prostora u kojem se nalazi problem zovemo fenotipi, dok njihove reprezentante u EA zovemo genotipi. Reprezentacija je početni korak u kojem specificiramo preslikavanje iz skupa fenotipa u skup genotipa koji ih reprezentiraju. To preslikavanje mora biti invertibilno; za svaki genotip mora postojati najviše jedan fenotip. Kao trivijalan primjer možemo promatrati

problem optimizacije na $\mathbb{N}$. Tada je $\mathbb{N}$ skup fenotipa, dok kao primjer genotipa možemo promatrati binarni zapis broja. Važno je napomenuti da prostor u kojem se nalaze fenotipi može biti bitno različit od prostora u kojem se nalaze genotipi. EA rješava problem u prostoru genotipa; rješenje problema u fenotipskom prostoru dobivamo dekodiranjem genotipa nakon kraja EA.

1.2.2 Funkcija podobnosti

Funkcijom podobnosti reprezentiramo uvjete na koje se jedinka mora prilagoditi. Ona omogućuje selekciju, točnije pomoću nje možemo ocijeniti podobnost neke jedinke i definirati poboljšanje. To je funkcija koja svakom genotipu pridružuje mjeru njegove kvalitete. Matematički gledano, ta funkcija je kompozicija neke mjere kvalitete i inverza reprezentacije. Konkretno, držimo li se primjera iz prethodne cjeline, želimo li maksimizirati x^2 na $\mathbb{N}$, kvalitetu genotipa 10010 definiramo kao kvadrat odgovarajućeg fenotipa $18^2 = 324$.

1.2.3 Populacija

Populacija sadrži reprezentaciju nekih mogućih rješenja, točnije populacija je multiskup genotipa. Važno je istaknuti da se jedinke ne mijenjaju ni ne prilagođavaju; to radi populacija, dakle populacija je osnovna jedinica evolucije. U većini slučajeva, prvu populaciju definiramo nasumičnim generiranjem određenog broja jedinki, dok za neke, sofisticiranije EA možemo zadati i neke dodatne uvjete. U skoro svim primjenama EA veličina populacije je konstantna.

Raznovrsnost populacije je mjera broja različitih rješenja. Ne postoji jedinstvena mjera za raznovrsnost. Obično se koriste broj različitih vrijednosti podobnosti, broj različitih fenotipa i broj različitih genotipa. Također se mogu koristiti statističke mjere poput entropije. Primijetimo da postojanje samo jedne vrijednosti podobnosti ne znači nužno i postojanje samo jednog fenotipa te da postojanje samo jednog fenotipa ne znači nužno i postojanje samo jednog genotipa. Obrnuto, jednom genotipu pridružen je točno jedan fenotip te podobnost.

1.2.4 Mehanizam odabira roditelja

Uloga mehanizma odabira roditelja je da, među svim raspoloživim jedinkama, omogući odabir "boljih" roditelja (naravno, boljih u kontekstu bolje podobnosti). Zajedno s mehanizmom odabira jedinki koje će preživjeti do sljedeće generacije, odabir roditelja uzrok je poboljšavanja svojstava populacije kroz generacije. Napomenimo da je odabir roditelja najčešće probabilistički, pa iako pojedinci s boljom podobnosti imaju bitno veće šanse da budu odabrani, postoji i šansa da se za roditelje odaberu i manje kvalitetne jedinke. Iako

na prvi pogled nije jasna prednost takvog pristupa, kad bi uvijek uzimali samo najkvalitetnije pojedince, algoritam bi bio previše pohlepan i postojala bi velika šansa da zapnemo u lokalnom optimumu.

1.2.5 Operatori varijacije

Operatori varijacije stvaraju nove jedinke iz već postojećih. U odgovarajućem fenotipskom prostoru to je ekvivalentno generiranju novih kandidata za rješenje. Iz perspektive "generiraj i testiraj" algoritama, operatori varijacije odgovaraju "generiraj" koraku. Operatore varijacije u EA dijelimo na 2 tipa, ovisno o tome jesu li unarni ili binarni.

Operator mutacije

Unarni operator varijacije nazivamo operator mutacije. Ulazni podatak operatora mutacije je jedan genotip, a izlazni je modificirani genotip. Promjena između ta dva genotipa je obično mala. Mutacija je uvijek stohastički operator; genotip potomka je rezultat niza nasumičnih promjena. Važno je napomenuti da svaki unarni operator ne smatramo nužno mutacijom, ukoliko ne ispunjava i uvjete nepristranosti i nasumičnosti. Zbog toga se mnogi heuristički operatori ne smatraju mutacijama u strogom smislu riječi. Uloga mutacije se razlikuje u raznim granama EA, kasnije ćemo se detaljnije pozabaviti ulogom mutacije u genetskim algoritmima.

Generiranje potomka možemo promatrati i kao pomak na neku drugu točku u genotipskom prostoru. Iz te perspektive, operator mutacije ima i teorijsku ulogu, naime, on nam jamči da je prostor povezan (ukoliko ne bi imali operator mutacije, nikad ne bi mogli doći do nekih točaka u prostoru, konkretno onih točaka koje ne možemo dobiti kombinacijama već postojećih jedinki). Postoje teoremi [3] koji nam jamče da će EA naći globalni optimum (uz dovoljno vremena), uz uvjet povezanosti prostora. Taj zahtjev trivijalno zadovoljava operator mutacije definiran tako da u svakom koraku može, uz pozitivnu vjerojatnost, skočiti u bilo koju točku prostora.

Operator rekombinacije

Binarni [4] operator varijacije nazivamo operator rekombinacije (eng. recombination, crossover). Kao što i samo ime govori, ovaj operator spaja genetske informacije dva (ili više) roditelja u jedan ili više potomaka. Slično kao i mutacija, operator rekombinacije je stohastički; izbor dijelova koji će svaki roditelj prenijeti na potomke te način spajanja tih dijelova je nasumičan.

[3] vidi A.E. Eiben, E.H.L Aarts, K.M.Van Hee. Global convergence of genetic algorithms: a Markov chain analysis. In: Schwefel. Maenner [343]. pp 4-12

[4] lako se poopći i na n-arni. To poopćenje nema biološkog ekvivalenta.

Razlog zbog čega je rekombinacija poželjna je jednostavan; rezultat kombiniranja genetskog materijala više jedinki s različitim, ali poželjnim karakteristikama, je potomak koji bi mogao objedinjavati te karakteristike.[5] U EA, stvara se određeni broj potomaka nasumičnom rekombinacijom te se nadamo da će neki od njih imati bolje karakteristike od svojih roditelja. Operator rekombinacije najčešće se primjenjuje probabilistički. [6]

Na kraju, napomenimo još da za različite reprezentacije genotipa, moramo definirati različite operatore varijacije. Na primjer, ako je genotip niz bitova, operator mutacije možemo definirati kao operator koji će s određenom vjerojatnosti promijeniti 1 u 0 i obratno. Ukoliko je genotip reprezentiran na neki drugi način, npr. binarnim stablom, trebamo i drugačije definiran operator.

1.2.6 Mehanizam odabira jedinki koje preživljavaju do sljedeće generacije

Ovaj mehanizam omogućuje nam odabir jedinki koje će preživjeti, uzimajući u obzir njihovu podobnost ili neka druga svojstva. Po tome je sličan mehanizmu odabira roditelja, no obavlja se u drugoj fazi evolucijskog ciklusa 1.1. Kao što smo već spomenuli, veličina populacije je skoro uvijek konstantna, pa nam je potreban neki mehanizam koji će odlučivati koje jedinke će opstati do sljedeće generacije. Za razliku od mehanizma odabira roditelja, kao i operatora mutacije i rekombinacije koji su stohastički, ovaj operator je deterministički. Najčešće se odabire najboljih n jedinki (eng. fitness biased selection) ili se selekcija vrši s obzirom na dob (eng. age biased selection).

1.2.7 Inicijalizacija

Inicijalizacija je u većini implementacija EA relativno jednostavna. Prva populacija često je samo skup nasumično odabranih jedinki. No, možemo i iskoristiti neku heurističku metodu koja bi nam, ovisno o problemu dala kvalitetniju inicijalnu populaciju. No, postavlja se pitanje, je li uopće isplativo trošiti resurse na takvu vrstu inicijalizacije. O tome ćemo detaljnije raspravljati u 1.3.

1.2.8 Uvjet zaustavljanja

Ukoliko se bavimo problemom za koji nam je poznat globalni optimum, tada je najbolje kao uvjet zaustavljanja uzeti dovoljno dobru aproksimaciju tog optimuma (dakle približavanje optimumu na manje od nekog ϵ)

[5] Najbolji primjer korištenja ovog principa je razvoj raznih pasmina životinja i sorti biljaka.

[6] Biološki gledano, rekombinacija je superiorna mutaciji; većina složenijih organizama na Zemlji razmnožava se seksualno.

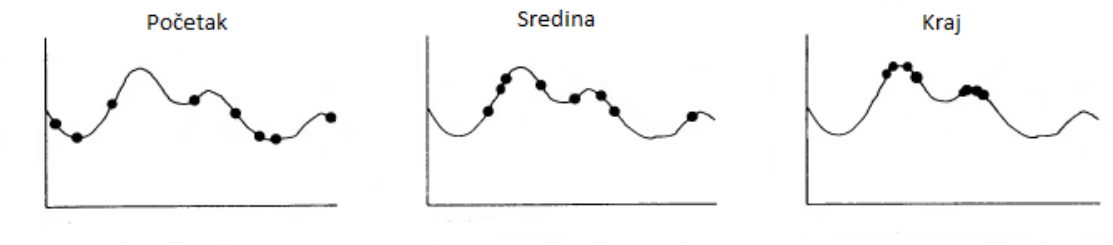
Međutim, kako su EA stohastički, uglavnom nije zajamčeno postizanje globalnog optimuma, pa se može dogoditi da taj uvjet nikad neće biti zadovoljen. Dakle, potrebno je dodati još jedan uvjet zaustavljanja koji sigurno zaustavlja algoritam. Obično se koriste neke od sljedećih opcija:

1. Prekoračeno vrijeme izvršavanja.
2. Prekoračen neki broj evaluacija podobnosti.
3. Za neki broj generacija (ili evaluacija podobnosti), poboljšanje podobnosti je manje od neke vrijednosti.
4. Raznolikost populacije pala je ispod neke vrijednosti.

Uvjet zaustavljanja najčešće je zadan kao disjunkcija: ili je postignut optimum, ili je zadovoljen neki od uvjeta 1-4.

1.3 Rad evolucijskog algoritma

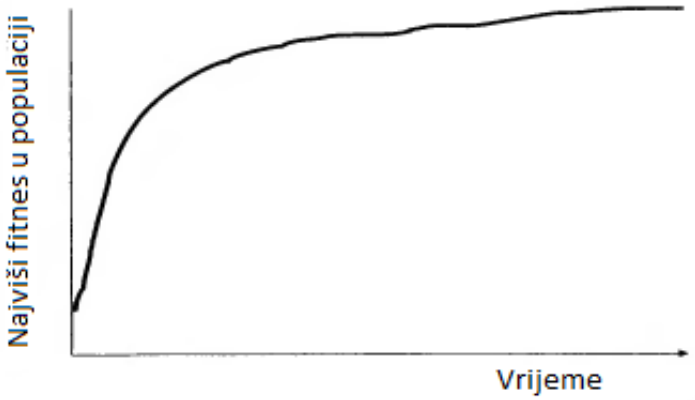
U ovom poglavlju iznijet ćemo neka generalna zapažanja o radu EA. Pretpostavimo da želimo maksimizirati jednodimenzionalnu funkciju cilja. Slika 1.2 prikazuje tri faze kroz koje prolazi evolucijski algoritam, konkretno kako je populacija distribuirana na početku, u sredini i na kraju evolucije. U prvoj fazi, odmah nakon inicijalizacije, jedinke su uniformno raspodijeljene na cijelom prostoru pretraživanja. Nakon početka rada EA situacija se mijenja. Zbog mutacije i rekombinacije populacija se postupno počinje ”penjati” prema područjima s boljom podobnosti. Još kasnije (blizu kraja pretraživanja, uz dobro zadane uvjete zaustavljanja), cijela populacija koncentrirala se oko nekoliko lokalnih maksimuma.



Slika 1.2: Pomicanje populacije tijekom evolucije. Na y osi nalazi se vrijednost podobnosti.

Različite faze traženja rješenja često se kategoriziraju s obzirom na procese istraživanja (eng. exploration) tijekom čega se generiraju nove jedinke u još neistraženim dijelovima prostorate iskorištavanja (eng. explotation) za vrijeme čega se pretraživanje koncentrira oko već postojećih dobrih rješenja. Kod traženja rješenja evolucijskim algoritmima moramo voditi računa o balansu te dvije komponente; koncentriramo li se previše na istraživanje, naš će algoritam biti neefikasan, dok će se u suprotnom slučaju algoritam prebrzo fokusirati na neki mali dio prostora što može dovesti do prerane konvergencije (eng. premature convergence) i zapinjanja u lokalnom optimumu.

Promotrimo graf 1.3 koji prikazuje kretanje vrijednosti podobnosti najbolje jedinke kroz vrijeme. Taj oblik krivulje karakterističan je za EA (kao i generalno za algoritme koji iterativno poboljšavaju rješenja) te lako možemo vidjeti kako je napredovanje algoritma u početku vrlo brzo te sve više usporava kroz vrijeme.



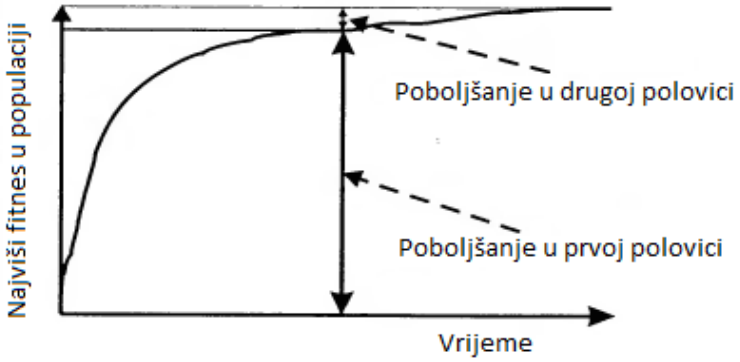
Slika 1.3: Prikaz tipičnog napredovanja podobnosti najbolje jedinke kroz vrijeme.

Sjetimo se sada razmatranja o isplativosti primjene neke heuristike kako bi dobili bolju početnu populaciju. U većini slučajeva takvo nešto je neisplativo, jer će se ta razina podobnosti ionako postići već kroz nekoliko prvih generacija algoritma 1.4.

Također, graf 1.3 može nam pomoći u razmatranjima kako odabrati uvjete zaustavljanja. U grafu 1.5 podijelili smo izvođenje algoritma na dvije jednako duge faze. Očito je da je poboljšanje podobnosti najbolje jedinke bitno veće u prvoj nego u drugoj fazi. Iz toga možemo zaključiti da je neisplativo algoritam provoditi kroz jako veliki broj koraka; napredak nakon određenog vremena može postati zanemariv.

Na kraju ovog poglavlja, promotrimo efikasnost EA na širokom spektru problema. Graf 1.6 prikazuje usporedbu EA s nasumičnim pretraživanjem, kao i algoritmom namijenjenim

Slika 1.4: Usporedba najbolje podobnosti u nasumično inicijaliziranoj populaciji (a) i u populaciji inicijaliziranoj pomoću neke heurističke metode (b). Primijetimo da je poboljšanje malo, tj. malen je k, broj generacija potreban EA da dosegne tu razinu.

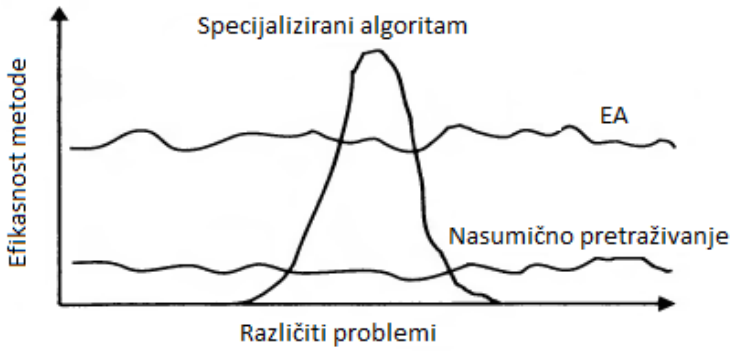


Slika 1.5: Usporedba napretka na početku algoritma i u kasnijoj fazi.

za neki konkretan problem. Iako EA nisu efikasni kao i algoritmi pisani za neki konkretan problem, njihova je najveća prednost podjednako dobra izvedba na širokom spektru problema. Postoji mogućnost kombinacije ta dva ekstrema u neku vrstu hibridnog algoritma.

1.3.1 Evolucijski algoritmi i globalna optimizacija

Kao što smo već spomenuli u uvodu, broj i kompleksnost problema s kojima se suočavamo u stalnom je i brzom porastu. Također, prisjetimo se da se EA često koriste kao alat za optimizaciju. Naravno, postoje i mnoge druge tehnike optimizacije te ćemo se u ovom



Slika 1.6: Usporedba EA, nasumičnog pretraživanja i algoritama za specifičan problem.

poglavlju pozabaviti usporedbom EA s ostalim metodama optimizacije.

Koristit ćemo pojam globalne optimizacije za proces pronalaženja rješenja x^* u skupu svih potencijalnih rješenja S, gdje x^* ima optimalnu vrijednost za neku funkciju podobnosti f. Drugim riječima, pokušavamo naći rješenje x^* tako da $x \neq x* \Rightarrow f(x^*) \geq f(x)$, u slučaju maksimizacije, dok u slučaju minimizacije samo okrenemo znak nejednakosti.

Primijetimo da postoje deterministički algoritmi, za koje, ukoliko im dozvolimo da se provedu do kraja, znamo da će pronaći egzaktno rješenje x^*. Trivijalan primjer je nabrajanje svih članova skupa S. No, primijetimo odmah i problem takvog pristupa, naime vrijeme potrebno za takvo rješavanje problema je najčešće preveliko za bilo kakvu praktičnu primjenu. Još jedan primjer determinističkih algoritama su takozvani box decomposition algoritmi, koji se baziraju na organizaciji elemenata iz S u neku vrstu stabla, nakon čega se na procjeni kvalitete rješenja u svakoj grani odlučuje hoćemo li provjeravati rješenja u toj grani. Iako takve metode mogu vrlo brzo doći do rješenja, u najgorem slučaju (kod suboptimalnog poretka pretraživanja), složenost je i dalje ista kao i kod nabrajanja.

Sljedeće, promotrimo heurističke metode, metode koje si možemo predočiti kao skup pravila koja govore koje će potencijalno rješenje iz S biti sljedeće generirano i testirano. Za neke heuristike, kao i za neke varijante EA, postoje dokazi o konvergenciji. No, njihov nedostatak je što ne mogu rješenje x^* prepoznati kao globalni optimum, već jednostavno kao najbolje dosad pronađeno rješenje, pa se postavlja pitanje kako prepoznati optimalan trenutak kada zaustaviti algoritam.

Takozvani algoritmi lokalnog pretraživanja uzimaju neko početno rješenje x te pretražuju $N(x)$[7] kako bi pronašli neki x' koji je bolje rješenje od x. Ukoliko takvo rješenje

[7] $N(x)$ je skup svih susjeda od x

postoji, tada ono postaje novo privremeno rješenje te se u sljedećem koraku pretražuje $N(x')$. Taj proces će dovesti do pronalaženja lokalnog optimuma. Prednost takvih algoritama[8] je u tome što brzo nalaze relativno dobra rješenja, što je često dovoljno za praktične primjene. Negativna strana je ta što često zapne u lokalnom optimumu, što je problematično kod problema s mnogo lokalnih optimuma, budući da nemamo nikakvu ocjenu kvalitete rješenja u globalu.

Glavna razlika između raznih algoritama za lokalno pretraživanje i EA je korištenje populacije. Populacija omogućuje algoritmu da definira neuniformnu raspodjelu kandidata za rješenje, što omogućuje da se novi kandidati za rješenja s većom vjerojatnošću generiraju u "boljim" područjima(za razliku od globalnog nasumičnog pretraživanja), no ujedno i izbjegava fokusiranje na područje oko samo jednog lokalnog ekstrema(za razliku od lokalnog pretraživanja). Također, prednosti čuvanja različitih kandidata u populaciji posebno dolazi do izražaja kod velikih i nepovezanih prostora pretraživanja. Osim toga, mogućnost čuvanja više primjeraka jednog rješenja pokazala se korisnom kod rada s funkcijama podobnosti kod kojih se pojavljuje šum ili neka vrsta nasumičnosti.

[8] tzv. hill climbers

Poglavlje 2

Genetski algoritmi

U ovom poglavlju fokusirat ćemo se na jednu podvrstu EA, genetske algoritme [1], koji su ujedno i najčešće korištena grana EA. Na početku ćemo razmotriti četiri različita načina reprezentacija kandidata za rješenje. Nakon toga, opisat ćemo operatore mutacije i rekombinacije prikladne za svaku od četiri navedene reprezentacije, dok ćemo na kraju promotriti različite mehanizme selekcije i zamjene koje koristimo kako bi regulirali populaciju mogućih rješenja. Kao što će se jasno vidjeti u ovom poglavlju, ne postoji jedan način za kreiranje GA; on nastaje kombinacijom različitih operatora koji su prikladni za rješavanje nekog konkretnog problema.

2.1 Reprezentacija

2.1.1 Reprezentacija jedinki

Kao što smo spomenuli u prethodnom poglavlju, prvi korak u konstrukciji bilo kojeg EA je reprezentacija kandidata za rješenje. To uključuje definiciju genotipa te preslikavanja s genotipa u fenotip. U ovoj cjelini pobliže ćemo se pozabaviti neki uobičajenim načinima reprezentacije. Napomenimo kako se u praksi često koriste i kombinacije ovdje navedenih reprezentacija, budući da je to često najprirodniji način na koji se neki problem može prikazati.

Binarna reprezentacija

Prvi način reprezentacije koji ćemo analizirati, ujedno je i najjednostavniji; već smo ga spomenuli ranije u radu. U njemu je genotip prikazan kao string binarnih znamenki.

[1] U daljnjem tekstu GA

Za konkretan problem, prvo moramo odlučiti kolika će nam biti dužina stringa te kako ćemo ga interpretirati kako bi dobili fenotip. Kod odabira preslikavanja genotipa u fenotip, moramo osigurati da se svi stringovi preslikavaju u valjano rješenje problema te obrnuto, da sva rješenja mogu biti prikazana.

Kod problema u kojima se javljaju Booleove varijable, preslikavanje genotipa u fenotip je prirodno, no bit-stringovi se koriste i za kodiranje informacija koje izvorno nisu u binarnom obliku. Npr. bit-string duljine 80 možemo interpretirati kao 10 8-bitnih prirodnih brojeva ili pet 16-bitnih realnih brojeva. Napomenimo kako se obično bolji rezultati dobiju direktnom reprezentacijom kandidata za rješenje kao prirodnih ili realnih brojeva.

Jedan od problema s kojim se susrećemo kod reprezentacije brojeva u binarnom obliku je taj što bitovi na različitim mjestima imaju različitu vrijednost. Kod problema u kojem radimo s prirodnim brojevima, želimo da nam je vjerojatnost mutacije broja 7 u broj 8 ista kao i vjerojatnost mutacije broja 7 u broj 6. No, ukoliko reprezentiramo te brojeve u binarnom obliku, vjerojatnost promjene 0111 u 1000 je očito puno manje od vjerojatnosti promjene 0111 u 0110. Taj problem može se riješiti upotrebom Grayevog kodiranja. To je način reprezentiranja koji jamči da susjedni prirodni brojevi uvijek imaju Hammingovu udaljenost jednaku 1 [2].

Reprezentacija prirodnim brojevima

Promotrimo sada problem gdje svaki gen može poprimiti neku vrijednost iz skupa s više od 2 člana. Npr. promotrimo problem traženja optimalnog puta na ortogonalnoj mreži. Tada možemo vrijednosti gena prikazati kao 0,1,2,3 što reprezentira gore, dolje, lijevo, desno. Taj skup čak ne mora biti ni ograničen; rješenja možemo tražiti na čitavom $\mathbb{N}$. U svakom slučaju, prirodnije je kandidate za rješenja reprezentirati direktno kao prirodan broj nego kao bit-string. Vratimo se još malo na primjer s kretanjem po ortogonalnoj mreži. Još jedno pitanje koje je vrijedno razmotriti je kako definirati "bliske" smjerove. Konkretno, hoćemo li kod operatora mutacije dozvoliti direktnu promjenu iz smjera sjevera u smjer juga, ili ćemo dozvoliti samo promjenu u susjedne smjerove. To pitanje možemo poopćiti; postoji li za svaki skup vrijednosti koje gen može poprimiti neka prirodna relacija bliskosti?

Reprezentacija realnim brojevima

Ovaj način reprezentiranja kandidata za rješenje zapravo je identičan reprezentaciji prirodnim brojevima, uz razliku što ovdje vrijednosti koje želimo reprezentirati genima potječu iz neprekidne, a ne iz diskretne distribucije.

[2] Hammingova udaljenost (eng. Hamming measure) za 2 stringa jednake duljine definira se kao broj pozicija na kojima su oni različiti

Reprezentacija permutacijama

Rješavanje mnogih problema svodi se na određivanje poretka u kojem se moraju odigrati određeni događaji. Prirodan način reprezentacije takvih problema je kao permutacija skupa prirodnih brojeva. Uočimo, za razliku od reprezentacije skupom prirodnih brojeva, u ovom slučaju svaki se broj može pojaviti samo jednom te tome moramo prilagoditi i operatore mutacije i rekombinacije.

Postoje 2 klase problema za koje se najčešće koristi reprezentacija permutacijama. U prvoj klasi nalaze se problemi u kojima je ključan poredak događaja, npr. kad imamo limitirane resurse ili vrijeme, a završetak nekih zadataka je bitniji od drugih. Tako npr. niz $[1, 2, 3, 4]$ može imati bitno različitu vrijednost funkcije podobnosti od $[4, 1, 2, 3]$. U drugu klasu problema spadaju problemi u kojima je fokus na pojmu susjedstva ili bliskosti dva alela.[3] Tipičan predstavnik tog tipa problema je problem trgovačkog putnika; dakle problem pronalaska potpunog obilaska n gradova minimalne duljine. Možemo odmah vidjeti bitnu razliku u odnosu na prvu klasu problema; naime $[1, 2, 3, 4]$ i $[4, 1, 2, 3]$ imaju istu podobnost; poredak obilaska nije bitan, bitna je samo ukupna duljina puta.

Postoje 2 načina kodiranja reprezentacije permutacijama. Prva, i najčešće korištena, je ona u kojoj i-ti element niza reprezentira događaj koji se dogodio i-ti po redu, dok u drugom vrijednost i-tog elementa reprezentira poziciju na kojoj se dogodio i-ti događaj. Npr. za problem trgovačkog putnika na 4 grada $[A, B, C, D]$, permutacija $[2, 3, 1, 4]$ u prvom načinu kodiranja označava ciklus $[B, C, A, D]$, dok u drugom načinu kodiranja odgovara ciklusu $[C, A, B, D]$.

2.2 Operatori varijacije

2.2.1 Mutacija

Mutacija je zajedničko ime za sve operatore koji iz genotipa samo jednog roditelja kreiraju samo jedno dijete, uz pomoć neke vrste nasumične promjene. U ovom poglavlju opisat ćemo razne vrste operatora mutacije.

Mutacija kod binarne reprezentacije

Uz nekoliko iznimaka, najčešće korišten operator mutacije kod birarnog kodiranja je operator koji zasebno promatra svaki gen te svakom daje malu vjerojatnost p_m[4] da je promijeni

[3] Alel je oblik pojedinog gena. Npr. mogući aleli svakog gena u binarnom prikazu su 0 i 1.

[4] p_m nazivamo parametar mutacije

iz 0 u 1 ili obrnuto. Broj promjena nije fiksiran; za string duljine L u prosjeku će se promijeniti $L * p_m$ vrijednosti. Slika 2.1 prikazuje string u kojem je doslo do mutacije na poziciji 6.

Slika 2.1: Primjer mutacije bit-stringa

Problem biranja optimalnog parametra p_m je dobro proučen; uglavnom ovisi o tipu problema s kojim se susrećemo. Npr. želimo li populaciju u kojoj sve jedinke imaju visoku podobnost, ili želimo pronaći samo jednu takvu jedinku. No, možemo reći da se generalno uzima takav parametar da u prosjeku dolazi do između jedne mutacije po generaciji i jedne mutacije u potomstvu.

Mutacija kod reprezentacija prirodnim brojevima

Kod reprezentacije prirodnim brojevima postoje 2 glavna oblika mutacije. Oba ta oblika nezavisno promatraju svaki gen te ga mutiraju s vjerojatnošću p_m.

Nasumično resetiranje

U ovom slučaju promjenu bitova iz prethodnog poglavlja proširujemo tako da novu vrijednost nasumično biramo iz skupa dozvoljenih vrijednosti na svakoj poziciji. Ovaj tip mutacije pogodan je kad nije bitno koliko ćemo se mutiranjem pomaknuti u prostoru, budući da je vjerojatnost odabira jednaka za svaku vrijednost.

Mutacija pomakom

Ovdje mutaciju implementiramo kao mali pomak (pozitivni ili negativni) od trenutne vrijednosti za svaki gen s nekom vjerojatnosti p_m. Vrijednost pomaka se obično svaki put nasumično bira iz neke distribucije simetrične oko 0, za koju je vjerojatnost odabira malog parametra veća od vjerojatnosti odabira velikog parametra (npr. normalna distribucija). Problem biranja optimalne veličine pomaka nije trivijalan te se ponekad koristi više od jednog operatora. Uobičajena rješenja su korištenje tzv. malog i velikog pomaka, ili korištenje malog pomaka u paru s operatorom nasumičnog resetiranja. Naravno, uloga malog pomaka je fino ugađanje rješenja, dok je funkcija velikog pomaka, kao i operatora nasumičnog resetiranja omogućavanje bržeg pretraživanja cijelog prostora, kao i bijeg iz lokalnog minimuma.

Operatori mutacije kod reprezentacija realnim brojevima

U ovom slučaju, operatori mutacije definirani su tako da vrijednost svakog alela nasumično mijenjaju unutar njegove domene ($[L_i, U_i]$). Dakle, rezultat mutacije je sljedeća transformacija:

$$< x_1, \ldots, x_n > \rightarrow < x'_1, \ldots x'_n >, \quad x_i, x'_i \in [L_i, U_i]$$

Ponovno razlikujemo 2 slučaja:

Uniformna mutacija

Kod ovog operatora, vrijednosti x'_i nasumično se izvlače iz $[L_i, U_i]$. To je najjednostavniji tip mutacije, analogan promjeni bitova kod binarne reprezentacije i nasumičnom resetiranju kod prirodnih brojeva.

Neuniformna mutacija s fiksnom distribucijom

Ovo je najčešće korišten tip mutacije kod reprezentacija realnim brojevima. Analogan je mutaciji pomakom kod prirodnih brojeva. Kao i u mutaciji pomakom, trenutnoj vrijednosti alela dodajemo neku vrijednost. Opet, želimo da je pomak u većini slučajeva malen, što postižemo podešavanjem parametara distribucije iz koje izvlačimo vrijednost pomaka. Također, napomenimo da se ovaj tip mutacije najčešće primijenjuje s vjerojatnosti 1 po genu, dok parametrom mutacije kontroliramo standardnu devijaciju (najčešće Gaussove) distribucije.

Operatori mutacije za reprezentacije permutacijama

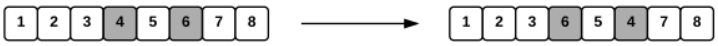
Kod reprezentacija permutacijama, više nije moguće nezavisno promatrati svaki gen (npr. rezultat mutacije samo jednog alela više nije permutacija). Umjesto toga, mutacija će pomicati alele po genomu. Posljedica toga je da će parametar mutacije u ovom slučaju biti vjerojatnost podvrgavanja stringa mutaciji, umjesto vjerojatnosti da mutaciji podvrgavamo neki gen. Promotrit ćemo 4 tipa mutacija.

Mutacija zamjenom

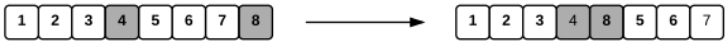
U ovom tipu mutacije nasumično biramo 2 pozicije u stringu i zamijenimo njihove vrijednosti. Vidi sliku 2.2.

Mutacija umetanjem

U ovom slučaju nasumično odaberemo 2 alela, jednog pomaknemo tako da je pokraj drugog, a ostale pomaknemo za jedno mjesto, ukoliko je to potrebno. Primjer je dan na slici 2.3.



Slika 2.2: Primjer mutacije zamjenom



Slika 2.3: Primjer mutacije umetanjem

Mutacija permutiranjem

Ovdje je cijeli string, ili neki njegov podskup nasumično permutiran. Na slici 2.4 dan je primjer s permutiranim podskupom od treće do šeste pozicije.

Slika 2.4: Primjer mutacije permutiranjem

Mutacija invertiranjem

Mutacija invertiranjem funkcionira na sljedeći način: nasumično odaberemo 2 pozicije u stringu te zamijenimo poredak vrijednosti između njih. Primjer mutacije invertiranjem dan je na slici 2.5, ponovo između pozicija 3 i 6.

Slika 2.5: Primjer mutacije invertiranjem

2.2.2 Rekombinacija

Rekombinacija je proces u kojem je novo rješenje dobiveno kombinacijom genetskog materijala 2 (ili više) roditelja te je jedna od komponenti EA po kojoj se razlikuju od ostalih

algoritama za globalnu optimizaciju. Često se umjesto termina rekombinacija koristi eng. crossover, iako je strogo gledano crossover slučaj rekombinacije s točno dva roditelja.

Operator rekombinacije obično se primijenjuje s vjerojatnošću p_c [5], obično $p_c \in [0.5, 1]$. Uobičajen postupak je sljedeći: odaberu se 2 roditelja te se nasumično odabere neki broj iz $[0, 1]$ i usporedi s p_c. Ukoliko je taj broj manji od p_c, rekombinacijom se dobiju 2 potomka, dok se u suprotnom slučaju kopiraju roditelji. Posljedica toga je da se u skupu potomaka nalaze neke nove jedinke, ali i neke kopije već postojećih. Dakle, za razliku od parametra mutacije p_m, koji daje vjerojatnost da će dijelovi kromosoma biti mutirani, nezavisno jedni od drugih, p_c daje vjerojatnost da će neki izabrani par roditelja biti podvrgnut operatoru rekombinacije.

Operatori rekombinacije za binarne reprezentacije

Tri standardna oblika rekombinacije koriste se kod binarnih reprezentacija. Pobliže ćemo se pozabaviti slučajevima u kojima od 2 roditelja dobivamo dvoje potomaka, što se lako poopći.

Rekombinacija u jednoj točci

U ovom tipu operatora rekombinacije odabiremo nasumičan broj iz $[0, l-1]$, gdje je l duljina stringa te nakon toga "presječemo" oba roditelja u toj točci i zamijenimo njihove zadnje dijelove. (slika 2.6)

Slika 2.6: Primjer rekombinacije u jednoj točci

Rekombinacija u n točaka

Prirodna generalizacija rekombinacije u jednoj točci, string smo presjekli na više od jedne točke te potomke dobili tako da smo alternirali dijelove iz prvog i drugog roditelja. Na slici 2.7 dan je primjer rekombinacije u 2 točke.

Primijetimo da rekombinacija u n točaka ima tendenciju da one gene koji su na bliskim pozicijama ostavlja zajedno u potomcima; štoviše, za parni n to se događa i s genima koji su na suprotnim stranama stringova. Sljedeći način rekombinacije imun je na taj problem.

[5] Vjerojatnost rekombinacije, eng. crossover rate

Slika 2.7: Primjer rekombinacije u 2 točke

Uniformna rekombinacija

Dok su prethodna 2 operatora dijelila roditelje na neki broj podskupova gena roditelja i spajala ih kako bi dobili potomke, kod ove metode svaki gen promatramo zasebno te nasumično biramo od kojeg ćemo ga roditelja naslijediti. Ta se metoda najčešće implementira pomoću stringa od L slučajnih varijabli iz uniformne distribucije na $[0, 1]$. Na svakoj poziciji, ukoliko je vrijednost slučajne varijable ispod neke vrijednosti (najčešće 0.5) kopiramo taj gen iz prvog roditelja, inače ga kopiramo od drugog. Drugi potomak dobije se suprotnim postupkom. Vidi sliku 2.8

Slika 2.8: Primjer uniformne rekombinacije

Operatori rekombinacije za reprezentacije prirodnim brojevima

Kod ovog tipa reprezentacije, operatori rekombinacije definiraju se identično kao i kod binarne reprezentacije.

Operatori rekombinacije za reprezentacije realnim brojevima

Postoje 2 opcije za rekombinaciju stringova realnih brojeva:

- Analogna definicija operatora kao i u ostalim slučajevima. Operatori tog tipa poznati su i kao diskretne rekombinacije. Primijetimo da je glavni nedostatak ovako definiranih operatora taj što je mutacija jedini način kojim se u populaciju mogu uvesti nove vrijednosti; drugim riječima, ako iz roditelja x i y kreiramo dijete z vrijedi $z_i = x_i$ ili y_i.

- Operator koji, za svaki gen, kreira novu vrijednost alela kod potomka. Dakle, $z_i = \alpha x_i + (1-\alpha)y_i$, za neki $\alpha \in [0,1]$. Prednost ovako definiranog operatora je mogućnost stvaranja novog genetskog materijala. S druge strane, primijetimo da se u svakoj novoj generaciji, jer je z_i uvijek između x_i i y_i, smanjuje raspon vrijednosti alela. Operatori tog tipa poznati su i kao aritmetičke rekombinacije. Primijetimo da ovaj tip operatora ne možemo definirati na binarnim reprezentacijama i na reprezentacijama prirodnim brojevima, budući da se može dogoditi da z_i definiran na gore opisan način više nije binaran, odnosno prirodan broj.

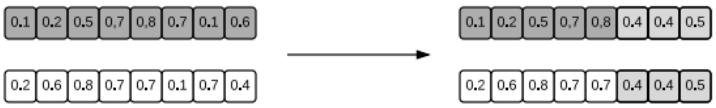
Aritmetička rekombinacija

Tri tipa aritmetičke rekombinacije pobliže ćemo opisati u ovom poglavlju. U sva 3 slučaja, parametar α može se odabrati nasumično nad $[0,1]$, ali najčešće se koristi neka konstanta. Ukoliko je $\alpha = 0.5$ tada se radi o uniformnoj aritmetičkoj rekombinaciji.

Jednostavna rekombinacija

Prvo biramo točku rekombinacije k. Tada, prvih k vrijednosti iz roditelja 1 kopiramo u dijete 1, a prvih k vrijednosti iz roditelja 2 kopiramo u dijete 2. Ostatak mjesta u oba djeteta popunimo aritmetičkim sredinama roditelja.

$$\text{Dijete 1: } < x_1, \ldots, x_k, \alpha y_{k+1} + (1-\alpha)x_{k+1}, \ldots, \alpha y_n + (1-\alpha)x_n >$$
$$\text{Dijete 2: } < y_1, \ldots, y_k, \alpha x_{k+1} + (1-\alpha)y_{k+1}, \ldots, \alpha x_n + (1-\alpha)y_n >$$



Slika 2.9: Primjer jednostavne aritmetičke rekombinacije, $k = 5, \alpha = 0.5$

Rekombinacija na jednom mjestu

Odaberemo neki alel k. Na toj poziciji, u oba djeteta stavimo prosjek alela roditelja na toj poziciji. Ostale alele prepišemo od roditelja.

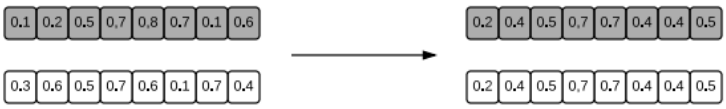
$$\text{Dijete 1: } < x_1, \ldots x_k, \alpha y_k + (1-\alpha)x_k, x_{k+1}, \ldots, x_n >$$
$$\text{Dijete 2: } < y_1, \ldots y_k, \alpha x_k + (1-\alpha)y_k, y_{k+1}, \ldots, y_n >$$

Uniformna rekombinacija

Ovo je najčešće korištena aritmetička rekombinacija. Kreira 2 djeteta na sljedeći način.

$$\text{Dijete 1: } \alpha\vec{x} + (1-\alpha)\vec{y}$$
$$\text{Dijete 2: } \alpha\vec{y} + (1-\alpha)\vec{x}$$

Slika 2.10: Primjer aritmetičke rekombinacije na jednom mjestu, $k = 2, \alpha = 0.5$



Slika 2.11: Primjer uniformne aritmetičke rekombinacije, $\alpha = 0.5$

Operatori rekombinacije za reprezentacije permutacijama

Već na prvi pogled jasno je da za ovakav tip reprezentacija ne možemo operator rekombinacije dizajnirati na isti način kao u prethodnim slučajevima. Zbog toga postoje specijalno dizajnirani operatori specijalizirani za rad s permutacijama. Ovdje ćemo detaljno opisati 2 najpoznatija takva operatora.

Parcijalno križanje

Parcijalno križanje prvi su upotrijebili Goldberg i Lingle kao operator rekombinacije za problem trgovačkog putnika u [8], nakon čega su se pojavile mnoge varijacije tog algoritma. Slijedi algoritam jedne od varijacija, upotrijebljene od strane Whitleya. Grafički prikaz dan je na slikama 2.12, 2.13, 2.14.

1. Nasumično odaberemo 2 točke križanja (T1, T2), segment između njih kopiramo iz prvog roditelja (R1) u prvo dijete (D1).

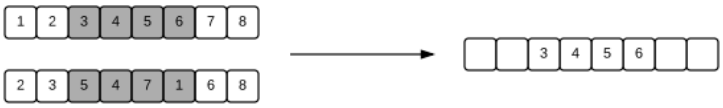
2. Počevši od T1 tražimo elemente u drugom roditelju (R2) koji još nisu kopirani.

3. Za svaki od tih elemenata i, u D1 pogledamo koji element $j \in R1$ je kopiran na njegovo mjesto.

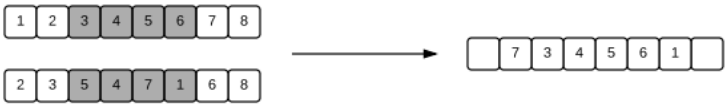
4. Stavimo i na poziciju koju je j zauzimao u R2. Znamo da na tu poziciju nećemo stavljati j jer njega već imamo u djetetu.

5. Ako je mjesto koje j zauzima u R2 već zauzeto u potomku nekim elementom k, stavimo i na poziciju koju k zauzima u R2.

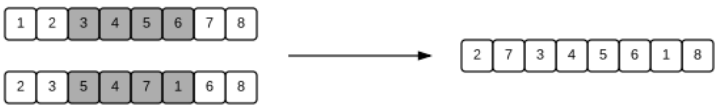
6. Time smo završili s elementima koji se pojavljuju u segmentu križanja. Ostatak potomka popunimo elementima iz R2, dok drugo dijete kreiramo analogno, sa zamijenjenim ulogama roditelja.



Slika 2.12: Parcijalno križanje, korak 1.



Slika 2.13: Parcijalno križanje, korak 2. Primijetimo, 7 je prvi element u R2 koji se ne nalazi u srednjem segmentu u R1. Pozicija koju 7 zauzima u R2 u djetetu zauzima 5. Sada gledamo na kojoj je poziciji u R2 5. Vidimo da tu poziciju u djetetu već zauzima 3, pa tražimo poziciju koju u R2 zauzima 3. konačno, to je pozicija 2, koja je slobodna, pa na nju možemo staviti 7. Analogno ponavljamo za 1.



Slika 2.14: Parcijalno križanje, rezultat.

Promotrimo sada susjedne alele u roditeljima. Vidimo da su u oba roditelja susjedne vrijednosti 2-3 i 4-5. No, u D1, samo je veza 4-5 ostala sačuvana. To narušava tzv. svojstvo sačuvanja veza koje kaže da svaka informacija prisutna u oba roditelja također treba biti prisutna i u njihovim potomcima. Razmislimo li, primjećujemo da je to svojstvo zadovoljeno za sve dosad opisane operatore, osim za parcijalno križanje. No, postoji nekoliko operatora za permutacijske reprezentacije koji zadovoljavaju i to svojstvo. Opisujemo ih u nastavku.

Križanje sa sačuvanim vezama

Križanje sa sačuvanim vezama (eng. Edge Crossover u daljnjem tekstu EC) se zasniva na ideji da, koliko god je to moguće, u potomstvu trebaju biti susjedni samo oni elementi koji već imaju to svojstvo u bar jednom od roditelja. Ovdje ćemo ukratko opisati jednu od verzija tog algoritma.

Konstruirat ćemo tzv. tablicu veza, koja za svaki element daje sve elemente koji su njemu susjedni u bar jednom roditelju. "+" u tablici znači da je veza prisutna u oba roditelja. Operator radi na sljedeći način:

1. Konstruiraj tablicu veza.
2. Nasumično odaberi početni element i stavi ga u dijete.
3. Postavi *trenutni-element=unos*.
4. U tablici obriši sva pojavljivanja *trenutni-element*
5. Za *trenutni-element*
   - Ako postoji veza prisutna u oba roditelja, nju odaberemo kao sljedeći element.
   - Inače odaberemo element povezan s trenutnim elementom koji ima najmanje drugih veza.
   - Ukoliko 2 elementa zadovoljavaju neki od gornjih uvjeta nasumično odaberemo jednog od njih.
6. Ukoliko dođemo do elementa za kojeg više nemamo nijednu vezu u popisu, dijete nadopunjavamo na drugu stranu. Ukoliko ni to ne možemo napraviti, novi element biramo nasumičnim odabirom.

Očito, samo u zadnjem slučaju može doći do kreiranja novih veza. Primjer EC dat ćemo na ista 2 roditelja kao i ranijim primjerima, dakle $[1, 2, 3, 4, 5, 6, 7, 8]$ i $[2, 3, 5, 4, 7, 1, 6, 8]$. Tablica veza prikazana je na tablici 2.1, dok je konstrukcija djeteta detaljno prikazana u tablici 2.2.

Križanje s očuvanim poretkom

Početak je sličan kao i kod parcijalnog križanja, no drugi dio algoritma je različit; ovdje je cilj sačuvati informacije o relativnom poretku elemenata niza. Navodimo korake algoritma:

1. Nasumično odaberemo 2 točke križanja, kopiramo segment između te dvije točke iz R1 u D1.

Element	Veze
1	2,6,7,8
2	1,3+,8
3	2+,4,5
4	3,5+,7
5	3,4+,6
6	1,5,7,8
7	1,4,6,8
8	1,2,6,7

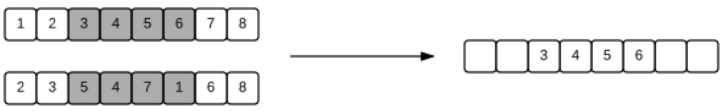
Tablica 2.1: Tablica veza

Mogući izbori	Izbor	Razlog	Dijete
1-8	1	Nasumično	[1]
2,6,7,8	2	Najmanje veza	[1,2]
3,5	3	Zajednička veza	[1,2,3]
4,5	5	Nasumično	[1,2,3,5]
4,6	4	Zajednička veza	[1,2,3,5,4]
7	7	Jedini izbor	[1,2,3,5,4,7]
6,8	8	Nasumično	[1,2,3,5,4,7,8]
6	6	Jedini izbor	[1,2,3,5,4,7,8,6]

Tablica 2.2: Konstrukcija djeteta kod EC

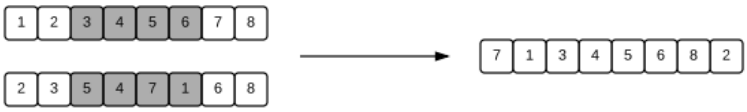
2. Počevši od druge točke križanja u R2, prepisujemo preostale neiskorištene brojeve u prvo dijete u poretku istom kao u R2. Kad dođemo do kraja R2, nastavljamo od početka.

3. D2 kreiramo analogno, sa zamijenjenim ulogama R1 i R2.



Slika 2.15: Križanje s očuvanim poretkom, korak 1.

Cikličko križanje



Slika 2.16: Križanje s očuvanim poretkom, korak 2.

Posljednji tip križanja s kojim ćemo se detaljnije pozabaviti je cikličko križanje. Ono se koristi kada želimo što bolje sačuvati informacije o tome na kojoj se poziciji nalazi koji element. Na početku podijelimo elemente u cikluse. Ciklus je podskup elemenata koji ima svojstvo da je, kod poravnatih roditelja, svaki element ciklusa na istoj poziciji u nekom od roditelja kao i još neki element iz tog ciklusa u drugom roditelju. Nakon što smo podijelili permutaciju u cikluse, potomke stvaramo naizmjenično birajući cikluse iz svakog roditelja. Opišimo sada detaljnije proces stvaranja ciklusa:

1. Počinjemo s prvom neiskorištenom pozicijom U R1.

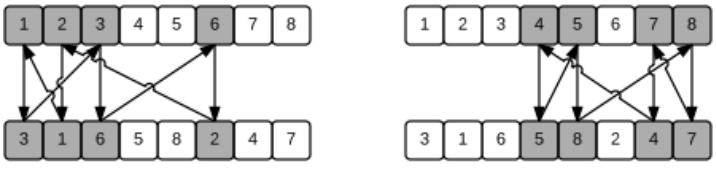
2. Promotrimo alel na *istoj poziciji* u R2.

3. Pomaknemo se na poziciju s *istim alelom* u R1.

4. Dodamo taj alel u ciklus.

5. Ponavljamo korake 2-4 dok se ne vratimo na početak.

Sljedeće 2 slike prikazuju identifikaciju ciklusa na roditeljima $[1, 2, 3, 4, 5, 6, 7, 8]$ i $[3, 1, 6, 5, 8, 2, 4, 7]$ (slika 2.17) i konstrukciju potomstva (slika 2.18)



Slika 2.17: Ciklično križanje, identifikacija ciklusa.

Slika 2.18: Ciklično križanje, kreiranje djece.

Rekombinacija s više roditelja

Dosad smo se bavili unarnim (mutacija) i binarnim (križanje) operatorima varijacije. Prirodno se nameće ideja popćenja operatora rekombinacije na više (n) roditelja. To popćenje je relativno lako definirati i implementirati, iako ne postoji direktan biološki ekvivalent. N-arne operatore rekombinacije možemo kategorizirati s obzirom na osnovni mehanizam koji se koristi kod kombiniranja informacija iz roditelja. Ovdje ćemo samo navesti različite kategorije te dati reference na radove u kojima se može detaljnije proučiti svaki od njih:

- Operatori bazirani na frekvencijama alela [12], generaliziraju uniformno križanje.
- Operatori bazirani na segmentaciji i rekombinaciji roditelja, generaliziraju križanje u n točaka. [4]
- Operatori bazirani na numeričkim operacijama na alelima s realnim vrijednostima, koji generaliziraju aritmetičke operatore rekombinacije. [16]

Povećanje broja roditelja u rekombinaciji ne mora nužno dovesti do poboljšanja performansi EA, to uvelike ovisi o tipu rekombinacije, kao i o samom problemu. No, velik broj eksperimenata na raznim problemima provedenih u [5] pokazuju da je korištenje više od 2 roditelja najčešće korisno.

2.3 Modeli populacije

Fokusirajmo se sada na još jedan bitan dio svakog GA, način na koji vršimo selekciju jedinki koje će preživjeti prijelaz iz generacije u generaciju, baziran na njihovoj podobnosti. Postoji jasna razlika između 2 modela GA; generacijskog modela i modela stacionarnog stanja.

Promotrimo najprije generacijski model. U svakoj generaciji imamo populaciju veličine μ iz koje odabiremo μ roditelja (mogu se i ponavljati). Nakon toga, kreiramo $\lambda(=\mu)$ poto-

maka pomoću operatora varijacije. Nakon svake generacije, cijela se generacija zamijeni sa svojim potomstvom. [6]

U modelu stacionarnog stanja, cijela populacija se ne mijenja istovremeno. Umjesto toga mijenjamo $\lambda(< \mu)$ jedinki iz prethodne generacije sa λ novih. Postotak zamijenjenih jedinki nazivamo generacijski jaz (eng. generation gap), λ/μ

Primijetimo da se operatori selekcije i zamjene koji nam omogućuju gore opisan postupak baziraju na podobnosti jedinki, dakle za razliku od operatora varijacije uzimaju u obzir cijelu jedinku, umjesto samo neke gene. Kao posljedica toga operatori selekcije i zamjene nezavisni su o reprezentaciji problema. Također napomenimo da postoje 2 mjesta u evolucijskom ciklusu (vidi sliku 1.1) gdje možemo primijeniti te operatore: tijekom selekcije jedinki koje će se razmnožavati te tijekom selekcije jedinki koje će preživjeti smjenu generacija.

2.4 Selekcija roditelja

2.4.1 Selekcija proporcionalna podobnosti

U ovom načinu selekcije vjerojatnost da jedinka f_i bude odabrana za razmnožavanje iznosi $f_i / \sum_{j=1}^{\mu} f_j$, dakle odnos podobnosti jedinke i podobnosti cijele populacije. No, kod ove metode nailazimo na neke poteškoće:

- Jedinke s vrlo velikom podobnosti prejako utječu na cijelu populaciju. To je poznato kao prijevremena konvergencija.
- Kada je podobnost svih jedinki relativno bliska, selekcija je skoro nasumična s istom vjerojatnosti za sve jedinke, što može dovesti do jako sporog napretka nakon što su najgore jedinke već izbačene.
- Na za konstantu pomaknutim funkcijama podobnosti ne dobivamo iste rezultate.

Postoje 2 načina izbjegavanja ovih problema, spomenimo prvo tzv. windowing. Od podobnosti $f(x)$ oduzimamo neku vrijednost β. Najčešće se uzima $\beta = min_{y \in P^t} f(y)$, tj. oduzimamo najmanju vrijednost podobnosti u trenutnoj populaciji P^t. Kako ta vrijednost može dosta varirati kroz generacije, možemo uzimati i prosjek zadnjih par generacija.

Druga metoda je tzv. Goldbergovo sigma skaliranje, koje u obzir uzima medijan $\bar{f}$ i standardnu devijaciju σ_f svih vrijednosti podobnosti u populaciji:

$$f'(x) = max(f(x) - (\bar{f} - c * \sigma_f), 0)$$

gdje je c neka konstanta.

[6] μ i λ su standardne oznake za veličinu populacije, odnosno broj potomaka.

2.4.2 Selekcija po rangu

Ideja ove metode je selekcija jedinki bazirana na njihovom poretku u populaciji, umjesto na samoj vrijednosti podobnosti; prvo poredamo sve jedinke po vrijednosti podobnosti, a nakon toga im pridružimo neku vjerojatnost odabira. Preslikavanje kojim rangu pridružujemo vjerojatnost može biti proizvoljno.

Uobičajena formula za računanje te vjerojatnosti dana je s

$$P_{lin_rank}(i) = \frac{(2-s)}{\mu} + \frac{2i(s-1)}{\mu(\mu-1)},$$

gdje je najgora jedinka na broju 1, najbolja na broju μ, dok je $1 \leq s \leq 2$ parametar. Primijetimo da je drugi član u gornjoj formuli proporcionalan rangu jedinke, dakle što je taj član veći, veći je selekcijski pritisak, tj. biranje kvalitetnijih jedinki. Dakle ukoliko je s bliže 2, povećavamo selekcijski pritisak, dok se za $s = 1$ naš odabir vrši nasumično.

2.4.3 Turnirska selekcija

Prethodne 2 metode zasnivale su se na podacima o cijeloj populaciji. Međutim, ako je populacija vrlo velika, doći do podataka o cijeloj populaciji može biti vrlo zahtjevno. Ponekad čak nije ni moguće definirati podobnost kao vrijednost (Zamislimo problem traženja optimalne strategije igranja neke društvene igre. Možemo znati samo koliko je neka strategija dobra u usporedbi s nekom drugom.)

Turnirska selekcija ne zahtijeva znanje o cijeloj populaciji, dovoljno je imati relaciju koja može usporediti bilo koje 2 jedinke. Pseudokod turnirske selekcije koja odabire μ roditelja dan je sa algoritmom 2.

Algorithm 2: Pseudokod za turnirsku selekciju

```
trenutni_član=1
while trenutni_član ≤ μ do
    odaberi nasumičnih k jedinki
    odaberi najboljeg među njima s obzirom na njihovu podobnost, označi ga s i
    roditelji[trenutni_član]=i
    trenutni_član=trenutni_član+1
end
```

Budući da turnirska selekcija radi s relativnom, a ne apsolutnom vrijednosti podobnosti, ima ista svojstva invarijantnosti na translaciju funkcije podobnosti kao i selekcija na bazi ranga. Vjerojatnost odabira jedinke kao pobjednika turnira ovisi o 4 faktora:

- Rang jedinke u populaciji.

- Veličina turnira. Što je turnir veći, veća je vjerojatnost da će sadržavati jedinke natprosječne podobnosti.

- Vjerojatnost p da u turniru uvijek pobjedi jedinka s najvećom podobnosti. Ako $p = 1$ turnir je deterministički, inače je stohastički.

- Dozvoljavamo li duplikate u turniru. Ukoliko ne, primijetimo da kod determinističkih turnira nikad ne možemo odabrati $n - 1$ jedinki s najmanjom podobnosti, gdje je n broj jedinki koje biramo u turniru.

2.5 Odabir jedinki koje preživljavaju smjenu generacija

Ovaj mehanizam odgovoran je za proces odabira μ jedinki iz skupa od μ roditelja i λ djece koji će preći u sljedeću generaciju. Glavne kategorije su mehanizmi koji biraju na osnovu podobnosti i oni koji biraju na osnovu dobi.

2.5.1 Izbor na osnovu dobi

Ovaj tip izbora ne uzima u obzir podobnost jedinki, već je dizajniran na način da svaka jedinka preživi u populaciji jednak broj generacija. Obično se koristi kod jednostavnih GA. Ukoliko je $\lambda = \mu$ svaka jedinka preživi točno jedan ciklus te se zamijeni s nekim potomkom. Naravno, implementacija te metode moguća je i za $\lambda < \mu$, npr. za $\lambda = 1$.

2.5.2 Izbor na osnovu podobnosti

Ranije smo već opisali turnirsku selekciju i selekciju proporcionalnu podobnosti, pa ćemo samo napomenuti da se one mogu koristiti i u ovom kontekstu, kao i stohastička verzija izbora na osnovu ranga. Detaljnije ćemo opisati još 2 uobičajena mehanizma.

Zamjena najgoreg (GENITOR)

Najgorih λ jedinki odaberemo i zamijenimo. Iako ovakav način zamjene može dovesti do jako brzog napretka podobnosti cijele populacije, riskiramo prebrzu konvergenciju prema jedinki s najvećom podobnosti. Zbog toga se često koristi kod velikih populacija te se ne dozvoljava dupliciranje jedinki.

Elitizam

Cilj kod primjene elitizma je spriječiti gubitak najbolje jedinke u populaciji. To se postiže tako da se u svakom trenutku prati trenutačno najkvalitetniji član te se njega uvijek ostavlja u populaciji. Konkretno, ako je on jedna od jedinki označenih za zamjenu, a među potomstvom nema jedinki s boljom kvalitetom, umjesto jednog od potomaka u populaciji ostavljamo tu, najkvalitetniju jedinku.

Poglavlje 3

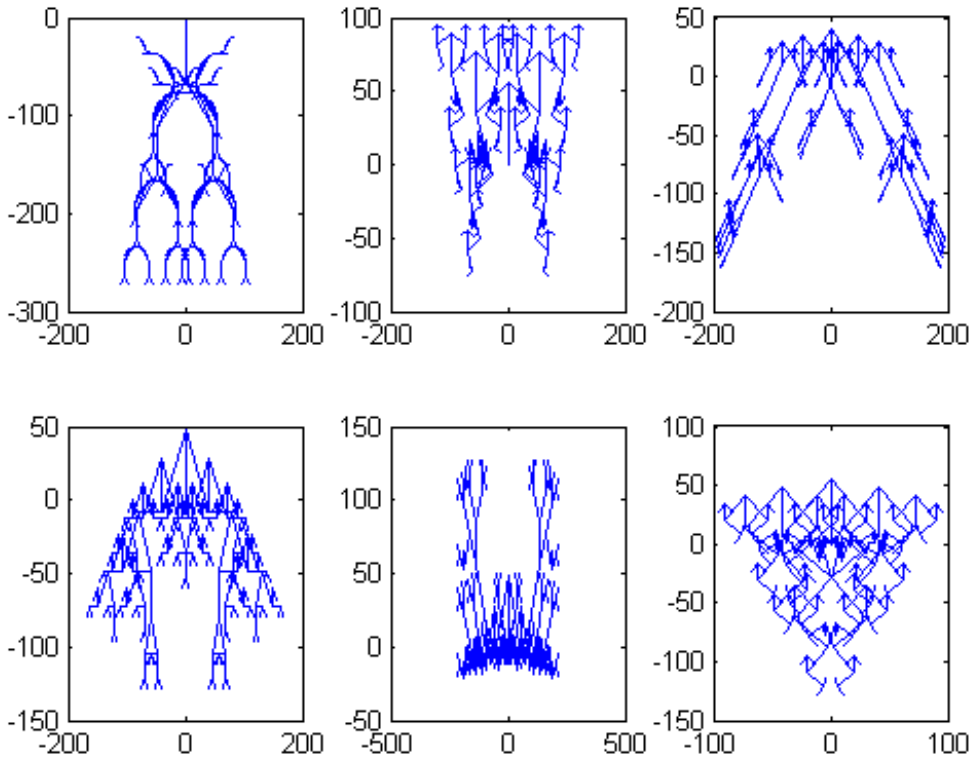
Eksperimenti s biomorfima

Nakon pregleda svih ključnih komponenti genetskih algoritama, u ovom ćemo poglavlju na primjeru biomorfa pokušati detaljnije proučiti njihov rad. Biomorfe [1] je u računarstvo prvi uveo Richard Dawkins u knjizi "The blind watchmaker" [2], kako bi pokazao da se gomilanjem malih promjena na genima kroz dovoljno generacija mogu razviti bitno drugačiji organizmi od početnog. U našim eksperimentima biomorfi su "bića" koja nastaju uzastopnim grananjem linija na način koji je određen genima svakog od tih bića. Slika 3.1 prikazuje nekoliko tipičnih primjera biomorfa. Eksperimente slične Dawkinsovima pokušat ćemo ponoviti i u ovom radu kako bi demonstrirali efikasnost genetskih algoritama, uz bitnu razliku da ćemo podobnost organizama vrednovati nekom funkcijom podobnosti, dok je Dawkins u originalnom radu ručno birao organizme koji će biti roditelji sljedeće generacije na temelju njihovog izgleda. Konkretno, pokrenut ćemo neki genetski algoritam na različitim problemima s različitim parametrima te usporediti rezultate.

3.1 Osnovni podaci o biomorfima

Pozabavimo se detaljnije strukturom biomorfa. Biomorfi koje ćemo koristiti u našim pokusima imat će 9 gena. Prvih 8 reprezentirat ćemo cijelim brojevima u rasponu od -7 do 8, dok ćemo deveti gen reprezentirati prirodnim brojevima od 1 do 8. Taj, deveti gen reprezentira broj "nivoa" od kojih će se sastojati određeni biomorf, dok prvih 8 utječe na duljinu i nagib njegovih grana. Konkretno, predznak prvih 8 gena govorit će nam u kojem smjeru vodi određena grana, dok će njihov iznos utjecati na duljinu. Slika 3.2 prikazuje jedan, relativno jednostavan biomorf u sredini (genotipa $[1, 1, 1, 1, 1, 1, 1, 1, 4]$), dok su oko njega neki biomorfi nastali mutacijom samo jednog od njegovih gena za +1 ili -1. Svaka grana biomorfa se, kad dođe do kraja, razdvaja na točno dvije nove grane. Duljina grane, osim što ovisi o genima, linearno se smanjuje s brojem grananja od korijena do te grane. Ukoliko se

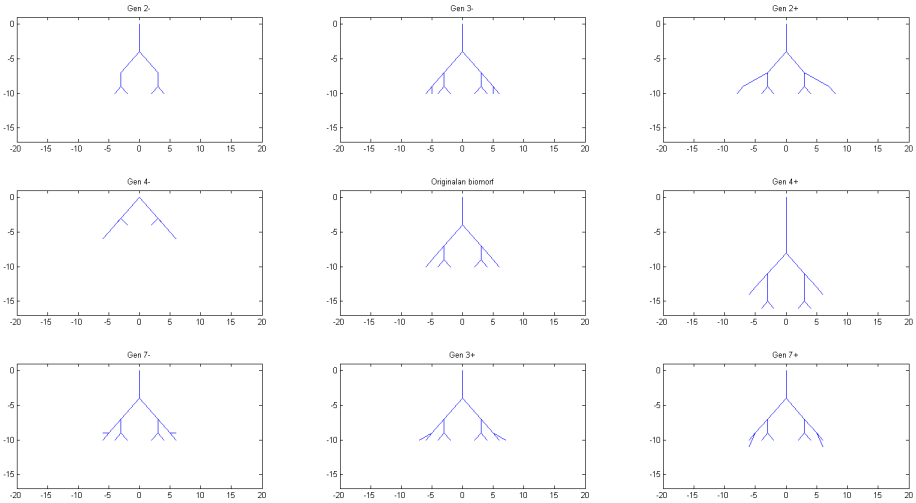
[1] Biomorf u širem smislu riječi je bilo koji objekt koji nalikuje na živo biće.



Slika 3.1: Neki tipični biomorfi. Primijetimo i razliku na skali veličine između njih.

neka od tih grana ne vidi na crtežu, njezina duljina je 0, no još uvijek se iz nje mogu granati nove grane. To napominjemo zbog toga jer se može dogoditi da iz jedne točke (na kraju grane x) izlazi jedna ili više od 2 grane. Ukoliko izlaze više od dvije grane, to znači da se grana x razdvojila na 2 grane od kojih je jedna duljine 0 te se ta grana naknadno razdvojila na još dvije grane.

Objasnimo sada malo pobliže proces grananja biomorfa. U trenutku kada grana dođe do svojeg kraja, geni određuju duljinu i kut grananja grana nastalih iz nje. Primjetimo da je prva grana biomorfa usmjerena ravno prema dolje (kada gen 4 smanjimo za 1 u ovom konkretnom slučaju on će biti 0, duljina te grane je 0, pa se zato čini da ne postoji ta prva, prema dolje usmjerena grana), dakle raste "u smjeru juga". Grane nastale iz nje tada rastu u smjeru "jugoistoka" i "jugozapada". Zapravo, ako zamislimo ružu vjetrova s 8 krakova, grananje iz neke grane uvijek će ići "u smjeru" 2 susjedna kraka. Navodnike smo ovdje stavili jer ti smjerovi nisu pravilno raspoređeni kao na ruži vjetrova, bitno je samo da ih



Slika 3.2: Prikaz biomorfa i njemu bliskih biomorfa.

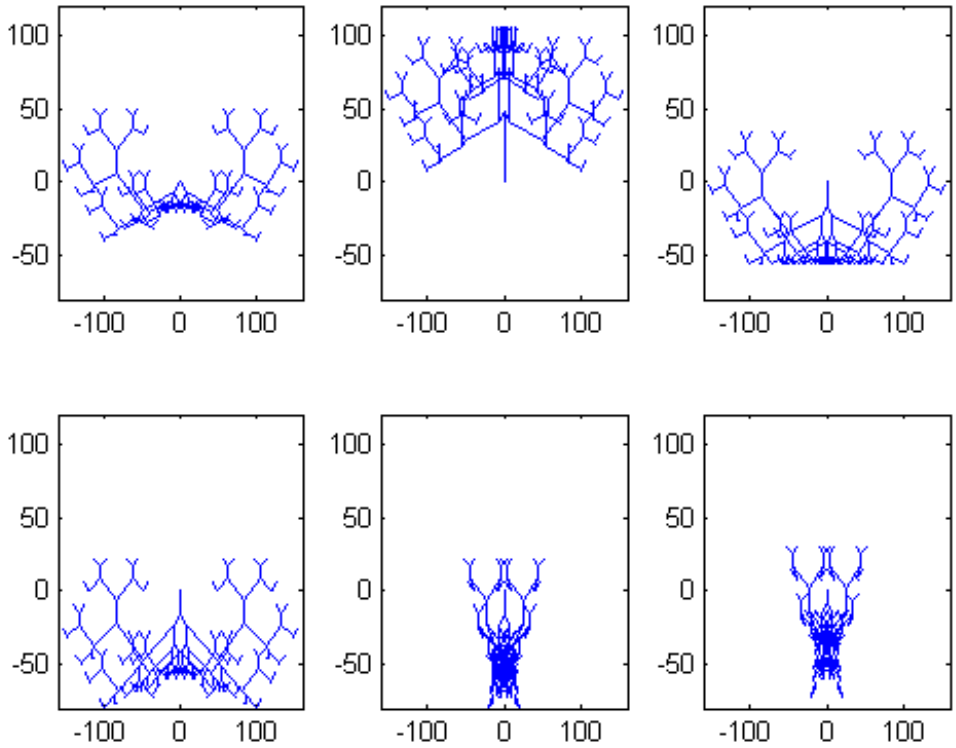
ima 8. Kad bi za svaki od tih smjerova jedan gen na neki način određivao pomak u smjeru x, a jedan u smjeru y, očito bi trebali 16 različitih gena. No, primijetimo da su svi biomorfi simetrični s obzirom na y. To nam omogućuje redukciju broja gena na 8. U tablici 3.1, za svaki smjer ruže vjetrova dan je indeks gena koji na njega utječe u smjeru x i y osi. Smjer 3 je ravno dolje, 7 je ravno gore (zbog toga jer je dx u tim smjerovima jednak 0), dok su ostali smjerovi raspoređeni u smjeru kazaljke na satu.

dx(1)	=	-dx(5)	dy(1)	=	dy(5)
dx(2)	=	-dx(4)	dy(2)	=	dy(4)
dx(3)	=	0	dy(3)	=	gen(4)
dx(4)	=	gen(1)	dy(4)	=	gen(5)
dx(5)	=	gen(2)	dy(5)	=	gen(6)
dx(6)	=	gen(3)	dy(6)	=	gen(7)
dx(7)	=	0	dy(7)	=	gen(8)
dx(8)	=	-dx(6)	dy(8)	=	dy(6)

Tablica 3.1: Veza gena i rasta biomorfa.

Iako na prvi pogled i nije jasno koliko veliku varijabilnost fenotipa biomorfa možemo dobiti na ovaj način, na slici 3.3 dano je 6 biomorfa koji su dobiveni jedan od drugog mutacijom samo jednog gena, dakle drugi biomorf dobiven je mutacijom jednog gena iz

prvog itd. (mutacija je ovdje implementirana kao nasumično biranje bilo kojeg $n \in [-7, 8]$). Razlike u fenotipu su drastične, iako je promijenjen samo jedan gen.



Slika 3.3: Ilustracija promjena na biomorfu nastalih mutacijom jednog gena.

Nakon što smo ukratko opisali biomorfe, pozabavimo se nekim konkretnim eksperimentima.

3.2 Eksperimenti s biomorfima

Za početak, kao najjednostavniji primjer, zamislimo biomorf kao organizam koji je bolje adaptiran na okolinu što je njegova ukupna duljina svih grana veća. Dakle, funkcija podobnosti bit će jednostavno zbroj svih duljina njegovih grana. Također, podsjetimo se da su geni naših biomorfa u rasponu od $[-7, 8]$ te da je duljina svake grane na neki način proporcionalna apsolutnoj vrijednosti nekih od gena. To možemo iskoristiti kako bi vidjeli

je li pronađeno globalno najbolje rješenje (jedinka s genotipom $[8,8,8,8,8,8,8,8,8]$), ili smo zapeli u nekom lokalnom optimumu (pojavljuje se jedan ili više alela -7 u genotipu rješenja). U sljedećoj tablici 3.2 dajemo pregled svih elemenata GA koji će biti isti za sve eksperimente, dok ćemo njihove parametre dati kasnije.

Populacija	500 jedinki, nasumično inicijaliziranih
Mehanizam odabira roditelja	Turnirska selekcija (deterministička, biramo λ roditelja)
Mutacija	Biranje nasumičnog broja iz $[-7,8]$ na točno jednom mjestu u jedinki s vjerojatnošću p
Rekombinacija	Križanje u jednoj točki, broj nivoa uzimamo nasumično iz jedne od jedinki
Odabir preživjelih	Izbacujemo λ najmanje podobnih jedinki
Uvjet zaustavljanja	Maksimalan broj generacija postignut ili nema poboljšanja u 20 generacija

Tablica 3.2: Parametri GA

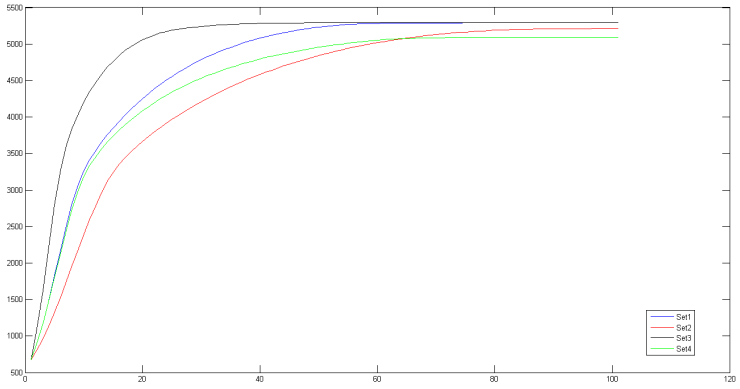
GA ćemo pokrenuti 100 puta sa svakom kombinacijom parametara te ćemo za svaki od njih, u svakoj generaciji pamtiti najbolju i prosječnu podobnost, kao i genotip najbolje jedinke. Dobivene rezultate grafički ćemo prikazati. Parametri koje smo koristili dani su u tablici 3.3.

	Set1	Set2	Set3	Set4
Parametar mutacije (p)	0.05	0.05	0.05	0.25
Broj djece u svakoj generaciji (λ)	100	60	100	100
Veličina turnira	2	2	6	2

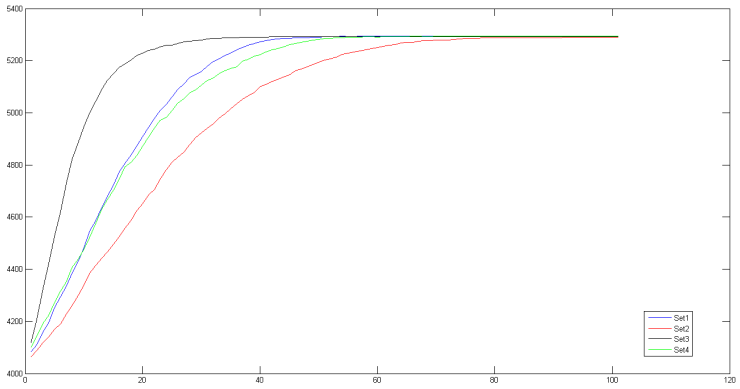
Tablica 3.3: Parametri GA

Na grafu 3.4 prikazan je rast prosječne podobnosti (točnije prosjeka prosječne podobnosti za svih 100 pokrenutih algoritama s tim setom parametara) za svaki set parametara, dok je na grafu 3.5 prikazan rast maksimalne podobnosti (također, zapravo se radi o prosječnoj maksimalnoj podobnosti) u svakoj generaciji. Primijetimo da se grafovi dobro poklapaju s grafom 1.3, dakle podobnost brzo napreduje u početku, a kasnije usporava.

Također valja spomenuti u kolikom broju pokretanja algoritama, od 100 provedenih algoritama u svakom setu, nije nađeno, ili je nađeno suboptimalno rješenje te koliko je vremena u prosjeku trebalo algoritmu da nađe rješenje. Ukoliko nije nađeno, to znači da rješenje nije nađeno do 80-e iteracije, budući da je kriterij zaustavljanja 20 iteracija bez poboljšanja. Zapravo, to se dogodilo u sva 3 slučaja koji su navedeni kao "nije nađeno rješenje", štoviše, sva su 3 našla optimalno rješenje, samo prekasno da bi se aktivirao uvjet zaustavljanja. Podaci su dani u tablici 3.4. Primijetimo da je u velikoj većini slučajeva



Slika 3.4: Rast prosječne podobnosti.



Slika 3.5: Rast maksimalne podobnosti.

pronađeno upravo optimalno rješenje. Razlog je iznimno jednostavna funkcija podobnosti. Kasnije, u eksperimentima sa složenijom funkcijom podobnosti, postotak u kojem je algoritam uspio pronaći optimalno rješenje bit će znatno niži.

Analizirajmo rezultate u nekoliko rečenica. Primijetimo da prosječna i maksimalna podobnost najbrže rastu kad je povećana veličina turnira. To je očekivano, naime povećanjem veličine turnira (pogotovo deterministčkog) dolazi do jačeg selekcijskog pritiska, što ubrzava porast podobnosti. No, usporedimo li broj pronađenih suboptimalnih rješenja nađenih s veličinom turnira 6 s brojem suboptimalnih rješenja nađenih s veličinom turnira 2, vidimo

	Set1	Set2	Set3	Set4
Nije nađeno rješenje	0	3	0	0
Suboptimalna rješenja	2	8	4	3
Prosječno iteracija do nalaženja rješenja	60.41	80.56	45.92	65.56

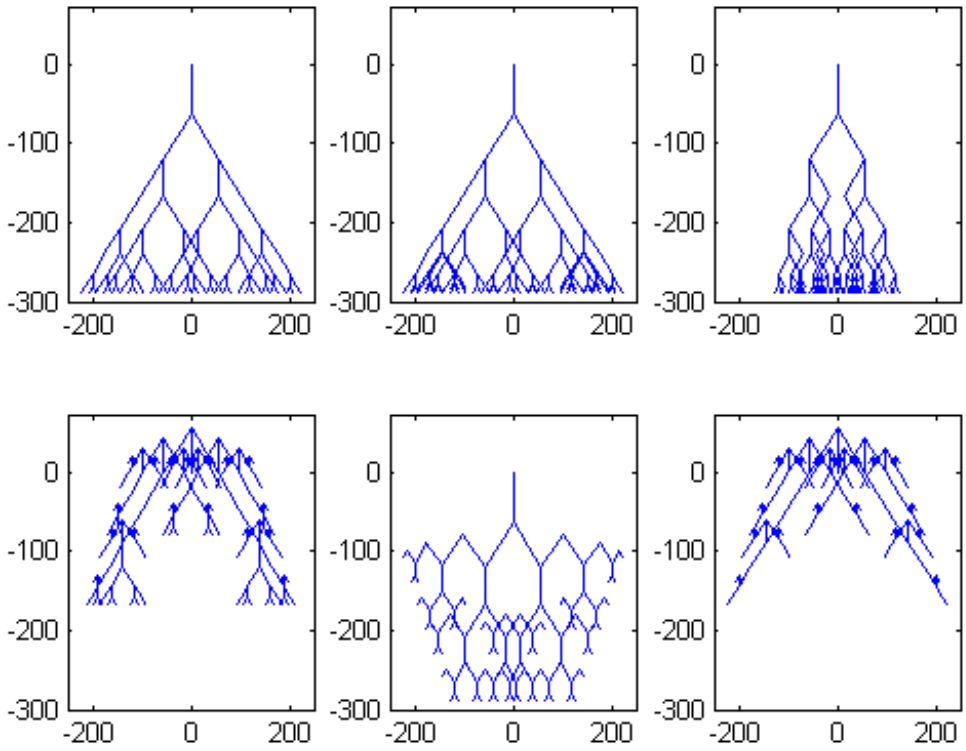
Tablica 3.4: Podaci o izvedbi algoritma

da povećanjem broja jedinki u turniru također povećavamo rizik zapinjanja u lokalnom optimumu kao rezultat brzog približavanja trenutno najpodobnijoj jedinki. Nadalje, usporedimo li krivulje dobivene iz seta 1 s krivuljama iz seta 2, vidjet ćemo da smanjenje broja djece u svakoj generaciji usporava porast podobnosti populacije. To je i očekivano, budući da smo time smanjili priljev novih, podobnijih jedinki, a također i objašnjava zašto je to jedini skup podataka u kojem se dogodilo da u 100 iteracija nije postignut uvjet zaustavljanja. Krivulje iz seta 4 najsličnije su onima iz seta 1, a i broj suboptimalnih rješenja je blizak, dakle parametar mutacije najmanje utječe na kvalitetu algoritma u ovom slučaju. No, algoritam funkcionira relativno dobro uz sva 4 seta parametara, što možemo pripisati relativno jednostavnoj funkciji podobnosti. Također, uz tako male razlike u performansama algoritma, postavlja se pitanje možemo li uopće izvući ikakve smislene zaključke. Zbog toga, pokušat ćemo isti algoritam primijeniti na neke složenije probleme kako bi istaknuli razlike u performansama u ovisnosti o odabiru parametara. Na slici 3.6 prikazano je optimalno rješenje (gore lijevo) kao i neka suboptimalna rješenja za gornji problem. Primijetimo izrazito velike razlike u fenotipu, iako se vrijednost funkcije podobnosti razlikuje za manje od 3%.

3.3 Složeniji primjeri

Pokušajmo sada isti algoritam primijeniti na neke složenije probleme. Zamislimo da konstruiramo takav biomorf kojem podobnost raste sa svakim slobodnim krajem grane unutar nekog prostora, no za konstrukciju njegovih grana potrebni su resursi, dakle što je suma duljina grana dulja, to je veći negativni efekt na podobnost jedinke (ovdje smo cijenu definirali kao duljinu grane). Možemo zamisliti da se radi o korijenu drva koje raste na tlu u kojem su nejednoliko raspoređene hranjive tvari. Točnije, količina hranjivih tvari linearno raste porastom dubine, do $y = -200$ nakon čega počne linearno padati. Primijenili smo isti genetski algoritam kao u prošlom problemu, koristeći sva 4 seta parametara. Također smo algoritam pokrenuli 100 puta te usporedili rješenja. Za razliku od prošlog primjera, maksimalan broj iteracija stavili smo na 200, budući da je funkcija podobnosti kompliciranija pa očekujemo sporije pronalaženje rješenja.

Tablica 3.5 prikazuje broj pokretanja kod kojih rješenje nije pronađeno te broj dobivenih rješenja koja su različita od najboljeg dobivenog rješenja, kao i odstupanja od najboljeg



Slika 3.6: Optimalno i još neka rješenja problema.

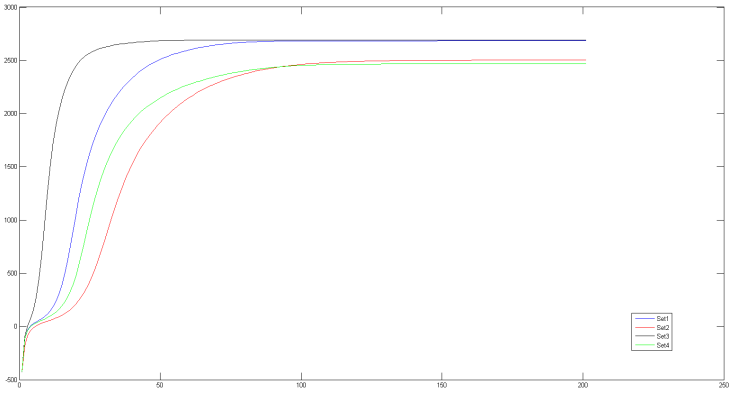
rješenja.

	Set1	Set2	Set3	Set4
Nije nađeno rješenje	0	0	0	0
Suboptimalna rješenja	29	49	45	46
Prosječno iteracija do nalaženja rješenja	85.23	108.09	56.90	95.08
Najveće odstupanje od najboljeg rješenja	17.89%	34.5%	4.56%	26.73%
Prosječno odstupanje od najboljeg rješenja	0.7%	2.21%	1.00%	1.34%

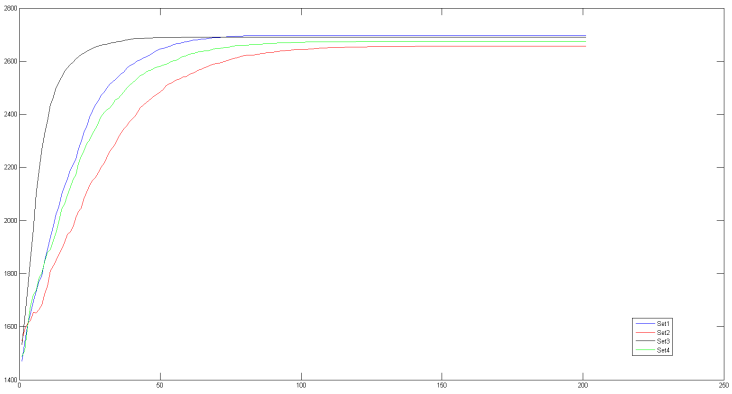
Tablica 3.5: Podaci o izvedbi algoritma

Primijetimo da je nalaženje rješenja zaista bilo sporije, kao i da su se opet najboljima pokazali parametri iz seta 1. Odnos ostalih pokazatelja također prilično dobro prati rezultate iz prošlog primjera; opet možemo primijetiti najbrže nalaženje rješenja kod parametara

iz seta 3 zbog najvećeg selekcijskog pritiska, kao i najsporije uz korištenje parametara iz seta 2 zbog najmanjeg broja djece u svakoj generaciji. To se također vidi i na grafovima koji prikazuju podobnost najbolje (graf 3.8) i prosječne (graf 3.7) jedinke kroz generacije. Zanimljivo je uočiti da odabir prevelikog parametra mutacije negativno utječe na kvalitetu rješenja.



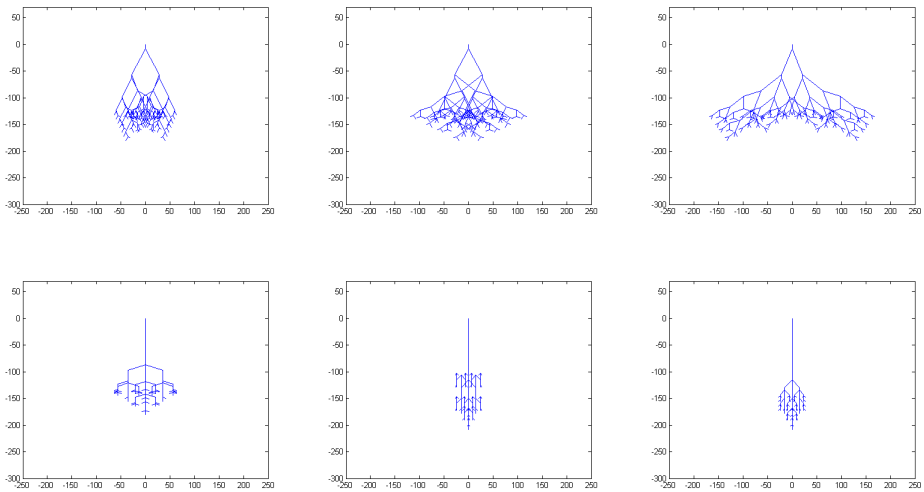
Slika 3.7: Rast prosječne podobnosti.



Slika 3.8: Rast maksimalne podobnosti.

No, crtanjem najbolje jedinke ispostavilo se da je algoritam kao optimalno rješenje pronašao okomitu crtu prema dolje (točnije, više crta koje sve putuju ravno prema dolje;

svaki pomak po osi x je nepoželjan jer ne nosi nikakvu nagradu, a negativno utječe na podobnost). Slika 3.9 prikazuje nekoliko biomorfa koji su u jednom pokretanju algoritma bili najbolje rješenje u svojoj generaciji. Promatramo li ih po redu (od gore lijevo prema dolje desno) možemo jasno vidjeti kako se biomorf stanjuje i približava ravnoj liniji (optimalnom rješenju). Ponovimo sada pokus tako da dodatno nagradimo jedinke koje će imati veći raspon "korijena".



Slika 3.9: Neki biomorfi dobiveni u jednom pokretanju algoritma.

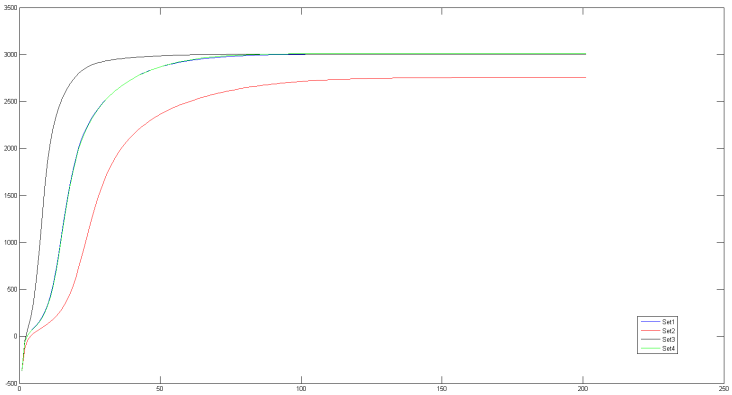
Koristit ćemo istu funkciju podobnosti kao i u prethodnom primjeru, no dodat ćemo još jedan pribrojnik koji je proporcionalan najvećoj horizontalnoj udaljenosti 2 slobodna kraja grana. Time se nadamo dobiti biomorfe koji se neće grupirati oko samo jedne linije, a ujedno ćemo dodatno zakomplicirati funkciju podobnosti te promotriti kako se naš algoritam nosi s tim. Opet smo algoritam pokrenuli 100 puta sa svakim setom podataka, s maksimalno 200 iteracija. Tablica 3.6 prikazuje broj pokretanja kod kojih rješenje nije uspješno pronađeno, broj dobivenih rješenja koja su različita od najboljeg dobivenog rješenja te odstupanja od najboljeg rješenja.

Možemo primijetiti da se prosječno odstupanje od najboljeg rješenja smanjilo, iako se broj suboptimalnih rješenja povećao. Posebno vrijedi istaknuti porast u kvaliteti performansi seta parametara s većim parametrom mutacije. To možemo pripisati strukturi prostora rješenja. U prošlom primjeru velik parametar mutacije onemogućavao je brzo penjanje prema optimalnom rješenju što je usporavalo algoritam i često nas izbacivalo s puta prema tom optimumu; ovdje se to pokazalo korisnim zbog kompleksnije strukture pros-

	Set1	Set2	Set3	Set4
Nije nađeno rješenje	0	0	0	0
Suboptimalna rješenja	36	56	58	36
Prosječno iteracija do nalaženja rješenja	90.26	108.81	65.64	89.94
Najveće odstupanje od najboljeg rješenja	2.9%	20.66%	4.98%	3.22%
Prosječno odstupanje od najboljeg rješenja	0.44%	2.04%	0.80%	0.48%

Tablica 3.6: Podaci o izvedbi algoritma

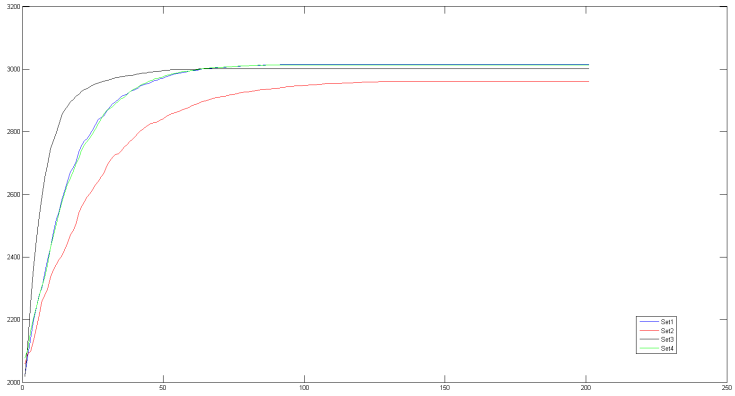
tora (većeg broja bliskih po podobnosti lokalnih optimuma) te omogućilo skakanje između njih. Međusobni odnos rezultata dobivenih različitim parametrima ostao je vrlo sličan kao i u prethodnim primjerima. Ponovno prilažemo grafove koji prikazuju podobnost najbolje (graf 3.11) i prosječne (graf 3.10) jedinke kroz generacije. Možemo primijetiti da su se performanse algoritma s parametrima iz seta 4 skoro izjednačile s onima iz prvog seta, iako su u jednostavnijim primjerima bile dosta lošije. Set 2 i dalje je najlošiji, dok set 3 daje najbrže nalaženje rješenja, no i nešto lošiju kvalitetu rezultata.



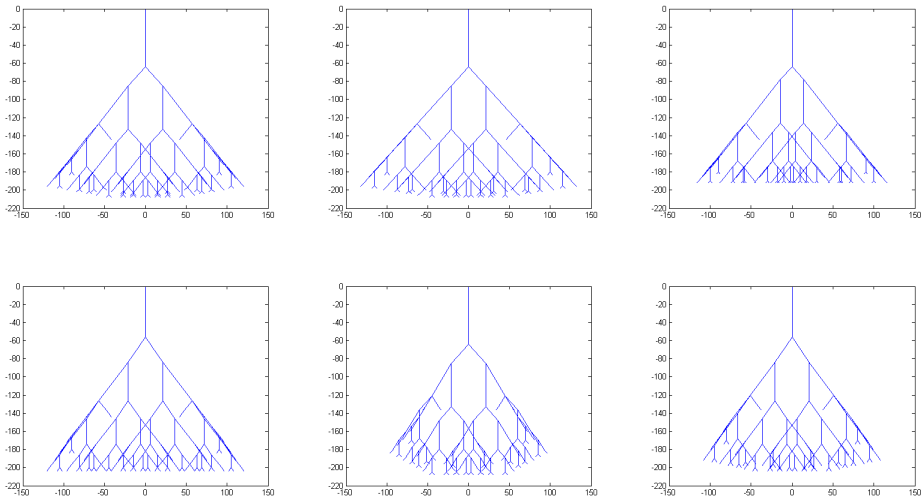
Slika 3.10: Rast prosječne podobnosti.

Također, zanimljivo je pogledati koliko su različiti biomorfi koje je algoritam vratio kao rješenja. Na slici 3.12 prikazan je biomorf koji je optimalno rješenje problema (gore lijevo), kao i 5 najboljih rješenja koja nisu optimalna. Ostala rješenja poredana su po podobnosti (dolje desno je najlošije rješenje).

Kao zadnji primjer sa simetričnim biomorfima promotrimo prostor u kojem su biomorfi kažnjeni ukoliko im grane imaju vrhove u određenim dijelovima ravnine, a nagrađeni ukoliko imaju vrhove u nekim drugim dijelovima ravnine. To znači da postoji velika vjero-



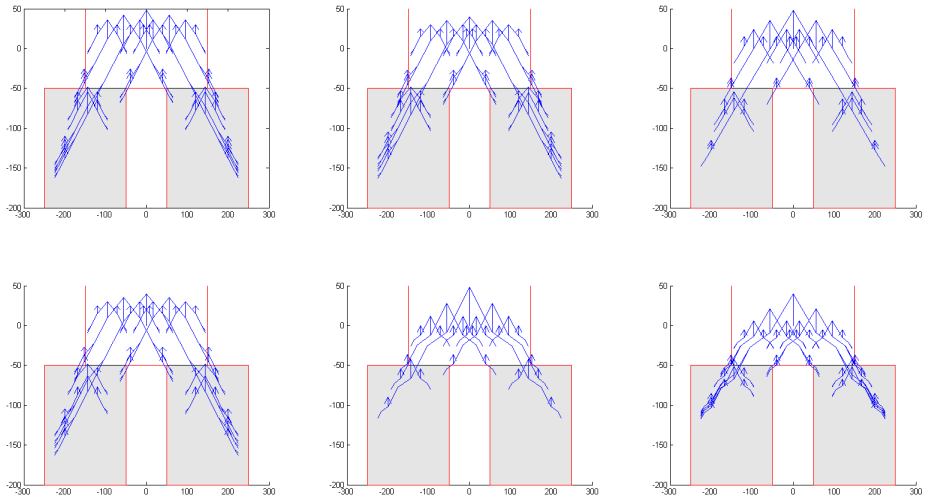
Slika 3.11: Rast maksimalne podobnosti.



Slika 3.12: Fenotip biomorfa koje je algoritam vratio kao rješenje problema.

jatnost da algoritam zapne u lokalnom optimumu (situacija u kojoj nijedan biomorf nema nijednu granu ni u području koje se nagrađuje, ni u području koje se kažnjava). Neka rješenja (gore lijevo je optimalno rješenje), kao i dijelovi za koje biomorfi dobivaju kaznu, odnosno nagradu prikazani su na slici 3.13.

U tablici 3.7 dani su podaci o izvršavanju algoritma. Vidimo velik pad u kvaliteti



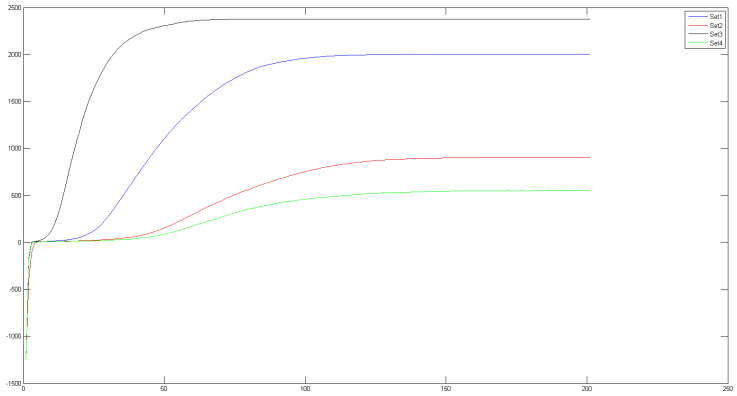
Slika 3.13: Fenotip biomorfa koje je algoritam vratio kao rješenje problema. Crvena crta je granica koju biomorfi ne mogu prijeći bez kazne. Sivo osjenčano područje je područje u kojem dobivaju nagradu.

rješenja, tj. izrazito veliko najveće i prosječno odstupanje od najboljeg rješenja. Ipak, algoritam je sa sva 4 seta podataka pronašao identično najbolje rješenje. Pregledom rješenja s jako velikim odstupanjem od najboljeg rješenja ispostavilo se da to zaista jesu biomorfi koji nisu uspjeli doći do područja s nagradom (ostali su u neosjenčanom području oko ishodišta, ali i unutar crvene linije). Ovo je dobar primjer problema u kojem bi dobra inicijalizacija mogla bitno poboljšati algoritam. Naime, ukoliko na početku u populaciju ubacimo par jedinki koje imaju neke grane u sivom području, a da ne prelaze crvenu liniju, možemo smanjiti vjerojatnost zapinjanja u lokalnom optimumu. Primijetimo pozitivan učinak koji ima povećanje broja jedinki u turniru, kao i negativan utjecaj koji ima smanjenje broja djece. Također, povećanje parametra mutacije bitno je pogoršalo prosječnu kvalitetu rješenja.

Sljedeća 2 grafa prikazuju tok prosječne (3.14) i maksimalne podobnosti (3.15) kroz generacije. Jako spor rast podobnosti za set s velikim parametrom mutacije možemo objasniti time što mutacija samo jednog gena može velik dio biomorfa izbaciti iz prostora u kojem dobiva nagradu u prostor u kojem biva kažnjen, pa je teško očekivati stabilan rast uz čestu mutaciju. To također objašnjava i jako nepravilnu krivulju rasta maksimalne podobnosti. Također se jasno vidi pozitivan efekt većeg turnira, kao i negativan efekt manjeg broja djece.

	Set1	Set2	Set3	Set4
Nije nađeno rješenje	0	0	0	0
Suboptimalna rješenja	87	94	88	98
Prosječno iteracija do nalaženja rješenja	91.95	68.27	64.49	65.23
Najveće odstupanje od najboljeg rješenja	89.98%	79.97%	32.33%	71.24%
Prosječno odstupanje od najboljeg rješenja	13.59%	37.21%	7.35%	36.16%

Tablica 3.7: Podaci o izvedbi algoritma.

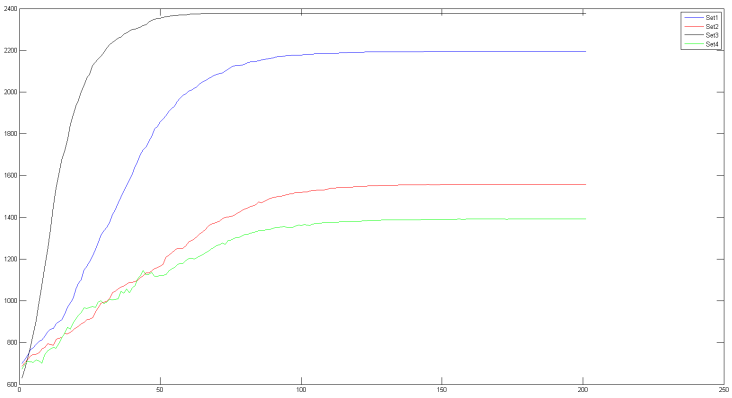


Slika 3.14: Rast prosječne podobnosti.

3.3.1 Primjer s asimetričnim biomorfima

U ovom primjeru koristit ćemo sličnu funkciju podobnosti kao i u prošlim, no na asimetričnim biomorfima. Konkretno, biomorf je to podobniji što ima više slobodnih krajeva što je bliže moguće pravcu $y = -100$ (nagrada linearno ovisi o udaljenosti), a za konstrukciju grana potrebni su resursi. Također smo biomorf dodatno nagradili u ovisnosti o duljini x komponente grane sa slobodnim krajem. To smo učinili ne bi li tako izbjegli grupiranje svih linija na jednom pravcu. Dakle, umjesto 8 gena koji određuju smjer i kut među granama, koristit ćemo 16 gena, dok će sedamnaesti gen određivati broj nivoa u biomorfu. I dalje ćemo koristiti 4 seta parametara i ograničenje na 200 iteracija, kao i u prethodnim primjerima te ćemo algoritam sa svakim setom parametara pokrenuti 100 puta. Detaljni podaci o izvedbi algoritma nalaze se u tablici 3.8. Neki primjeri asimetričnih biomorfa nalaze se na slici 3.16

Primijetimo velik broj pokretanja algoritama koji nisu uspjeli naći rješenje do dvjestote iteracije, kao i da je ovaj problem prvi u kojem najbolje pronađeno rješenje nije bilo

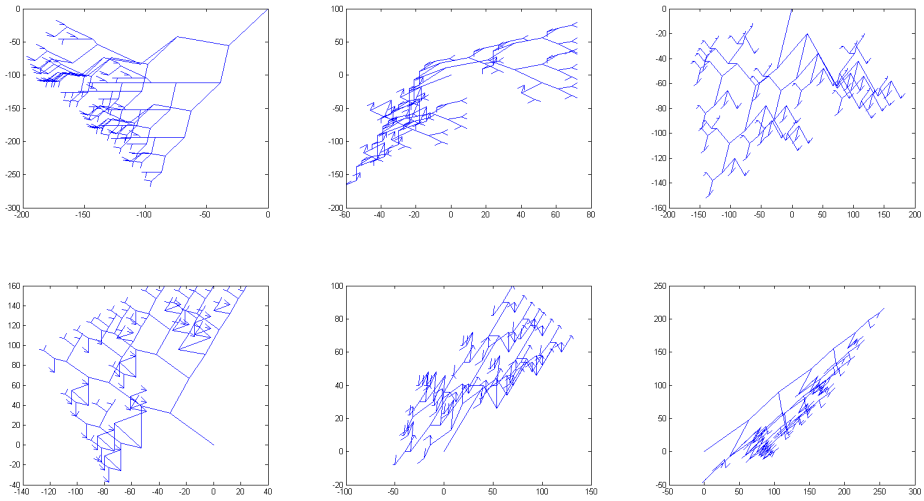


Slika 3.15: Rast maksimalne podobnosti.

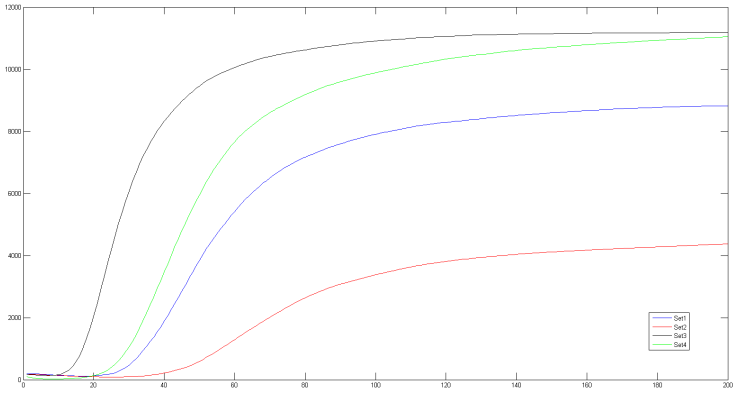
	Set1	Set2	Set3	Set4
Nije nađeno rješenje	72	40	40	96
Suboptimalna rješenja	88	99	65	99
Prosječno iteracija do nalaženja rješenja	164.30	175.33	175.10	195.08
Najveće odstupanje od najboljeg rješenja	74.83%	74.20%	10.71%	9.27%
Prosječno odstupanje od najboljeg rješenja	18.09%	43.00%	4.22%	2.42%
Podobnost najboljeg rješenja	11672	11482	11672	11669

Tablica 3.8: Podaci o izvedbi algoritma na asimetričnim biomorfima.

identično s bilo kojim setom parametara, iako je ta razlika relativno mala. To možemo pripisati udvostručenju broja gena, što je bitno povećalo kompleksnost problema. Ako promatramo samo rješenja pronađena prije dvjestote iteracije, parametri iz seta 3 daju najbolje rezultate. No, promotrimo li pažljivije rezultate dobivene s parametrima iz seta 4, primijetit ćemo da u trenutku zaustavljanja algoritma, na dvjestotoj iteraciji, oni ipak imaju bolje rezultate. Također, iz 3.18 vidimo da krivulja koja pripada setu 4 najstrmije raste na desnom rubu, dakle vjerojatno će se rješenje još poboljšati. Razlog zbog kojeg se algoritam teže zaustavlja je velik parametar mutacije koji mijenja i najbolja rješenja. Da je u setu 1 parametar mutacije bio veći, velika je vjerojatnost da se algoritam ne bi zaustavio kod rješenja koje je od optimalnog odstupalo čak 75%. Parametri iz seta 2 ponovno su se pokazali najgorima u traženju rješenja (vidi 3.18). Logičan korak bio bi, za manji broj djece, postrožiti uvjet zaustavljanja (povećati broj iteracija u kojima zahtijevamo da algoritam nije uspio naći bolje rješenje od do tada najboljeg kako bi ga zaustavili), kako bi se omogućilo više vremena za priljev novih jedinki.

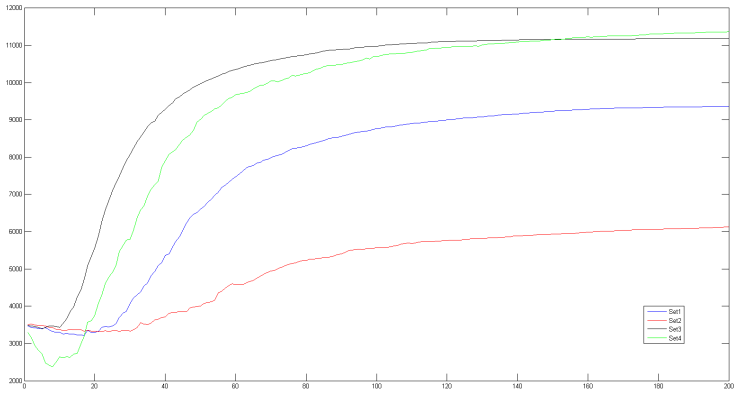


Slika 3.16: Neki asimetrični biomorfi.

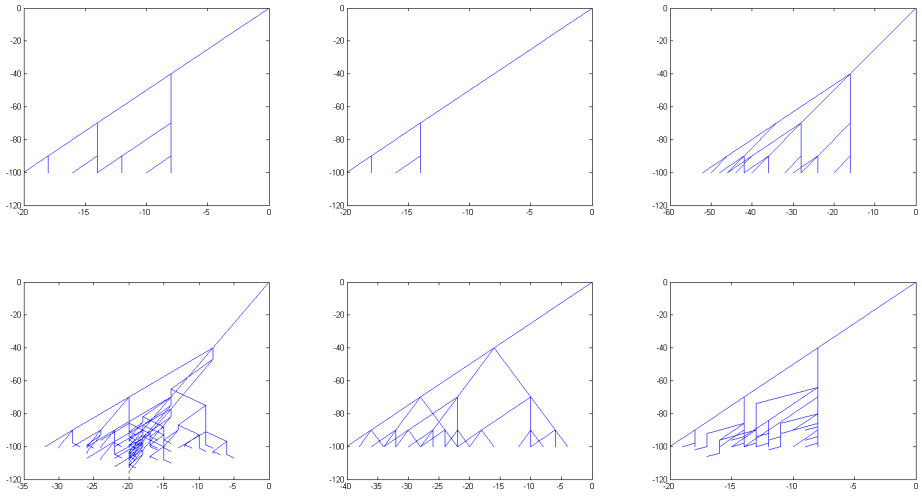


Slika 3.17: Rast prosječne podobnosti. Parametri iz seta 3 omogućuju jako brz porast prosječne podobnosti, ali ih prosječna podobnost dobivena parametrima iz seta 4 sustiže kako se približavamo kraju.

Na kraju još prilažemo sliku na kojoj su prikazana neka od rješenja problema. 3.19



Slika 3.18: Rast maksimalne podobnosti.



Slika 3.19: Neka rješenja problema.

3.3.2 Zaključak

Kao zaključak ovih eksperimenata možemo istaknuti da izbor optimalnih parametara uvelike ovisi o tipu problema koji želimo riješiti. Dok kod najjednostavnijih problema izbor najboljih parametara i nije bio od presudne važnosti jer su se svi setovi parametara dobro

nosili s problemom, kod kompleksnijih problema počele su se primijećivati neke razlike. Od bitnijih stvari možemo istaknuti negativan efekt koji relativno velik parametar mutacije ima kod traženja rješenja na prostoru s jednim globalnim optimumom mnogo boljim od lokalnih optimuma, dok kod prostora s više lokalnih optimuma bliskih po vrijednosti globalnom optimumu povećanje parametra mutacije može imati pozitivan efekt. Također, na svim primjerima potvrdilo se da povećanje broja jedinki u turniru dovodi do mnogo bržeg pronalaska rješenja, no povećava i šansu zapinjanja u lokalnom optimumu. Ipak, najnegativniji je učinak na izvedbu algoritma u svim primjerima imalo preveliko smanjivanje broja djece.

Bibliografija

[1] Y. Davidor, H. P. Schwefel i Männer, *Proceedings of the 3rd International Conference on Parallel Problem Solving from Nature*, Springer, Berlin, Heidelberg, New York, 1994.

[2] R. Dawkins, *The blind watchmaker*, Norton & Company, Inc, New York, NY, 1986.

[3] A.E. Eiben, T. Bäck, M. Schoenauer i H. P. Schwefel, *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, Springer, Berlin, Heidelberg, New York, 1998.

[4] A.E. Eiben, P. E. Raué i Z. Ruttkay, *Genetic Algorithms with multi-parent recombination*, objavljeno u [1], 78–87.

[5] A.E. Eiben i C. A. Schippers, *Multi-parent's niche: n-ary crossovers on NK-landscapes*, objavljeno u [17], 319–328.

[6] A.E. Eiben i J.E. Smith, *Introduction to Evolutionary Computing*, Springer, 2003.

[7] L. J. Fogel, P. J. Angeline i M. J. Walsh, *Artificial intelligence through a simulation of evolution*, Wiley, Chichecter, UK, 1966.

[8] D.F. Goldberg i R. Lingle, *Alleles, loci, and the traveling salesman problem.*, objavljeno u [9], 154–1599.

[9] J.J. Greffenstette, *Proceedings of the 1st International Conference on Genetic Algorithms and Their Applications*, Lawrence Erlbaum, Hillsdale, NJ, 1985.

[10] J. H. Holland, *Genetic alghorithms and the optimal allocation of trials*, SIAM Journal of Computing **2** (1973), 88–105.

[11] J.R. Koza, *Genetic Programming*, MIT Press, Cambridge, MA, 1992.

[12] H. Mühlenbein, *Parallel genetic algorithms, population genetics and combinatorial optimization*, objavljeno u: [14], 416–421.

[13] I. Rechenberg, *Evolutionstrategie: Optimierung Technisher Systeme nach Prinzipien des Biologischen Evolution*, Fromman-Hozlboog Verlag, Stuttgart, 1973.

[14] J.D. Schaffer, *Proceedings of the 3rd International Conference on Genetic Algorithms*, Morgan Kaufmann, San Francisco, 1989.

[15] H P. Schwefel, *Numerische Optimierung von Computer-Modellen mittels der Evolutionstrategie*, ISR **26** (1977).

[16] S. Tsutsui, *Multi-parent recombination in genetic algorithms with search spaceboundary extension by mirroring*, objavljeno u [3], 428–437.

[17] H. M. Voigt, W. Ebeling, I. Rechenberg i H. P. Schwefel, *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, Springer, Berlin, Heidelberg, New York, 1996.

Sažetak

Ovaj rad sastoji se od 2 dijela. U prvom dijelu fokus je na definiranju općih pojmova vezanih uz evolucijske algoritme te nekim važnim zapažanjima vezanim uz njihov rad. Nakon toga, koncentriramo se na genetske algoritme, detaljno opisujemo sve bitne dijelove genetskih algoritama, opisujemo njihovu funkciju, način implementacije te navodimo za koje je probleme svaka od tih implementacija najpogodnija.

Drugi dio ovog rada fokusira se na demonstraciju rada genetskih algoritama pomoću biomorfa. U ovom dijelu primijenit ćemo genetski algoritam na rješavanje raznih problema iz prostora biomorfa. Koristit ćemo razne kombinacije parametara genetskih algoritama te u ovisnosti o kompleksnosti problema proučiti kako oni utječu na izvedbu samog algoritma.

Summary

This work consists of 2 major parts. In the first part, the focus is on defining key concepts of evolutionary algorithms, making some important observations regarding the way they obtain solutions and the way they function in general. Subsequently, genetic algorithms are studied in more detail, all their parts are thoroughly described, as well as their role in the algorithm itself. Also, different implementations suitable for various problems are described.

In the second part, an example genetic algorithm is applied to different problems from the biomorph space, varying in difficulty and complexity. The algorithm is executed with various parameters, and the obtained results are used in order to study how the algorithm solves diverse problems depending on the chosen parameters.

Životopis

Rođen sam u Varaždinu 3.9.1990. godine, gdje sam i pohađao VII. osnovnu školu. Nakon toga upisao sam I. gimnaziju Varaždin, koju sam završio s odličnim uspjehom. Akademske godine 2009/10 upisao sam studij matematike na Prirodoslovno-matematičkom fakultetu. Akademske godine 2011/12 završio sam preddiplomski studij s vrlo dobrim uspjehom, nakon čega sam upisao diplomski studij primijenjene matematike.