SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5896

Primjena samoadaptivnog segregacijskog genetskog algoritma s aspektima simuliranog kaljenja na primjeru učenja robota Robby

Mihael Jakšić
Mentor: prof. dr. sc. Marin Golub

Zagreb, lipanj 2019.

Sadržaj

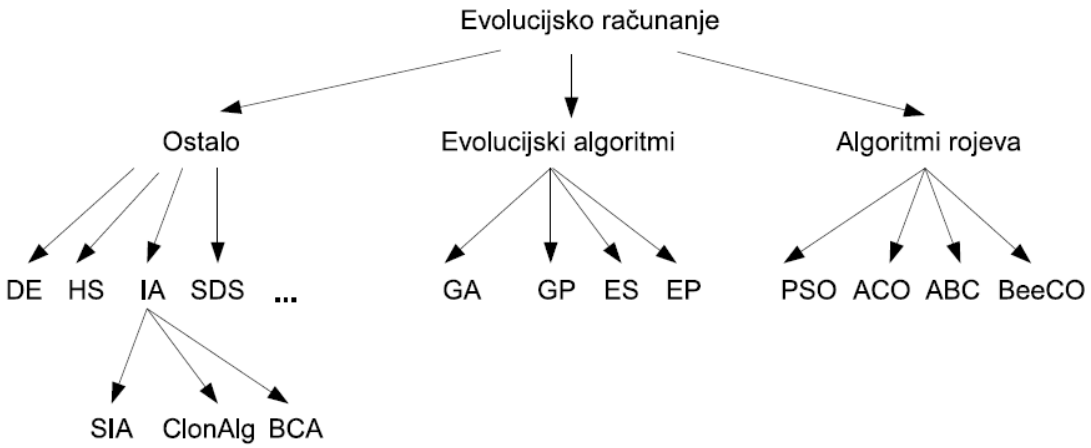
1. Uvod

Svakog dana čovjek se susreće s nizom problema čiji je broj mogućih rješenja izrazito velik. Kod nekih od njih ubrzo i velikom lakoćom uočava se koja su rješenja loša, a koja bi mogla biti blizu onoga što se traži. Tako na primjer, želi li netko tko živi i radi u Zagrebu pronaći najkraći put od objekta stanovanja do svog posla, sigurno zna da najkraći put ne prolazi kroz Dubrovnik te će taj put prirodno i bez previše razmišljanja ukloniti iz bilo kakvog razmatranja.

Međutim, nisu svi problemi s kojima se čovjek može susresti rješivi na tako jednostavan način – postoje i problemi koje čovjek ne može riješiti koristeći intuiciju. Razvojem modernih računala i naglim porastom procesorske snage, ljudi su takve probleme počeli delegirati računalima i rješavati ih tehnikom iscrpne pretrage (u literaturi se još može naći i termin *gruba sila*, engl. *brute force*) tj. pretraživanjem svih mogućih rješenja. Takva pretraga dat će prihvatljiva rješenja samo ako je problem dovoljno jednostavan tj. ako je broj mogućih rješenja takav da se ona sva mogu pretražiti u prihvatljivom vremenu.

Nažalost, većina takvih čovjeku neintuitivnih problema ima toliko mogućih rješenja da ih najbrže računalo današnjice ne može ni sve pobrojati u nekom prihvatljivom vremenu, a kamoli svako od njih vrednovati i izabrati ono najbolje. Tu na scenu stupa grana umjetne inteligencije koja se naziva evolucijsko računanje (negdje se može naći i pojam *evolucijsko računarstvo*, ne miješati s evolucijskim algoritmima koji su dio evolucijskog računanja).

Evolucijsko računanje najčešće se definira kao rod algoritama namijenjen pronalasku globalno najboljeg rješenja nekog problema. Rješavanje problema koristeći evolucijsko računanje počinje sa skupom nasumce generiranih rješenja koji se potom obrađuju i vrednuju. Naziv evolucijsko računarstvo dolazi od toga što skup rješenja evoluira (lat. *evolvo* – razmatati, razvijati) prema onom jednom najboljem rješenju – globalnom optimumu. Na slici 1.1. prikazana je osnovna podjela evolucijskog računanja na glavne grane i u svakoj grani su navedeni odabrani algoritmi.



Slika 1.1. Podjela evolucijskog računanja na glavne grane. U glavnim granama prikazani su odabrani algoritmi. Slika preuzeta iz [1].

Ovaj rad prikazuje primjer uporabe jednog algoritma evolucijskog računarstva zvanog genetski algoritam na problemu učenja robota Robby izloženog u knjizi [2].
Rad je strukturiran kako slijedi:

- 2. poglavlje se bavi genetskim algoritmom (u daljnjem tekstu: GA), njegovom biološkom pozadinom, operatorima odabira roditeljskih rješenja, križanja i mutiranja rješenja djeteta, te su objašnjeni generacijski i eliminacijski genetski algoritam
- u 3. poglavlju je spomenut čest problem GA, problem preuranjene konvergencije te će biti izložene dvije strategije rješavanja problema preuranjene konvergencije raspodjelom jedne populacije na više potpopulacija i razmjenom genetskog materijala između potpopulacija
- u 4. poglavlju se opisuje problem učenja robota Robby, pojašnjava primjena GA i strategija usporavanja preuranjene konvergencije u rješavanju tog problema, izloženi su eksperimentalni rezultati, opisane primjedbe i zaključci te je dan prijedlog ubrzavanja optimizacije paralelizacijom izvođenja
- u zaključku su opisani zaključci izvedeni iz eksperimentalnih rezultata

2. Genetski algoritam

Prvo poglavlje ovog rada posvećeno je genetskom algoritmu. Potpoglavlje 2.1. donosi kratku povijesnu priču o okolnostima nastanka genetskog algoritma. Potpoglavlje 2.2. govori o biološkoj inspiraciji za genetski algoritam. Potpoglavlje 2.3. govori o samom genetskom algoritmu i njegovim uobičajenim implementacijama te uključuje pseudokodove odabranih implementacija. Potpoglavlje 2.4. bavi se tipovima operatora koje genetski algoritam koristi u svom radu i izlaže primjere odabranih operatora.

2.1. Povijesna crtica

Danas se kao jedan od pionira računarstva smatra John von Neumann, američki znanstvenik i polihistor[1] koji je zaslužan za arhitekturu računala u kojoj su naredbe programa i podaci pohranjeni u isti adresni prostor. Ta arhitektura (njemu u čast nazvana von Neumannova arhitektura) predstavljala je prekretnicu u odnosu na dotadašnja računala koja se programiralo koristeći papirnu vrpcu, a također se koristi i danas pri izradi modernih kućnih i mikroračunala.

Kada su von Neumanna 50-ih godina 20. stoljeća upitali mogu li računala reproducirati sama sebe, on je na to pitanje odgovorio potvrdno. Von Neumann je želio poduzeti i naredne dodatne korake i programski simulirati instinkt za preživljavanje, evoluciju i adaptaciju, međutim ubrzo je preminuo i pritom ostavio pitanje neodgovoreno. Nakon njegove smrti, drugi znanstvenici su ubrzo preuzeli ono gdje je on stao, tražeći odgovor na isto pitanje. Godine 1966. prvi konkretni prijedlog takve ideje evoluiranja dali su L. J. Fogel, A. J. Owens i M. J. Walsh koji su svojim radom [3] udarili temelj evolucijskom računanju.

Iz evolucijskog računanja, neovisno jedan od drugoga, razvila su se dva osnovna pristupa koji sadrže evolucijske mehanizme; u SAD-u su se razvijali genetski algoritmi (u daljnjem tekstu: GA), a u Njemačkoj evolucijske strategije (u daljnjem tekstu: ES).

Obje ideje imaju isti osnovni princip; obje rade s populacijom gdje je genetska informacija svake jedinke drugačija. Svaka sadrži niz (polje) u kojem su zapisane sve bitne informacije o jedinci. Na samom početku, populacije se pune jedinkama slučajno generirane genetske informacije, a evolucija, tj. zamjena stare generacije novom generacijom s evoluiranim jedinkama, se odvija sve dok nije ispunjen neki kriterij završetka.

S druge strane, razlika između GA i ES je u načinu predstavljanja genetske informacije. Također, razlika je i u operatorima križanja, mutiranja (o njima će riječi biti

[1] Često se koristi i pojam polimat (grč. πολυμαθής, polymathēs, "onaj koji je mnogo naučio") - definira osobu čija se stručnost proteže na više potpuno različitih područja

kasnije u ovom poglavlju) i rekombinacije (kojom se bavi 3. poglavlje). GA operator mutiranja koristi isključivo kako bi izbjegao stagnaciju, dok ES mutaciju koristi kao primarni operator manipulacije nad genetskom informacijom. Za više informacija o njima, zainteresiranog čitatelja se upućuje na rad [4].
Ostatak ovog rada bavit će se genetskim algoritmom.

Autor genetskog algoritma je američki računarski znanstvenik i psiholog John Holland. Holland je, na neki način, akademski unuk John von Neumanna – naime, znanstvenik koji je bio savjetnik Hollandu pri izradi njegovog doktorskog rada, Arthur Burks, asistirao je upravo von Neumannu u projektiranju računala EDVAC.
Potpoglavlje 2.2. bavi se biološkom inspiracijom za nastanak GA.

2.2. Biološka crtica

Hollanda je k Darwinovoj teoriji evolucije privukla knjiga [5] bazirana upravo na Darwinovoj teoriji evolucije, opisanoj u [6]. Darwin je svoju teoriju temeljio na pet osnovnih ideja:

1. evolucija tj. promjena u lozi, se događa i događala se tijekom vremena,
2. sva živa bića imaju jedno zajedničko podrijetlo,
3. prirodna selekcija određuje promjene u prirodi
4. postepena promjena – priroda se progresivno mijenja
5. razdvajanje vrsta – Darwin je tvrdio da proces prirodne selekcije rezultira time da se populacije razdvajaju i time tijekom vremena postaju zasebne vrste.

Iako neke od Darwinovih ideja nisu bile nove, one su izgradile prve čvrste temelje na kojima je nastala evolucijska biologija.
S aspekta genetike i stvaranja novih generacija, GA se oslanja na pet temeljnih postavki:

1. plodnost vrsta – potomaka uvijek ima više negoli je potrebno,
2. količina hrane je ograničena (iz čega posljedično slijedi borba za hranu u kojoj veću šansu za preživljavanje imaju jače i prilagodljivije jedinke, a time i one direktno dobivaju mogućnost da se i same križaju i proizvode potomke),
3. veličina populacije je konstantna odnosno varira oko nekog konstantnog broja,
4. kod vrsta koje se spolno razmnožavaju nema identičnih kopija[2] (djeca su određena genetskim materijalom svojih roditelja, ali nisu istovjetna roditeljima već postoji određeno odstupanje),
5. najveći dio varijacija se prenosi nasljeđem (današnja genetika kaže da se pojedini geni mogu prenositi i do 10 generacija). [5,7]

[2] Moderna genetika kaže da su genetski najbliži jednojajčani blizanci. Unatoč tome, između njih također postoje određene genetske varijacije.

Holland je ostao oduševljen poveznicom između evolucije i uzgoja i rasploda životinja. Holland je sam kasnije izjavio: „*That's where genetic algorithms came from. I began to wonder if you could breed programs the way people would say, breed good horses and breed good corn*“ (citat preuzet iz [2]).
Cijelu priču adaptaciji živih bića kao odgovor na druge organizme ili kao promjenu okruženja i prijedlog kako bi se to moglo iskoristiti u računarstvu dao je u svojoj knjizi [7]. U njoj izlaže općenite principe adaptacije uključujući prijedlog genetskog algoritma.

Odjeljak 2.2.1. ukratko izlaže jedan primjer iz stvarnog života kako se križanjem više vrsta konja kako bi se dobila današnja vrsta Hrvatskog posavca.

2.2.1. Studija slučaja – nastanak Hrvatskog posavca

Hrvatski posavac je hrvatska autohtona vrsta konja nastala na području slivnog toka rijeke Save. Filogenetske temelje vuče iz konja keltskih i ilirskih plemena koje su Hrvati zatekli na ovim područjima, kao i u konjima koje su sa sobom doveli iz svoje domovine.

Krvnom analizom posavskog konja pokazano je da je on najsličniji konju u Poljskoj. Unatoč tome, pedološke, klimatske, prehrambene i druge prilike odigrale su ključnu ulogu u stvaranju pasmina i njihovih karakteristika. Tako su se, s obzirom na tjelesne mjere, težinu i temperament formirale dvije grupe konja. Konji sa sjevera i zapada spadaju u grupu hladnokrvnih konja, mirnijeg su temperamenta, težeg okvira i većih dimenzija, a najpoznatiji predstavnici su flamanci, ardenci, brabanti, oldenburzi, noričani, šajeri i drugi. S druge strane, na jugu i istoku se razvijala toplokrvna vrsta življeg temperamenta, lakšeg okvira i manjih dimenzija čiji su predstavnici pasmine lipicanac, nonius, arap, engleski punokrvnjak i drugi.

Hrvatski posavac nastao je poboljšavanjem arhaične populacije bušaka, tj. prvotne populacije posavskog bušaka koja je nastala na području Austro-Ugarske Monarhije. Bušak je bio konj u tipu toplokrvnih pasmina jer je selekcijski naglasak stavljen na poslušnost, izdržljivost i brzinu za vojne svrhe, te snagu i ustrajnost za poljodjelske svrhe.

Ključnu ulogu u formiranju današnjeg hrvatskog posavca odigrao je čovjek koji je kroz niz godina dobio današnje osnovne tjelesne dimenzije posavskog konja. Čovjek je želio selekcijom dobiti one osobine konja koje odgovaraju njegovoj svrsi, već prema tome za što ga je čovjek želio koristiti. Budući da se konj koristio za prijevoz, vuču i rad u poljodjelstvu, poboljšavanje je provođeno tako da se prvotni bušak poboljšavao križanjem uglavnom s toplokrvnim pasminama konja. Upravo iz tog razloga se u okretnosti i temperamentu hrvatskog posavca vidi utjecaj ponajviše arapskih, a zatim i lipicanskih te nonius pastuha.

Prof. Josip Ubl godine 1885. među prvima opisuje posavskog konja „Posavski konji su male bagre konji (13 - 14, ½ šakah) 136-154 cm veliki, male ili srednje velike glave, široka čela, otvorenih nozdrvah, srednjeg češće poput jelenova vrata, oštrog grebena, britkog hrptišta, kratkih leđa, kosih križićah, dubokih i širokih prsah, gdješto omašnog trbuha, što proizlazi iz načina krmljena sa mnogo nu manje hranive krme. Noge su čvrste, razmjerno kratke, često ispod koljena tanahne. Uztrajni i brzi su za kas, manje za trk, prilagodjeni uplivom pod kojim i žive. Zadovoljni i lošom krmom, priučni na sunčanu žegu, buru, kišu i snieg, prava su blagodat za predjele, gdje treba sad u vodu, sad u blato, a leti u silnu brznim kasom uzvitlanu prašinu.“ [8]

Uvođenjem poljoprivredne mehanizacije, posavski konj je izgubio vrijednost kao radni konj što je dovelo do pada broja konja. Unatoč tome, tijekom 60-ih godina pristupa se značajnom uvođenju krvi teških (hladnokrvnih) konja, primarno ardenskog tipa belgijskog konja, a s ciljem proizvodnje i izvoza konjskog mesa, čime posavski konj polagano prelazi iz skupine toplokrvnih u skupinu hladnokrvnih konja. Zainteresiranog čitatelja se upućuje na [9].

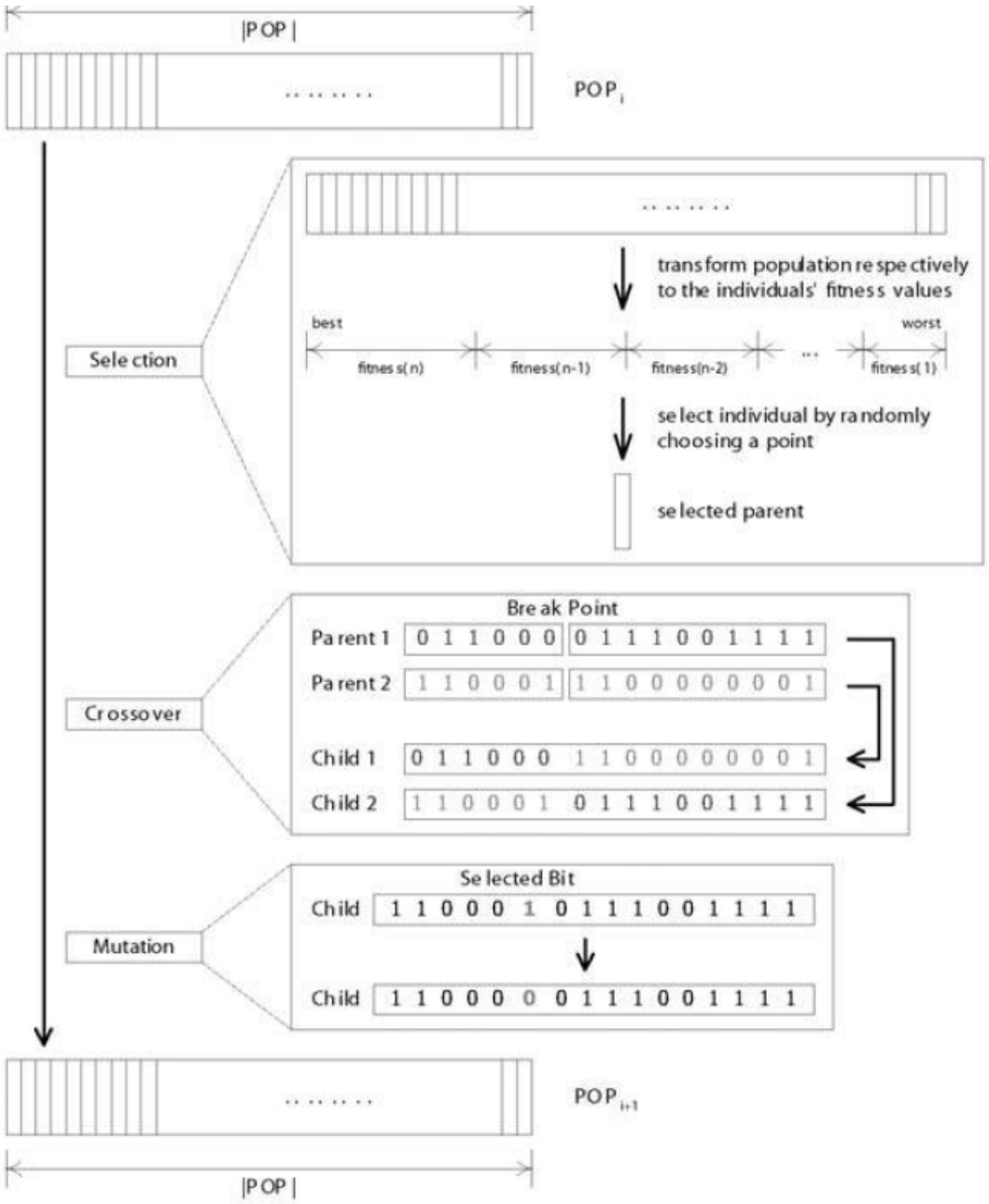
2.3. Genetski algoritam (GA)

Kao što je već rečeno u potpoglavlju 2.2., općenitu ideju genetskog algoritma dao je J. H. Holland u svojoj knjizi [7]. Ovo poglavlje bavit će se upravo genetskim algoritmom. Odjeljak 2.3.1. daje općenito pojašnjenje kako radi GA. Odjeljak 2.3.2. opisuje vrste GA i pojašnjenje osnovnih tipova genetskih algoritama; generacijskog GA, generacijskog GA s elitizmom i eliminacijskog GA.

2.3.1. Kako radi GA?

Genetski algoritam je iterativna procedura koja najčešće radi nad populacijom jedinki konstantne veličine zvane još i „kandidati za rješenje“ i „kromosomi“. Budući da su u prirodi upravo kromosomi nositelji genetskog materijala, u nastavku će se najčešće koristiti taj termin. Svaka od tih jedinki (najčešće prikazana nizom - vektorom) je jedno moguće rješenje problema. Svakoj jedinki moguće je odrediti dobrotu (engl. *fitness*) – mjeru u kojoj količini dano rješenje/kromosom rješava problem kojem je pridružena. Ovisno o tipu problema, može se tražiti da dobrota bude što manja (ako se rješava minimizacijski problem) ili što veća (ako se rješava maksimizacijski problem). Dakako, zbog načina zapisa brojeva u računalu (i mogućeg preljeva) u oba slučaja moraju postojati donja ili gornja granica.
Kako bi se stvorio novi kromosom, prvo moraju biti odabrane dvije jedinke koristeći neki operator selekcije (operatorima selekcije bavi se potpoglavlje 2.4.1.). Dva odabrana roditelja križaju se nekim od operatora križanja (operatorima križanja bavi se potpoglavlje 2.4.2.) čime se emulira izmjena (rekombinacija) genetskog materijala,

te se nakon toga (budući da u prirodi ne postoje potpune kopije) dobiveni genetski materijal mutira s određenom vjerojatnošću (operatorima mutacije bavi se potpoglavlje 2.4.3.). Dobiveni genetski materijal naziva se djetetom (dječjim kromosomom) te se kromosom dijete potom unosi u populaciju rješenja zamjenjujući pritom neko prijašnje rješenje.
Osnovni princip rada GA prikazan je na slici 2.1. Slika je preuzeta iz [10].



Slika 2.1. Osnovni princip rada genetskog algoritma. Slika je preuzeta iz [10].

Genetski algoritam je, kao što je u uvodnom dijelu rečeno, optimizacijski algoritam. GA, pritom, spada u domenu algoritama namijenjenih kombinatoričkim optimizacijama. Njegov je cilj pronaći takvu kombinaciju elemenata rješenja da dobrota tog rješenja bude optimalna.

Postavlja se pritom pitanje što se smatra optimalnim. Odgovor na to pitanje ovisi o problemu. Kod nekih problema cilj je pronaći rješenje čija dobrota teži nekoj maksimalnoj vrijednost – takvi se problemi često zovu problemi maksimizacije. Kod drugih, minimizacijskih problema, cilj je pronaći takvo rješenje čija je dobrota što je moguće manja. Dakako, primjenom nekoliko jednostavnih matematičkih operacija minimizacijske probleme je moguće svesti na maksimizacijske i obratno.
Primjer maksimizacijskog problema je i problem učenja robota Robby kojim se bavi 4. poglavlje.

2.3.2. Vrste genetskog algoritma

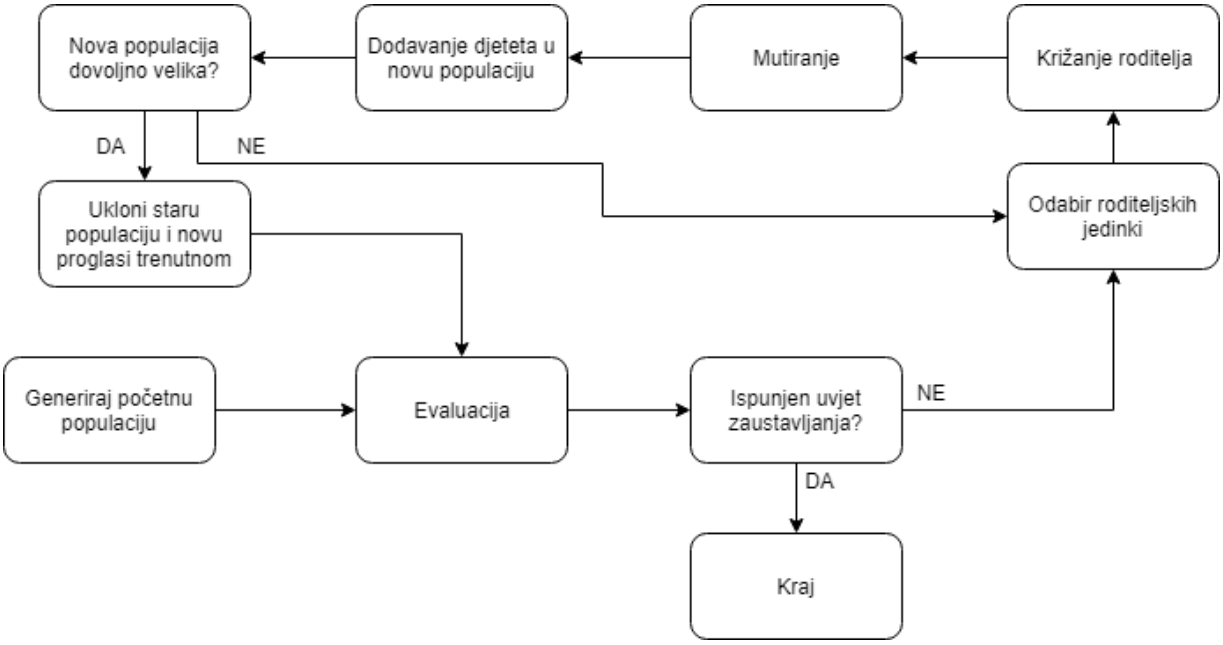
Najosnovnija podjela vrsta GA je na slijedne (sekvencijske) i paralelne. Ideja iza koje stoje paralelni GA je ta da se zadatak podijeli u više particija i da se potom ti podzadaci rješavaju paralelno (višedretveno na jednom računalu ili distribuirano na više računala). Ovo potpoglavlje se u nastavku bavi slijednim izvedbama algoritma.

Slijedne izvedbe GA obično se dijele na dvije verzije; generacijski genetski algoritam (engl. *generational genetic algorithm*) i eliminacijski genetski algoritam (engl. *steady-state genetic algorithm*). Usto, često se uz generacijski GA spominje i verzija s elitizmom. U nastavku su opisana sva tri tipa GA.

2.3.2.1. Generacijski genetski algoritam

Kao što je rečeno u prethodnom odlomku, generacijski GA svrstava se u slijedne izvedbe GA. Kod generacijskog GA postoji točna linija razdiobe između populacije roditelja i populacije djece. Generacijski GA iz trenutne populacije roditelja bira dva roditelja operatorom selekcije, njih križa i mutira, te dobiveno dijete smješta u iduću generaciju. U trenutku kad se iduća generacija napuni tj. kad je broj djece u idućoj generaciji jednak broju roditelja u trenutnoj, populacija roditelja izumire i populacija novostvorene djece postaje trenutna populacija koja se potom evaluira i na temelju koje se, ako nije ispunjen uvjet za prekid, zatim postupak ponavlja.
Ovakva izvedba GA najbolje pokazuje dominaciju nekoliko jedinki koje bi mogle pomoći u izbjegavanju preuranjene konvergencije[3]. Više o preuranjenoj konvergenciji bit će riječi u 3. poglavlju.
Princip rada generacijskog GA prikazan je slijednim dijagramom na slici 2.2.

[3] Zainteresiranog čitatelja upućuje se na [11]



Slika 2.2. Slijedni dijagram generacijskog genetskog algoritma

Generacijski GA prikazan je pseudokôdom 2.1.

```
procedura generacijskiGA()
   trenutnaPopulacija = generiraj_početnu_populaciju_jedinki
   evaluiraj(trenutnaPopulacija)

   dok nije pronađen željeni fitness ili nije maksimalni broj generacija
   čini
      novaPopulacija = ∅

      dok je veličina(novaPopulacija) < velična(trenutnaPopulacija) tada
         roditelj1, rodtielj2 = odaberi_dvije_jedinke(trenutnaPopulacija)
         dijete = križaj(roditelj1, roditelj2)
         mutiraj(dijete)
         dodaj(dijete, novaPopulacija)
      kraj

      trenutnaPopulacija = novaPopulacija
      evaluiraj(trenutnaPopulacija)
   kraj
kraj procedure
```

Pseudokôd 2.1. Pseudokôd generacijskog genetskog algoritma

2.3.2.2. Generacijski genetski algoritam s elitizmom

Klasični generacijski GA zbog izumiranja prethodne populacije može naići na problem da se u nekom trenutku evolucije dobrota počne smanjivati. Tu na scenu nastupa strategija elitizma.

Ideja generacijskog GA s elitizmom je istovjetna generacijskom GA, osim što će se kod elitističkog najbolja jedinka (ili najboljih *n* jedinki, respektivno s obzirom na dobrotu) zadržati za iduću generaciju, čime se teoretski omogućava i besmrtnost najbolje dosad pronađene jedinke, ali uz to može dovesti i do preuranjene konvergencije.

Najčešće se za iduću generaciju zadržava tek najbolja jedinka prethodne generacije što se često zove *model zlatnog kaveza* (engl. *golden cage model*). Ako se na elitnu jedinku prije prenošenja u iduću generaciju primjeni mutacija, takav mehanizam naziva se *slabi elitizam* (engl. *weak elitism*).

Generacijski GA s elitizmom prikazan je pseudokôdom 2.2.

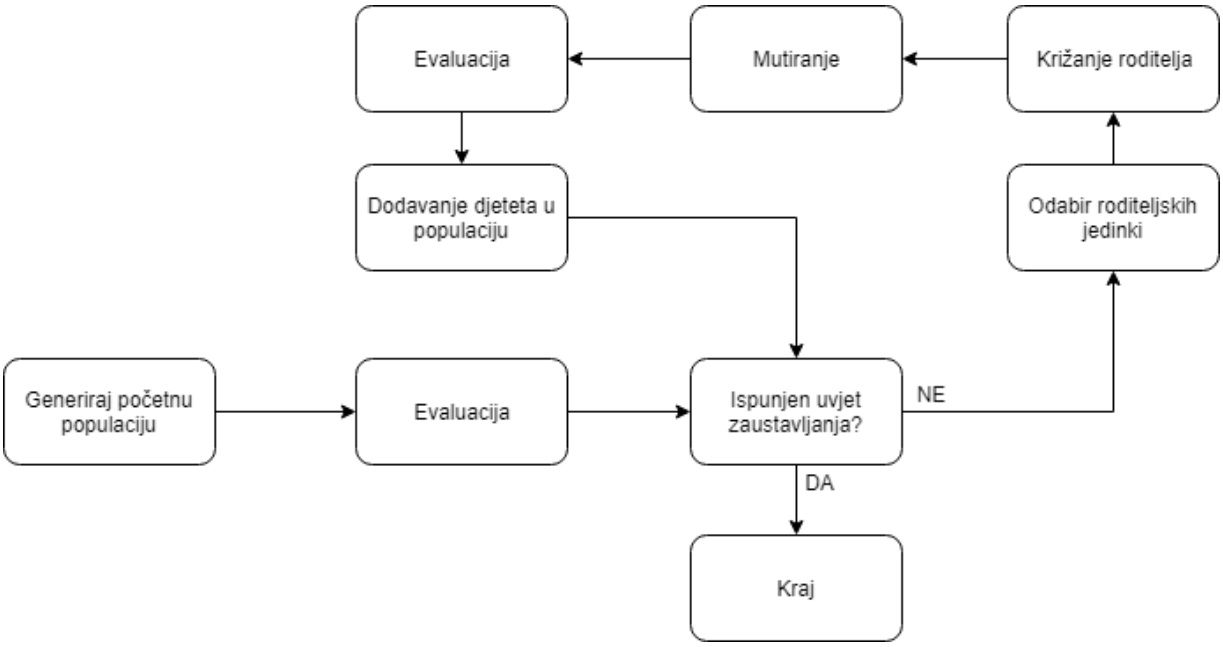
```
procedura elitističkiGeneracijskiGA()
   trenutnaPopulacija = generiraj_početnu_populaciju_jedinki
   evaluiraj(trenutnaPopulacija)

   dok nije pronađen željeni fitness ili nije maksimalni broj generacija
   čini
      novaPopulacija = ∅
      dodaj_najbolju_jedinku(novaPopulacija)

      dok je veličina(novaPopulacija) < velična(trenutnaPopulacija) tada
          roditelj1, rodtielj2 = odaberi_dvije_jedinke(trenutnaPopulacija)
          dijete = križaj(roditelj1, roditelj2)
          mutiraj(dijete)
          dodaj(dijete, novaPopulacija)
      kraj

      trenutnaPopulacija = novaPopulacija
      evaluiraj(trenutnaPopulacija)
   kraj
kraj procedure
```

Pseudokôd 2.2.: Pseudokôd generacijskog genetskog algoritma s elitizmom

2.3.2.3. Eliminacijski genetski algoritam

Za razliku od ideje razdvajanja dviju generacija predstavljene u generacijskom GA, eliminacijski GA ide u smjeru da još više oponaša prirodu i Darwinovu evoluciju. Darwin je, naime, utvrdio da jače i sposobnije jedinke imaju veće šanse za preživljavanje od slabijih i manje sposobnih na promjene. Upravo se tom idejom vodi eliminacijski GA.

Eliminacijski GA će, kao i generacijski, iz populacije izabrati dva roditelja operatorom selekcije, roditelje križati i mutirati te tako dobiveno dijete evaluirati i zatim ubaciti u populaciju tako da ono zamijeni neku drugu jedinku u populaciji (na primjer, najgoru).
Jedinka koja će u populaciji biti zamijenjena može se također odrediti operatorom selekcije, pri čemu bi bilo poželjno da se operator selekcije prilagodi tako da veću vjerojatnost da budu odabranu dobiju lošije jedinke.
Alternative tomu su da se iz populacije izbaci najstarija jedinka, najgora jedinka ili pak nasumično odabrana jedinka.
Princip rada eliminacijskog GA prikazan je slijednim dijagramom na slici 2.3.



Slika 2.3. Slijedni dijagram eliminacijskog genetskog algoritma

Eliminacijski GA prikazan je pseudokôdom 2.3.

```
procedura eliminacijskiGA()
    populacija = generiraj_početnu_populaciju_jedinki
    evaluiraj(populacija)

    dok nije pronađen željeni fitness ili nije dosegnut maksimalni broj
    nove djece čini
       roditelj1, rodtielj2 = odaberi_dvije_jedinke(trenutnaPopulacija)
       dijete = križaj(roditelj1, roditelj2)
       mutiraj(dijete)
       evaluiraj(dijete)

       lošaJedinka = odaberi_jedinku_za_zamjenu()
       ukloni(lošaJedinka, populacija)
       dodaj(dijete, populacija)
    kraj
kraj procedure
```

Pseudokôd 2.3. Pseudokôd eliminacijskog genetskog algoritma

Uz spomenute tri, GA ima i mnoštvo drugih verzija kojima se ovaj rad neće baviti. Zainteresiranog čitatelja se za to upućuje na [1,10].

2.3.3. Prikaz rješenja

Rješenja se mogu prikazati na različite načine; nizom bitova, nizom cijelih (ili prirodnih) brojeva, matricama, permutacijski, nizom decimalnih brojeva, itd. Pravi odabir prikaza rješenja omogućava laku i efikasnu manipulaciju rješenjima. Osnovne manipulacije nad rješenjima su:

- selekcija – rješenja se moraju moći odabrati iz skupa mogućih rješenja (više o operatorima selekcije bit će riječi u potpoglavlju 2.4.1.),
- križanje – rješenja se kombiniraju te temeljem dva ili više postojećih rješenja, stvaraju se nova (više o operatorima križanja bit će riječi u potpoglavlju 2.4.2.),
- mutiranje – cilj mutiranja je omogućiti manju promjenu nad trenutnim rješenjem (više o operatorima mutacije bit će riječi u potpoglavlju 2.4.3.)
- evaluacija – svakom rješenju potrebno je odrediti dobrotu (engl. *fitness*); mjeru u kojoj ono zadovoljava potrebe problema.

Odabir potrebnog prikaza rješenja ovisi o problemu kojeg je potrebno riješiti. U nastavku ovog odjeljka bit će prikazana reprezentacija rješenja nizom brojeva.

2.3.3.1. Prikaz rješenja nizom (poljem) brojeva

Prikaz nizom (poljem) brojeva je najučestaliji prikaz rješenja kombinatoričkih optimizacija. Polje može sadržavati binarne, cijele, decimalne, ali i bilo koje druge kombinacije brojeva, ovisno o potrebi (problemu kojeg je potrebno riješiti).

Jedan primjer rješenja poljem dekadskih brojeva prikazan je na slici 2.4. Pretpostavljeno je da se vrijednosti u polju smiju ponavljati (tj. da prikaz nije permutacijski).

Slika 2.4. Primjer prikaza poljem dekadskih brojeva

2.4. Operatori

Do sad su u ovom radu u više puta spomenuti operatori selekcije, križanja i mutiranja. Ovo potpoglavlje donosi kratki pregled sva tri tipa operatora, te pojašnjava odabrane operatore svakog od tipova.

2.4.1. Operatori selekcije

Prvi operatori s kojima se GA susreće u svom radu su operatori selekcije. Operatori selekcije zaduženi su za odabir dvaju rješenja roditelja koji će potom operatorima križanja proizvesti novo rješenje – dijete.
U ovom odjeljku opisana su dva operatora selekcije roditelja – jednostavna troturnirska selekcija i proporcionalna selekcija (selekcija proporcionalna dobroti).

2.4.1.1. Jednostavna troturnirska selekcija

Jednostavna troturnirska selekcija je varijacija na temu klasične turnirske selekcije. Kod klasične turnirske selekcije iz populacije se izvlači slučajni uzorak od *k* rješenja i kao jedan roditelj se odabire rješenje s najvećom dobrotom u izvučenom uzorku.

Kod jednostavne troturnirske selekcije broj uzoraka *k* iznosi 3. Također, za razliku od klasične turnirske selekcije, jednostavna troturnirska selekcija dat će dva „pobjednika“ turnira tj. od tri izvučena rješenja, izabrat će dva najbolja i oni će biti odabrani kao roditelji. U eliminacijskom GA, treće rješenje (kromosom) će tada najčešće biti eliminiran tako da bude zamijenjen djetetom nastalim od dva pobjednika turnira.

2.4.1.2. Proporcionalna selekcija (selekcija proporcionalna dobroti)

Selekciju proporcionalnu dobroti (engl. *fitness proportional selection*, može se naći i pod nazivom engl. *Roulette-wheel selection*) predložio je J. H. Holland u svojoj knjizi [7]. Ideja proporcionalne selekcije je ta da bolje jedinke imaju veću šansu da budu izabrane i da ta šansa bude proporcionalna njihovoj dobroti, tako se vjerojatnost odabira *i*-te jedinke iz populacije određuje prema izrazu:

$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_j} = \frac{f_i}{n \cdot \bar{f}} \tag{2.1}$$

gdje je f_i iznos dobrote *i*-tog rješenja. Za ispravno funkcioniranje ove selekcije važno je da dobrote svih jedinki budu nenegativne (ili da budu svedene na nenegativne).

2.4.2. Operatori križanja

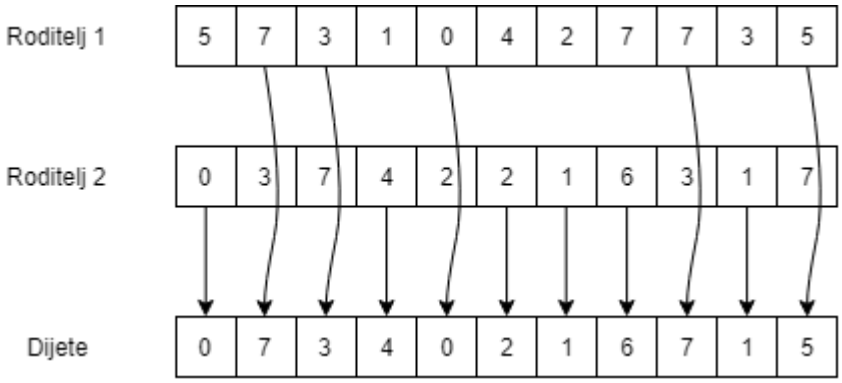
Do danas je u kontekstu GA razvijeno mnogo operatora križanja. Operatori križanja nisu isključivo ograničeni na GA, već se mogu slobodno koristiti i za potrebe drugih evolucijskih algoritama. Ovaj odjeljak donosi pregled odabranih operatora križanja. Svi operatori pretpostavljaju da je rješenje prikazano nizom (bitova, cijelih ili decimalnih brojeva i drugim).

2.4.2.1. Ravno križanje (engl. *flat crossover*)

Ideja ravnog križanja je da se *i*-ta komponenta genoma djeteta postavi na uniformno izabranu vrijednost u intervalu između manje vrijednosti *i*-te komponente jednog i veće vrijednosti i-te komponente drugog roditelja, tj. iz intervala [min(c_i^1, c_i^2), max(c_i^1, c_i^2)]. Pritom, oznaka c_i^n označava *i*-tu komponentu kromosoma roditelja *n* (pritom je *n* = 1, 2).
Valja napomenuti kako ovakvo križanje nije prikladno za slučajeve kad su rješenja prikazana poljem decimalnih brojeva.

2.4.2.2. Diskretno križanje (engl. *discrete crossover*)

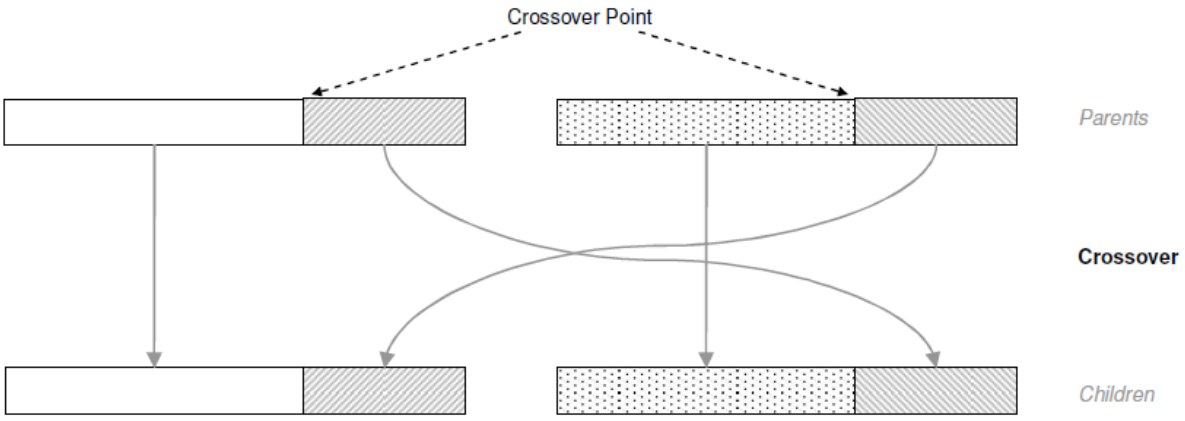
Diskretno križanje se provodi tako da se i-ta komponenta kromosoma djeteta postavi na i-tu komponentu jednog od roditelja. Ukratko, *i*-ta komponenta djeteta bit će ili *i*-ta komponenta prvog ili *i*-ta komponenta drugog roditelja.
Ideja diskretnog križanja prikazana je na slici 2.5. Pretpostavljen je prikaz rješenja nizom cijelih brojeva i da je skup mogućih vrijednosti u intervalu [0,7].



Slika 2.5. Diskretno križanje

2.4.2.3. Križanje s jednom točkom prekida

Križanje s jednom točkom prekida provodi se tako da se slučajno odabere indeks polja na kojemu se „reže“ oba roditeljska kromosoma i time se dobivaju dvije sekcije glave i dvije sekcije repa. Potom se dvije sekcije repa zamijene i time se dobivaju dvije nove jedinke.
Shematski prikaz križanja s jednom točkom prekida prikazan je na slici 2.6. Slika je preuzeta iz [10].



Slika 2.6. Shematski prikaz križanja s jednom točkom prekida. Slika je preuzeta iz [10].

2.4.2.4. Križanje s *n* točaka prekida

Križanje s *n* točaka prekida prirodni je nastavak na priču o križanju s jednom točkom prekida. Nasumce se odabire *n* točaka gdje se „režu“ oba roditeljska kromosoma te se potom na njima izmjenjuju dijelovi roditeljskih kromosoma. Odabir većeg broja točaka prekida ima svoje prednosti i mane. Prednost odabira većeg *n* je ta da zbog gotovo uniformnog uzorkovanja dolazi do prikladnijeg kombiniranja dobrih osobina prisutnih u nizovima, ali mana je to što tada takvo križanje remeti izgradnju dobrih kromosoma s porastom broja točaka prekida tj. evolucija dobrih rješenja postaje sve teža.

2.4.3. Operatori mutacije

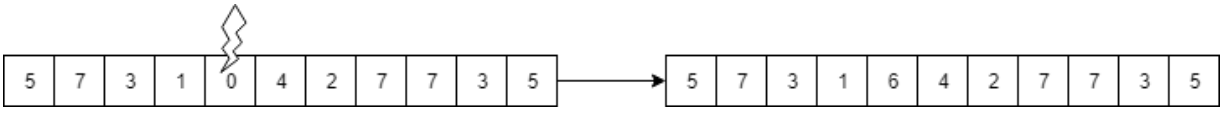
Kao što je izloženo u potpoglavlju 2.2., kod vrsta koje se spolno razmnožavaju nema identičnih kopija, već su djeca određena genetskim materijalom svojih roditelja, ali nisu istovjetna roditeljima, već postoji određeno odstupanje. Za to odstupanje brinu se operatori mutacije.

Iako su kod evolucijskih strategija operatori mutiranja primarni način generiranja novih jedinki, kod GA operatori mutacije služe kako bi se izbjegla jedinka istovjetna roditeljima, a time posljedično i preuranjena konvergencija (o kojoj će riječi biti u 3. poglavlju). Iako bi se iz ovoga moglo pomisliti da je mutacija većeg broja kromosoma nužno dobra, to najčešće nije istina. Prejaka mutacija kod GA ima razarajuću funkciju tako da dobra svojstva koja su možda dobivena operatorom križanja ona odvede u krivom smjeru u pretrazi čime evolucija dobrih rješenja postaje sve teža.

U nastavku su izložena dva najjednostavniji operatora mutacije; mutacija jednog gena i mutacija s određenom vjerojatnošću. Oba ovdje prikazana operatora mutacije namijenjena su prikazu rješenja nizom bitova ili cijelih brojeva.

2.4.3.1. Mutacija jednog gena

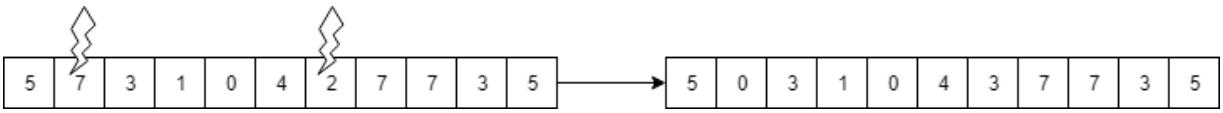
Kao što je u nazivu rečeno, mutacija jednog gena nasumce izabire jedan indeks u nizu i vrijednost tog indeksa postavlja na slučajno odabranu vrijednost iz skupa mogućih vrijednosti. Vrijednost može biti odabrana koristeći uniformnu ili normalnu (Gaussovu) razdiobu.
Mutacija jednog gena prikazana je na slici 2.7. Pretpostavljen je prikaz rješenja nizom cijelih brojeva i da je skup mogućih vrijednosti u intervalu [0,7].



Slika 2.7. Shematski prikaz mutacije jednog gena

2.4.3.2. Mutacija s vjerojatnošću

Mutacija s vjerojatnošću provodi se tako da se odabere vjerojatnost mutacije p_m (tipično između 1% i 5%, nerijetko i manje). Potom se svaki element niza kojim je prikazano rješenje mutira s tom vjerojatnošću. Ako je element u nizu izabran za mutaciju, njegova se vrijednost postavlja na slučajno odabranu vrijednost iz skupa mogućih vrijednosti.
Mutacija s vjerojatnošću prikazana je na slici 2.8. Pretpostavljen je prikaz rješenja nizom cijelih brojeva, skup mogućih vrijednosti u intervalu [0,7] te da su za mutiranje odabrani elementi s indeksom 1 i 6.



Slika 2.8. Shematski prikaz mutacije s vjerojatnošću

3. Preuranjena konvergencija. Strategije usporavanja preuranjene konvergencije

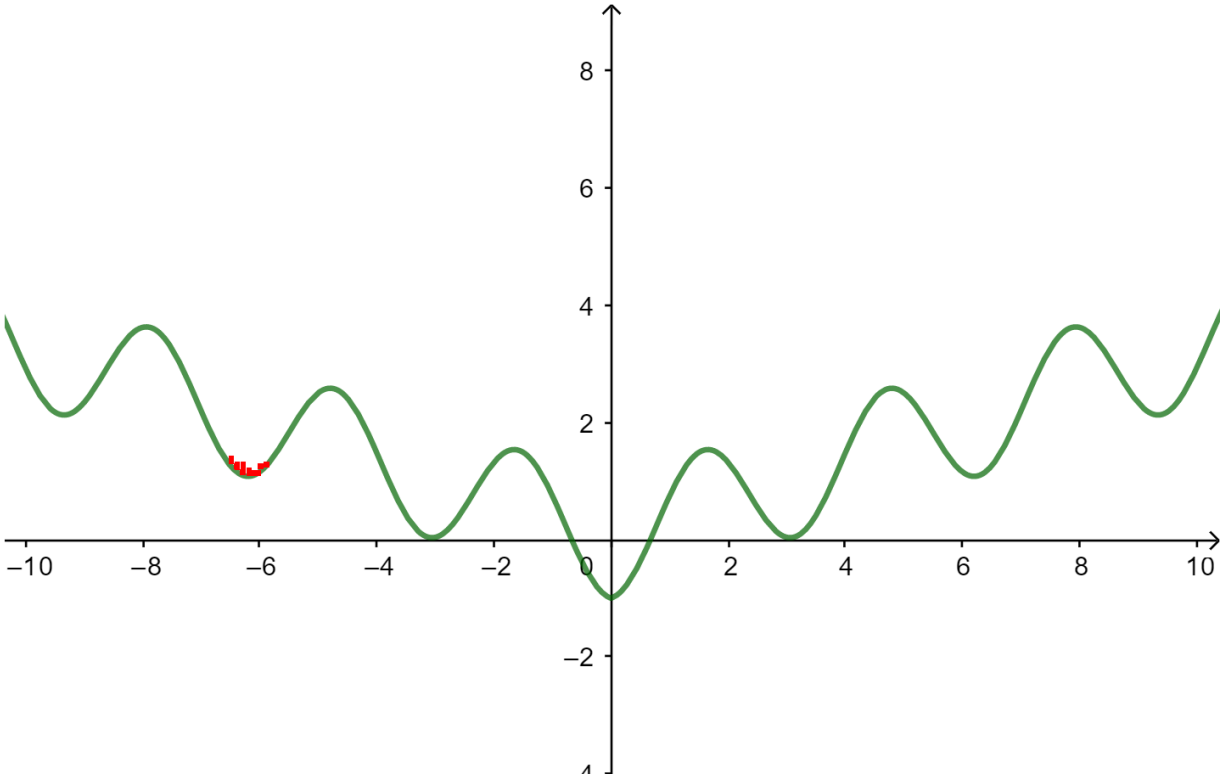
U uvodu i u 2. poglavlju spomenuto je kako operator mutacije kod GA služi primarno kako bi se izbjegla preuranjena konvergencija. Ovo poglavlje se bavi detaljnijim pojašnjenjem pojave preuranjene konvergencije. U potpoglavlju 3.1. opisano je što je preuranjena konvergencija i zašto je ona loša za postupak optimizacije. Potpoglavlje 3.2. se bavi strategijama usporavanja (izbjegavanja) preuranjene konvergencije izlažući dvije vrlo česte strategije; selekciju potomaka i samoadaptivni segregacijski genetski algoritam s aspektima simuliranog kaljenja.

3.1. Preuranjena konvergencija

Jedna od najvećih prijetnji postupku optimizacije korištenjem metaheurističkih optimizacija je stagnacija. Stagnacija je pojava da se „rješenja“ zabiju u neki dio prostora pretrage koji je lokalni, ali ne i globalni optimum.

Međutim, prilikom korištenja evolucijskih algoritama koji koriste operatore križanja korisnici mogu naići na problem stagnacije u lokalnom, ali ne i u globalnom optimumu. U kontekstu GA, ta se stagnacija naziva preuranjena konvergencija i pojavljuje se kad se populacija rješenja „zabije“ tj. kad dođu do nekog suboptimalnog stanja takvog da operatori križanja i mutiranja više ne mogu proizvesti djecu koja bi svojom dobrotom nadmašivala roditelje. Tada genetska informacija sadržana u rješenjima ne sadrži onu genetsku informaciju koja bi bila potrebna za daljnje poboljšanje rješenja.

Primjer preuranjene konvergencije prikazan je na slici 3.1. Pretpostavljen je minimizacijski problem tj. da se želi naći globalni minimum. U primjeru se dogodilo to da su sva rješenja (označena crveno) zapela oko točke $x = -6{,}2$ u kojoj funkcija $f(x)$ ima vrijednost $f(x) = 1{,}08$, dok globalni optimum u $x = 0$ iznosi $f(x) = -1$. Budući da su sva rješenja gotovo u istoj točki prostora i da se kod GA koristi slaba mutacija, šansa da se GA izvuče iz ovakve situacije je vrlo mala i proglašava se preuranjena konvergencija.



Slika 3.1. Ilustrativni prikaz stagnacije u lokalnom optimumu

U evoluciji u prirodi održavanje genetske raznolikosti igra ključnu ulogu jer bogat genetski „bazen“ omogućava da se određene vrste adaptiraju na promjenjive uvjete okoliša. Za razliku od toga, kod umjetne evolucije uvjeti okoliša su konstantni budući da je funkcija dobrote nepromijenjena za čitavog trajanja evolucije te se kromosomi moraju adaptirati upravo na nju. Upravo iz tog razloga, preuranjena konvergencija kod GA nije rezultat nemogućnosti adaptacije na funkciju dobrote već moguć gubitak ključne genetske informacije odnosno onog dijela rješenja koji je ujedno dio globalno najboljeg rješenja. Ako dođe do takvog gubitka, preuranjena konvergencija je gotovo sigurna jer tada jedini način za doći do te genetske informacije sada postaje operacijom mutacije.

Potpoglavlje 3.2. bavi se strategijama usporavanja preuranjene konvergencije kod GA.

3.2. Strategije usporavanja preuranjene konvergencije

Na prethodnoj slici koja ilustrira preuranjenu konvergenciju, globalni optimum je poznat. U stvarnim situacijama stvari ponekad nisu takve, već se želi naći globalni maksimum ili minimum ne znajući koji je njegov konkretan iznos i tada je preuranjena

konvergencija prilično nezgodna pojava jer može dati rezultat u lokalnom optimumu i njega prikazati kao globalni optimum, iako je razlika između lokalnog i globalnog optimuma vrlo velika. Takva pojava je prihvatljiva ako je problem kojeg je potrebno riješiti moguće riješiti u relativno kratkom vremenskom roku, te se tada evolucija jednostavno pokrene nekoliko puta i odabere najbolje rješenje na temelju nekoliko pokretanja. Međutim, složenije evolucije koje evoluiraju rješenja složenijih problema mogu trajati danima, pa čak i tjednima te u takvim situacijama nije svejedno hoće li doći do pojave preuranjene konvergencije ili ne. Upravo iz tog razloga znanstvenici su tijekom vremena razvili posebne strategije koje bi omogućile robusnost GA na preuranjenu konvergenciju te kako bi si time povećali vjerojatnost da će nakon nekoliko dana ili tjedana evolucije dobiti rješenje vrlo blisko ili jednako globalnom optimumu.
Jedan od prvih načina praćenja genetske raznolikosti u GA dali su Srinivas i Patnaik u radu [12] uzimajući kao mjeru genetske raznolikosti razliku između prosječne dobrote rješenja i najveće pronađene dobrote te u ovisnosti o toj mjeri adaptivno mijenjajući vjerojatnosti križanja i mutiranja.
Ovo potpoglavlje bavi se dvjema takvim strategijama; selekcijom potomaka (engl. *Offspring Selection*) i samoadaptivnim segregacijskim genetskim algoritmom s aspektima simuliranog kaljenja (engl. *Self Adaptive SEgregative Genetic Algorithm with Simulated Annealing aspects*).

3.2.1. Selekcija potomaka

Selekciju potomaka (engl. *Offspring Selection*, u daljnjem tekstu: OS) svrstava se u razred genetskih algoritama koji su dizajnirani da funkcioniraju kao građenje rješenja od gradivnih elemenata tj. da kombiniraju elemente kromosoma koji definiraju dobra rješenja.

Ideja selekcije potomaka je nakon odabira roditelja, križanja i mutiranja uvesti novi selekcijski mehanizam koji razmatra uspješnost upravo primijenjene reprodukcije. Taj mehanizam stvorit će samoadaptivni selekcijski pritisak koji ovisi o tome koliko je lako ili teško postići napredak tj. boljeg potomka.
Kako bi se osiguralo da se postiže napredak, OS inzistira na tome da se populacija djece u idućoj generaciji mora sastojati od određenog broja (postotka) djece koja nadmašuju roditelje. Takva djeca se nazivaju uspješnom djecom. Stoga se uvodi novi parametar, postotak uspješnosti $SuccRatio \in [0,1]$. *SuccRatio* je fiksna vrijednost, najčešće između 20% i 40%, a kazuje omjer uspješne djece i ukupnog broja djece u novoj populaciji. Pritom su populacije fiksne veličine, a veličina populacije je označena sa |POP|.

Sada se postavlja pitanje kada se dijete može smatrati uspješnim. Najčešće se kod strategije selekcije potomaka dijete smatra uspješnim ako je njegova dobrota

prelazi prag koji je smješten između dobrote lošijeg i dobrote boljeg roditelja. Kako bi se osiguralo progresivno podizanje kriterija uspješnog djeteta, uvodi se novi parametar, faktor usporedbe $CompFactor \in [0, 1]$. Kad faktor usporedbe (*CompFactor*) ima vrijednost 0.0, dijete se smatra uspješnim ako njegova dobrota barem jednaka dobroti lošijeg roditelja, dok faktor usporedbe 1.0 označava da dijete mora imati bolju dobrotu od boljeg roditelja da bi se smatralo uspješnim. Kao *CompFactor* najčešće se uzima omjer trenutne iteracije i ukupnog broja dopuštenih iteracija evolucije. Tijekom izvođenja algoritma, *CompFactor* progresivno raste od donje prema gornjoj granici rezultirajući širom pretragom u početku i usmjerenijom pretragom pri kraju izvođenja. Valja još napomenuti da je ta ideja posuđena iz algoritma simuliranog kaljenja.

Nakon stvaranja svakog djeteta, provjerava se je li dijete uspješno. Ako je dijete uspješno, odmah se smješta u iduću generaciju. Ako pak dijete nije uspješno, ne odbacuje se, već se smješta u priručni spremnik zvan bazen (engl. *pool*). Kad je u idućoj generaciji $SuccRatio * |POP|$ uspješne djece, ostatak $(1 - SuccRatio) * |POP|$ se puni sa slučajno odabranim jedinkama iz bazena (jedinkama koje nisu ostvarile uvjet uspješnosti) ili, ako je bazen manji od toga, ponovno se pristupa stvaranju potomaka – ovaj put svi potomci završavaju u idućoj generaciji sve dok se ona ne napuni tj. dok joj veličina nije |POP|. Tada prethodna generacija izumire i novu generaciju se proglašava aktualnom i postupak kreće ispočetka.

Uz to, uvodi se i novi parametar, *ActSelPress* koji se izračunava na kraju *i*-te generacije prema izrazu:

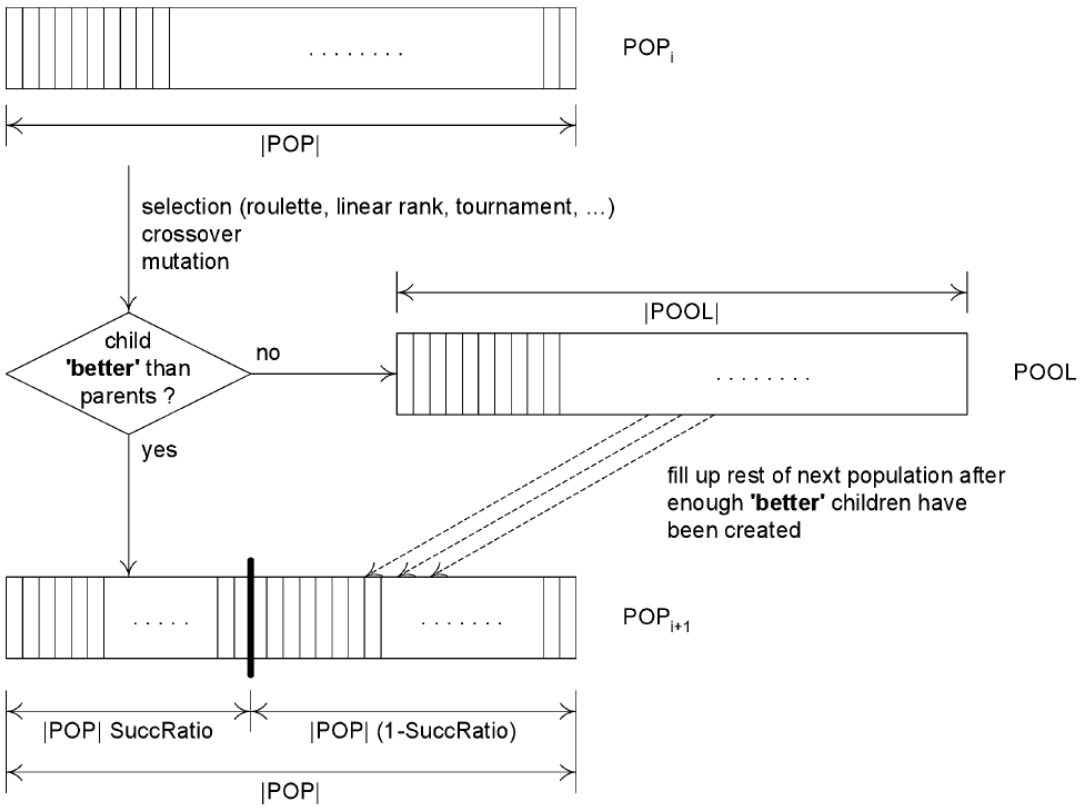
$$ActSelPress = \frac{|POP_{i+1}| + |POOL_i|}{|POP_i|} \tag{3.1}$$

gdje $|POP_{i+1}|$ označava veličinu populacije u *(i+1)*-generaciji, $|POOL_i|$ veličinu bazena na kraju *i*-te generacije, a $|POP_i|$ veličinu populacije u *i*-toj generaciji. Općenito vrijedi $|POP_{i+1}| = |POP_i|, \forall i.$

Nadalje, definira se i maksimalni selekcijski pritisak $MaxSelPress \in < 1, +\infty >$ koji je fiksna vrijednost i gornja granica selekcijskog pritiska. Ako niti nakon što je generirano $MaxSelPress * |POP|$ djece evolucija nije uspjela ostvariti $SuccRatio * |POP|$ uspješne djece, detektira se preuranjena konvergencija. Analogno tome, preuranjena konvergencija se detektira i ako je $ActSelPress \geq MaxSelPress$.

Važno je napomenuti kako algoritam ne dozvoljava ulazak klonova (duplikata u populaciju), što je čest razlog preuranjene konvergencije kod GA. Pritom valja paziti na to da je provjera često vremenski vrlo skupa te se savjetuje da se pri generiranju svakog rješenja izračuna kôd raspršivanja (engl. *hash code*) i njega spremi u pomoćnu varijablu rješenja. Ako funkcije raspršivanja rade dobro, vjerojatnost da će dva različita rješenja dati isti kôd raspršivanja je iznimno mala. Ako se, kojim slučajem

i dogodi da su dva kôda raspršivanja jednaka, tada se pristupa usporedbi svih elemenata rješenja dok se ne odredi je li novo rješenje duplikat ili različito od već postojećeg.
Dijagram tijeka strategije selekcije potomaka prikazan je na slici 3.2. Slika je preuzeta iz [13].



Slika 3.2. Dijagram tijeka algoritma selekcije potomaka. Slika preuzeta iz [13]

Za kraj, valja još skrenuti pozornost na mogućnost pretjeranog selekcijskog pritiska u implementaciji kojom bi ovaj algoritam zasigurno zapeo u lokalnom optimumu. Naime, jedna selekcija već postoji pri usporedbi potomka s roditeljima koristeći *SuccRatio* i jak dodatni selekcijski pritisak algoritam bi učinio pretjerano „agresivnim“ po pitanju selekcijskog pritiska i time bi povećao šanse da ga odvede u preuranjenu konvergenciju. Upravo iz tog razloga, kod ovog algoritma se savjetuje roditelje izabrati nekom manje „agresivnom“ selekcijom poput jednostavne troturnirske selekcije ili pak potpunim slučajnim odabirom (bez obzira na dobrotu).

3.2.2. Samoadaptivni segregacijski genetski algoritam s aspektima simuliranog kaljenja

Kod primjene GA na probleme kombinatoričke optimizacije s puno dimenzija, tijekom vremena, a pod utjecajem genetskog drifta, dolazi do odvlačenja rješenja u suboptimalne dijelove prostora pretrage. Time se u čitavoj populaciji gube dijelovi genetske informacije koji su ujedno i dijelovi globalnog optimuma. Ovo svojstvo posebno je izraženo ako su dobra rješenja (rješenja velike dobrote) skrivena među puno rješenja s lošijom ukupnom dobrotom.

Kako bi se doskočilo tom problemu, čitava se populacija dijeli na određeni broj potpopulacija (ponegdje zvanih i genetske niše) što potom uzrokuje odvojeno evoluiranje potpopulacija koje tada istražuju različite genetske informacije tj. različite dijelove prostora pretrage. Ideja je razviti koncepte koji će gradivne blokove globalno optimalnog rješenja, u početku razasute među potpopulacijama, približiti kako bi tvorile jednu veliku populaciju.

3.2.2.1. Biološka crtica

U prirodi je podjela populacije jedne vrste na više potpopulacija često viđen fenomen. Posljedica takve podjele je genetska diferencijacija potpopulacija – pojava da se potpopulacije razvijaju neovisno i sve više genetski udaljavaju. Pojava genetske diferencijacije uzrokovana je lokalnim podešavanjem različitih genotipova u različitim potpopulacijama i genetskim driftom potpopulacija. Posljedica takve strukture populacije je gubitak genetske varijacije (mogućih gena za neku karakteristiku) u potpopulacijama čime se svaka potpopulacija usmjerava u jednom smjeru razvoja.

Taj problem prvi je opisao genetičar Sten Wahlund[4]. Ujedno, Wahlund je opisao i da genetska varijacija ponovno raste ako se struktura potpopulacija razbije i križanje postane moguće na razini čitave populacije što se ostvaruje migracijama. Važno je pritom napomenuti da do razmjene genetskog materijala i protoka gena dolazi samo ako razmijenjene jedinke proizvedu potomka. Tada u potpopulaciju dolaze novi aleli[5].

Sve ovo u svom radu koristi samoadaptivni segregacijski genetski algoritam s aspektima simuliranog kaljenja. Njime se bavi idući odjeljak.

3.2.2.2. Algoritam

Samoadaptivni segregacijski genetski algoritam s aspektima simuliranog kaljenja (engl. *Self Adaptive SEgregative Genetic Algorithm with Simulated Annealing*

[4] Zainteresiranog čitatelja se upućuje na [14].

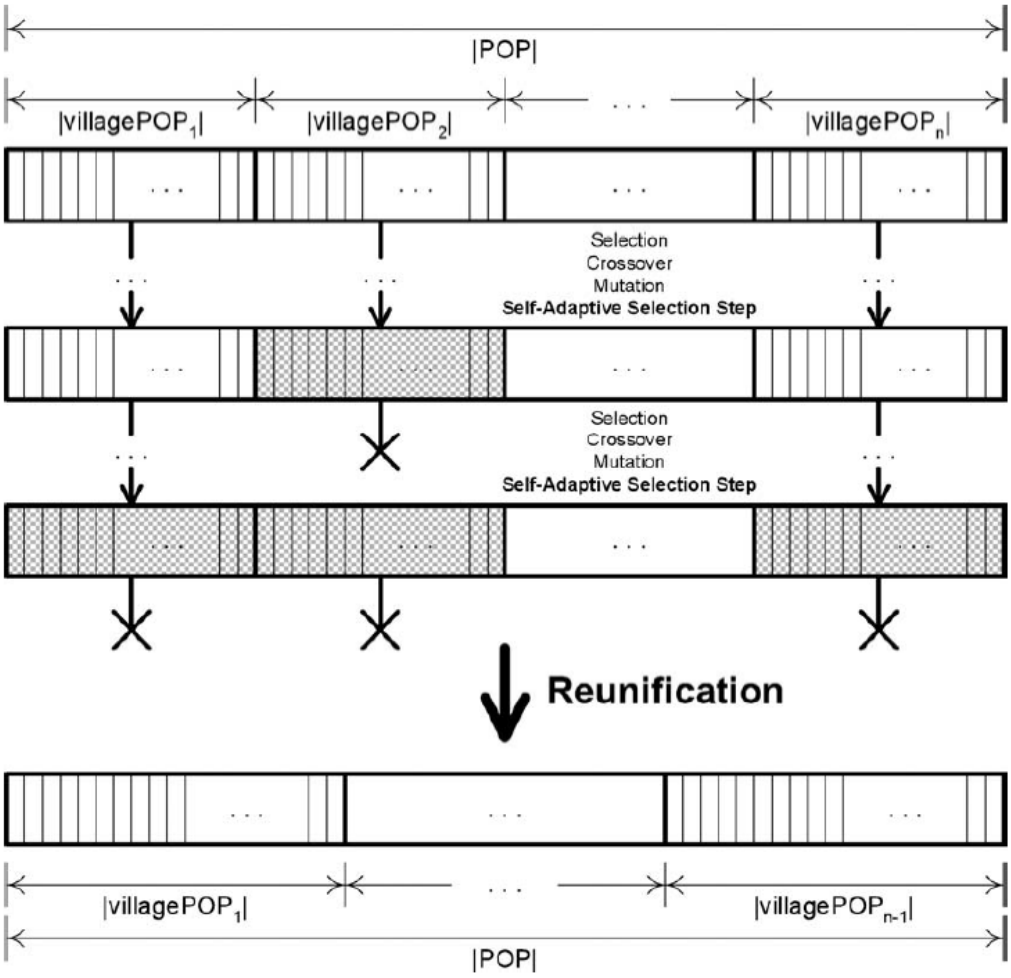
[5] Različite mogućnosti za neku od karakteristika (npr. aleli su različite boje očiju).

aspects, u daljnjem tekstu: SASEGASA) u odnosu na klasični GA donosi dva poboljšanja:

1. koristi varijabilni selekcijski pritisak kakav je predstavljen u potpoglavlju 3.2.1. (strategija selekcije potomaka) i
2. separira populaciju u potpopulacije s ciljem povećanja širine pretrage.

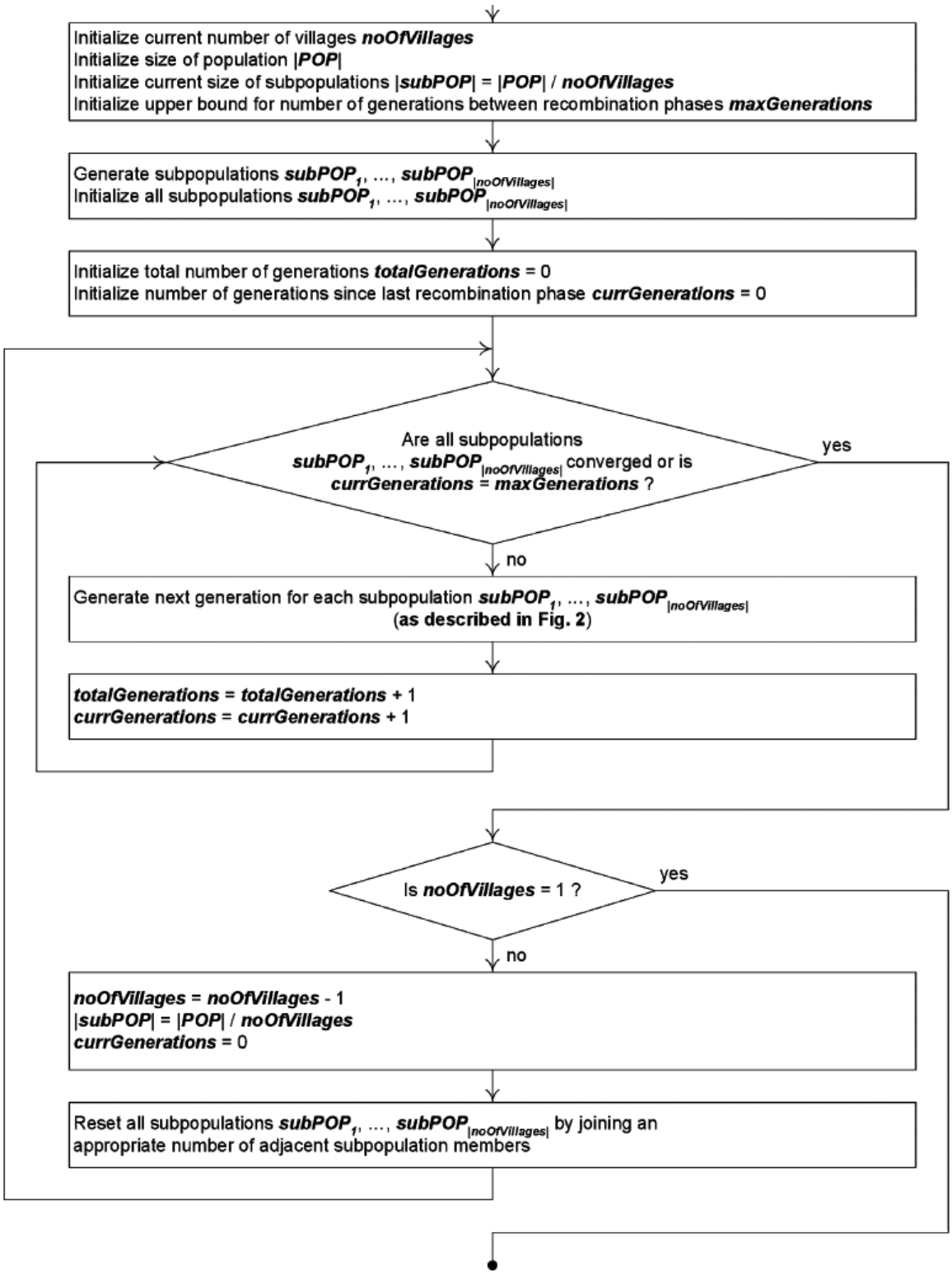
SASEGASA rad počinje s *n* potpopulacija koje se mogu zamisliti kao sela. Svako od tih sela ima drift prema različitim regijama prostora te se svako od njih evoluira potpuno neovisno od drugih sve dok ne dođe do preuranjene konvergencije u svim potpopulacijama. Evolucija potpopulacija se radi genetskim algoritmom sa strategijom selekcije potomaka koja, kao što je rečeno u potpoglavlju 3.2.1., ima ugrađenu detekciju preuranjene konvergencije. U trenutku kad se u svim potpopulacijama detektira preuranjena konvergencija, ukupni broj potpopulacija se smanjuje za 1 i primjenjuje se operator migracije tako da u jednu potpopulaciju dolaze kromosomi iz susjedne potpopulacije, te se time u svaku potpopulaciju unosi novi genetski materijal, što prema [14] u konačnici omogućava ponovni rast genetske varijacije, a time i pretraživanje šireg prostora.

Dijagram tijeka migracija među potpopulacijama prikazan je na slici 3.3. Oznaka $|POP|$ označava veličinu ukupne populacije, dok oznaka $|villagePOP_i|$ označava veličinu *i*-te potpopulacije („sela“). U početku vrijedi da je $i = 1, \dots, n - br\_iter$, gdje ***br_iter*** označava broj iteracije tj. koliko je migracija primijenjeno od početka evolucije. Svijetlo osjenčane potpopulacije su u postupku evolucije dok su tamno osjenčane potpopulacije preuranjeno konvergirale. Slika je preuzeta iz [13].



Slika 3.3. Dijagram tijeka evolucije potpopulacija i migracije. Slika preuzeta iz [13]

Dijagram tijeka rada SASEGASA strategije prikazan je na slici 3.4. Slika je preuzeta iz [13].



Slika 3.4. Dijagram tijeka izvođenja strategije SASEGASA. Slika je preuzeta iz [13]

4. Problem učenja robota Robby

Ovo potpoglavlje bavi se problemom učenja robota Robby izloženog u knjizi [2]. Potpoglavlje 4.1. opsuje problem kojeg se rješava. Potpoglavlje 4.2. opisuje programsko rješenje problema tj. izlaže načine rješavanja problema koristeći GA i strategije usporavanja preuranjene konvergencije. Eksperimentalni rezultati izvođenja, primjedbe i zaključci dani su u potpoglavlju 4.3.

4.1. Učenje robota Robby

Ideja robota Robby izložena je u knjizi „Complexity: A Guided Tour“ američke računarske znanstvenice Melanie Mitchell.
Robot Robby je uposlen da u koordinatnoj mreži dimenzija 10x10 polja (pločica) skuplja limenke ostavljene nakon zabave. Na svakom od polja može se nalaziti limenka ili ništa. Robbyjev svijet (koordinatna mreža) s vanjske strane je ograđena zidovima. Robby može stati na prazna polja i na polja na kojima je limenka, ali ni na koji način ne može prijeći zid ili stati na njega.

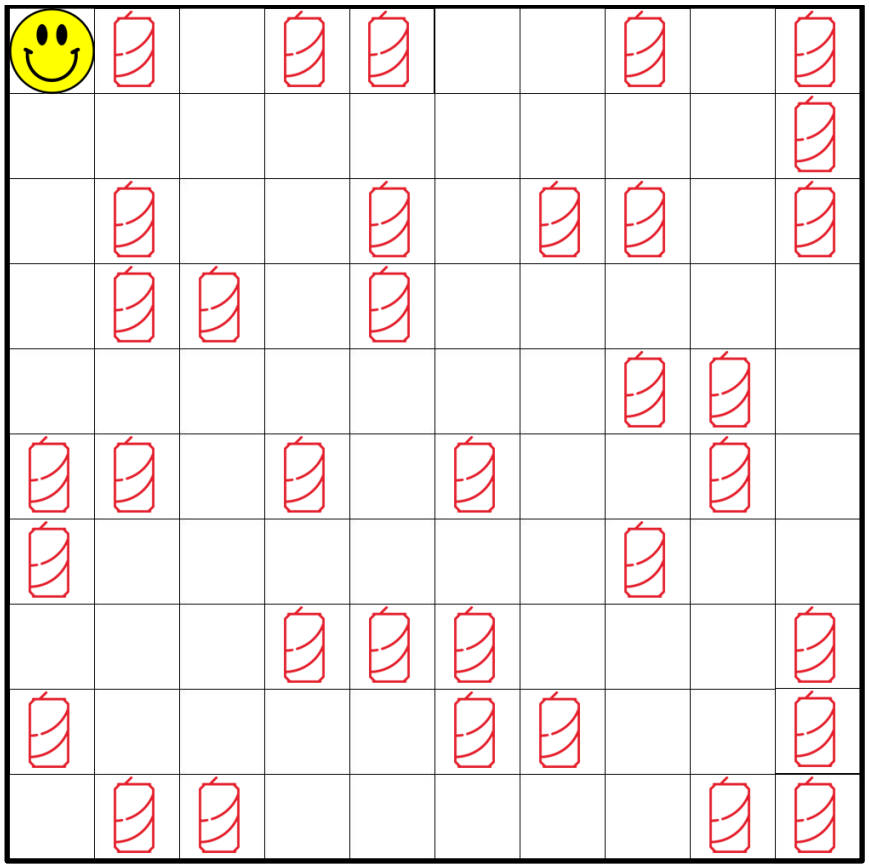
Međutim, Robbyjevi tvorci nisu imali novca pa su umjesto vrhunskih kamera ugradili najjeftinije, tako da Robby koristeći njih vidi što je na polju na kojem se trenutno nalazi te što je jedno polje gore, jedno dolje, jedno lijevo i jedno desno. Uz to što nisu imali novca za bolje kamere, nisu imali novca niti za bolju bateriju, tako da Robby po jednom čišćenju može napraviti maksimalno 200 akcija. Moguće akcije su:

1. pomak jedno polje gore,
2. pomak jedno polje dolje,
3. pomak jedno polje lijevo,
4. pomak jedno polje desno,
5. pomak jedno polje u slučajno odabranom smjeru (pritom polje na koje se pomiče mora s trenutnim dijeliti brid polja),
6. sagni se i pokupi limenku te
7. ništa.

Robbyjevi tvorci, iako nemaju novca, vrhunski su programeri te žele programski evoluirati „mozak“ robota kako bi on u najviše 200 akcija pokupio što je više moguće limenki koristeći pritom sve informacije koje daju kamere. Robby nema niti mnogo memorije – štoviše, ima točno onoliko koliko mu je potrebno da se u nju pohrani samo strategija sakupljanja limenki, ali ne i njegove trenutne koordinate ili polja koja je već posjetio. Ukratko, Robbyjeve akcije nisu motivirane trenutnom pozicijom u rešetci ili pozicijama koje je već obišao, već isključivo onime što vidi na svoje kamere.

Jedan od mogućih primjera svijeta robota Robby u početnom trenutku (prije početka skupljanja limenki) prikazan je na slici 4.1. Grafika smješka koja je ovdje korištena za

prikaz robota Robby preuzeta je sa http://www.clipart-library.com (autor nepoznat). Ikonu limenke izradio je autor *Feepik* sa stranice http://www.flaticon.com. Na slici je pretpostavljeno da Robby mora pokupiti 35 limenki.



Slika 4.1. Jedan od mogućih prikaza svjetova robota Robby

Početna koordinata robota je (0,0) i Robby te koordinate, kao što je prethodno rečeno, nije svjestan. Ono čega je Robby svjestan je njegova okolina prikazana na slici – s lijeve strane vidi zid, s gornje strane vidi zid, s desne strane vidi limenku, s donje strane vidi prazno polje i na polju gdje se on nalazi također vidi prazno polje. Na temelju tih pet informacija on mora donijeti odluku koju će akciju poduzeti. Važno je napomenuti da Robby, kad stane na polje gdje je limenka, ne mora nužno pokupiti tu limenku. On će limenku pokupiti samo ako mu to kaže strategija koju je cilj evoluirati.

Koristeći genetski algoritam, cilj je razviti strategiju rada robota. Strategija koju se razvija daje skup pravila. Svako od tih pravila za svaku situaciju koju Robby vidi preko svojih kamera, daje jednu od mogućih akcija koju Robby treba poduzeti. Dakle, strategija robota Robby nije niz od 200 akcija koje on treba poduzeti, već je niz ako-onda (preciznije rečeno, situacija-akcija) pravila npr. ako je trenutno polje prazno

i lijevo od tebe zid i iznad tebe zid i desno od tebe limenka i ispod tebe prazno polje, pomakni se jedno polje udesno.

Postavlja se sada pitanje koliko tih pravila uopće ima. Robby ima 5 kamera i na procjenu situacije utječe pogledom na pet polja. Na svakom od polja može se nalaziti ništa, limenka ili zid – dakle, tri su opcije za svako polje. Time se dolazi do toga da su na trenutnom polju moguće tri situacije i na svakom susjednom također po tri situacije. Što znači da ukupan broj pravila koja je potrebno pokriti iznosi $3 * 3 * 3 * 3 * 3 = 3^5 = 243$.

Zapravo, pravila ima nešto manje. Naime, budući da Robby ne može stajati na zidu, na trenutnom polju sigurno nije zid. Nadalje, budući da je mreža ograđena samo s vanjske strane, Robby se sigurno neće naći u situaciji da je sa sve četiri strane okružen zidom. Uz te, postoji i još nekoliko situacija koje su nemoguće, međutim budući da su one nemoguće, neće imati utjecaja na Robbyjev rad (jer se Robby nikad neće naći u takvoj situaciji), stoga se najčešće ostavlja 243 moguće situacije, bez obzira na to što do nekih nikad neće doći.

Za kraj, valja razmotriti i koliko je kombinacija pravila tada uopće moguće razviti. Za svaku od 243 moguće situacije, Robby može napraviti jednu od 7 mogućih akcija. Dakle, za prvu situaciju je moguće 7 akcija, za drugu 7, itd. Trivijalnim računom dolazi se do toga da je ukupno moguće razviti $7^{243} = 2{,}2847 * 10^{205}$ različitih kombinacija.

Kako bi se broj kombinacija sveo u razumljive mjere, neka je pretpostavka da se problem rješava iscrpnom (grubom) pretragom (engl. *brute force*) naprosto generirajući sve kombinacije. Ako evaluacija jednog takvog rješenja (kombinacije) traje svega $1\mu s$ (što je mnogo brže od onog koliko bi evaluacija trajala da se izvršava na najbržem modernom računalu), ukupno trajanje pretrage iznosilo bi okvirno $2{,}2847 * 10^{199}$ sekundi. Da se to svede na još razumljivije mjere dolazi se do okvirnog trajanja od $2{,}6443 * 10^{194}\ dana = 5{,}2465 * 10^{181}\ starosti\ svemira$[6]. S obzirom na iznos trajanja, trivijalno je zaključiti da problem u konačnom vremenu nije moguće riješiti iscrpnom pretragom.

4.2. Programsko rješenje

Kako bi se strategije mogle vrednovati (evaluirati) i time ih svesti na usporedive parametre, potrebno je odrediti funkciju evaluacije i odrediti nagrade i kazne. Za svaku pokupljenu limenku, Robby bi dobio 10 bodova nagrade. Ako bi se Robby sagnuo da pokupi limenku na mjestu na kojem nema limenke dobio bi 5 bodova

[6] Pod pojmom starost svemira u fizikalnoj kozmologiji se podrazumijeva vrijeme proteklo od Velikog praska što prema trenutnim mjerenjima unutar modela Lambda-CDM iznosi 13,799 ± 0,021 milijardi (10^9) godina. Pri izračunu je uzeta srednja vrijednost starosti svemira ($13{,}799 * 10^9$ godina). Za više detalja zainteresiranog čitatelja se upućuje na [15].

kazne (tj. -5 bodova). Ako bi se Robby sudario sa zidom, dobio bi -10 bodova i bio bi vraćen na polje na kojem je bio prije sudara sa zidom. U svim ostalim situacijama, Robby ne bi dobio nagradu niti kaznu (0 bodova). Vrijednosti nagrada i kazni se međusobno zbrajaju, pritom maksimalna dobrota iznosi $10 * broj\_limenki$ (gdje broj $broj\_limenki$ označava koliko je limenki raspoređeno na mreži), a najlošije moguće rješenje može imati dobrotu -2000 (u slučaju da se u svih 200 akcija zabije u zid). Svaka od strategija testirana je na 10 različitih razmještaja limenki (svjetova). Ukupna dobrota jednog rješenja je prosjek dobrote rješenja na svim svjetovima, tj.

$$\frac{\sum_{i=1}^{10} dobrota_i}{10} \tag{4.1}$$

gdje je dobrota rješenja na *i*-tom svijetu označena sa $dobrota_i$.
S ovakvom definicijom funkcije dobrote i opisom problema, zaključuje se da je cilj rješenja evoluirati tako da njihova dobrota bude što veća (uz gornju granicu od 350 bodova) - problem je maksimizacijski.

Programski su ostvareni generacijski GA, generacijski GA s elitizmom, eliminacijski GA i strategija SASEGASA. Za sve implementacije vrijede sljedeće postavke:

- Kao način prikaza rješenja odabran je prikaz poljem cijelih brojeva. Akcije su označene cijelim brojevima iz intervala [0,6]. Pojava brojeva izvan ovog intervala rezultiraju izvanrednim prekidanjem rada programa.
- Mogući sadržaji polja koje Robby vidi kamerama označeni su cijelim brojevima iz intervala [0,2] gdje 0 označava prazno polje, 1 limenku, a 2 zid.
- Inicijalna populacija rješenja generirana je slučajnim odabirom. Veličina populacije ovisi o korištenom algoritmu.
- Za selekciju roditelja implementirana je jednostavna troturnirska selekcija. Što se događa s odbačenim (najlošijim) roditeljem, ovisi o korištenom algoritmu (pojašnjeno kasnije).
- Za križanje roditelja implementirana su tri tipa križanja; diskretno, ravno i križanje s jednom točkom prekida. U svim implementacijama algoritama, sva tri križanja bit će odabrana za operaciju križanja s približno jednakom vjerojatnošću (~33,333%).
- Za mutiranje novonastalog rješenja djeteta implementirana je mutacija jednog gena koja slučajno odabrani gen postavlja na slučajno odabranu akciju iz skupa mogućih akcija.
- Sve implementacije algoritama testirane su na 10 svjetova. Broj svjetova se ne mijenja tijekom izvođenja.

- Kako se rješenja ne bi adaptirala na 10 inicijalno stvorenih svjetova, za svaki fiksni broj generacija ili nastale djece, stvaraju se novi svjetovi, a stari se odbacuju. Broj generacija ili nastale djece prije stvaranja novih svjetova i odbacivanja starih ovisi o korištenom algoritmu.

4.2.1. Specifični parametri korištenih algoritama

Svaki od implementiranih algoritama ima svoje specifične parametre pri izvođenju. Oni su postavljeni kako slijedi:

1. Generacijski GA i generacijski GA s elitizmom
    - Veličina populacije = 250 jedinki,
    - Maksimalni broj generacija = 40.000,
    - Minimalna željena dobrota = 340 (u trenutku kad se ona dosegne evolucija završava s radom),
    - U jednostavnoj troturnirskoj selekciji najlošiji roditelj se zanemaruje,
    - Skup svjetova za testiranje se izmjenjuje nakon svake generacije,
    - Kod elitističkog GA u iduću generaciju se smješta samo jedno, najbolje rješenje iz prethodne generacije.
2. Eliminacijski GA
    - Veličina populacije = 100 jedinki,
    - Maksimalni broj proizvedene djece = 10.000.000 (10 milijuna),
    - Minimalna željena dobrota = 340 (u trenutku kad se ona dosegne evolucija završava s radom),
    - Svjetovi se nanovo razmještaju nakon 1000 proizvedene djece,
    - U operatoru selekcije novostvoreno dijete se smješta na mjesto trećeg (najlošijeg) roditelja.
3. SASEGASA
    - Veličina populacije = 50 jedinki,
    - Broj potpopulacija = 10,
    - Minimalna željena dobrota nije definirana već se algoritam vrti sve dok se ne svede na jednu veliku populaciju,
    - Maksimalni broj generacija = 100.000 (služi i za brojanje generacija i za podizanje letvice prilikom vrednovanja uspješnosti djeteta)
    - Svjetovi se nanovo razmještaju nakon svake generacije u algoritmu selekcije potomaka,
    - Selekcija potomaka koja se vrši nad potpopulacijama je ograničena maksimalnim selekcijskim pritiskom i maksimalnim brojem iteracija.

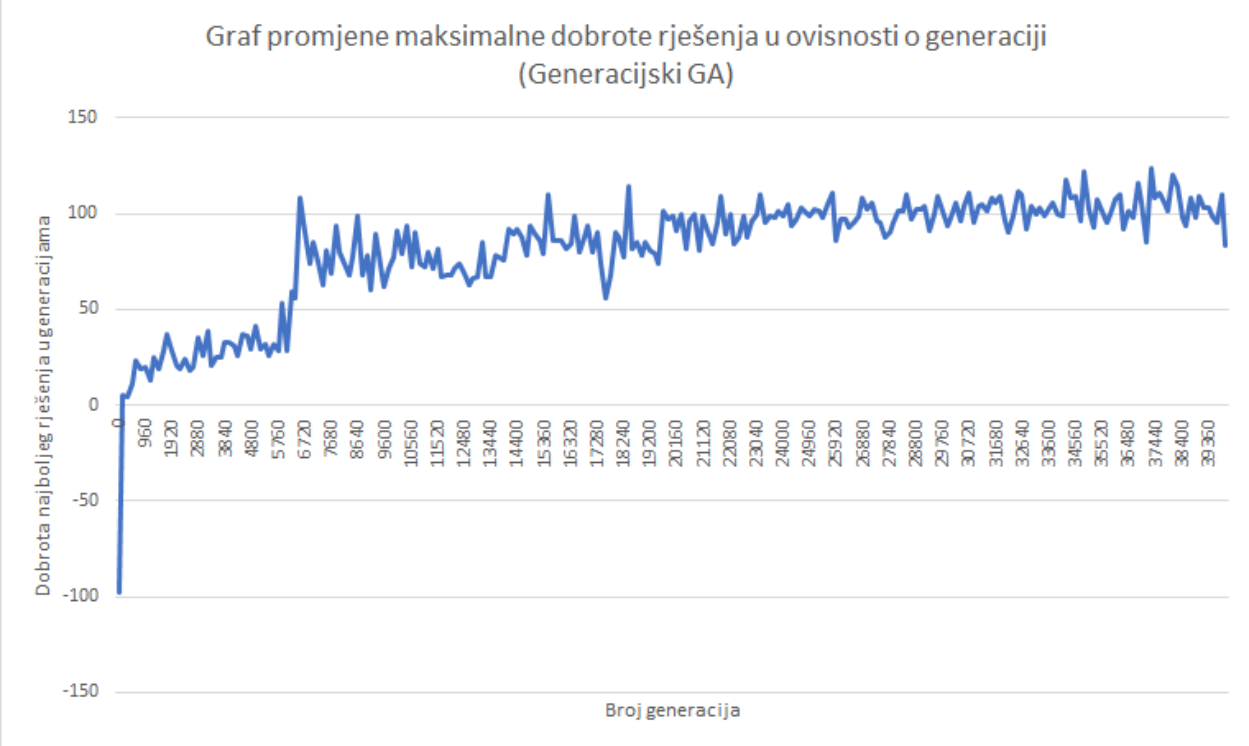
Programsko rješenje je implementirano koristeći programski jezik Java verzije 9. Grafičko korisničko sučelje za prikaz rješenja je implementirano koristeći biblioteku Java Swing koja je standardni dio programskog jezika Java.
Eksperimentalnim rezultatima i primjedbama na njih bavi se potpoglavlje 4.3.

4.3. Eksperimentalni rezultati

Eksperimentalni rezultati dobiveni su na prijenosnom računalu s procesorom Intel Core i7-6700HQ radnog takta 2,60 GHz s 8,00 GB radne memorije upogonjenim operacijskim sustavom *Windows 10 Education*. Programsko rješenje pisano je programskim jezikom Java (verzije 9) i upogonjeno na Oracle JVM virtualnom stroju. Valja još napomenuti da su dijagrami napravljeni koristeći *Microsoft Excel365*, te zbog njegovog ograničenja maksimalnog broja vrijednosti kao i zbog širine papira, grafovi nisu mogli biti preciznije prikazani.

4.3.1. Generacijski genetski algoritam

Generacijski GA je pokrenut s 250 jedinki u populaciji s maksimalnih 40.000 generacija. Rješenje se smatra zadovoljavajućim (uvjet prekida) ako je dobrota rješenja 340 bodova. Jedan od primjera promjene dobrote tijekom evolucije prikazan je na slici 4.2.



Slika 4.2. Graf promjene dobrote rješenja u ovisnosti o generaciji kod generacijskog GA

Primjer je pokrenut deset puta. Prethodni graf je iscrtan na temelju posljednje od deset evolucija. Rezultati svih pokretanja prikazani su u tablici 4.1.
Stupac „Broj potrebnih generacija“ označava koliko je generacija bilo potrebno do kraja izvođenja. Stupac „Najveća dobrota - kraj“ označava koja je najveća dobrota neke od jedinki zadnje generacije. Stupac „Najveća dobrota“ označava ukupno najveću nađenu dobrotu tijekom izvođenja. Stupac „Vrijeme izvođenja“ označava vremensko trajanje izvođenja programa, izraženo u sekundama i zaokruženo na jednu decimalu.

Tablica 4.1. Rezultati pokretanja implementacije generacijskog GA

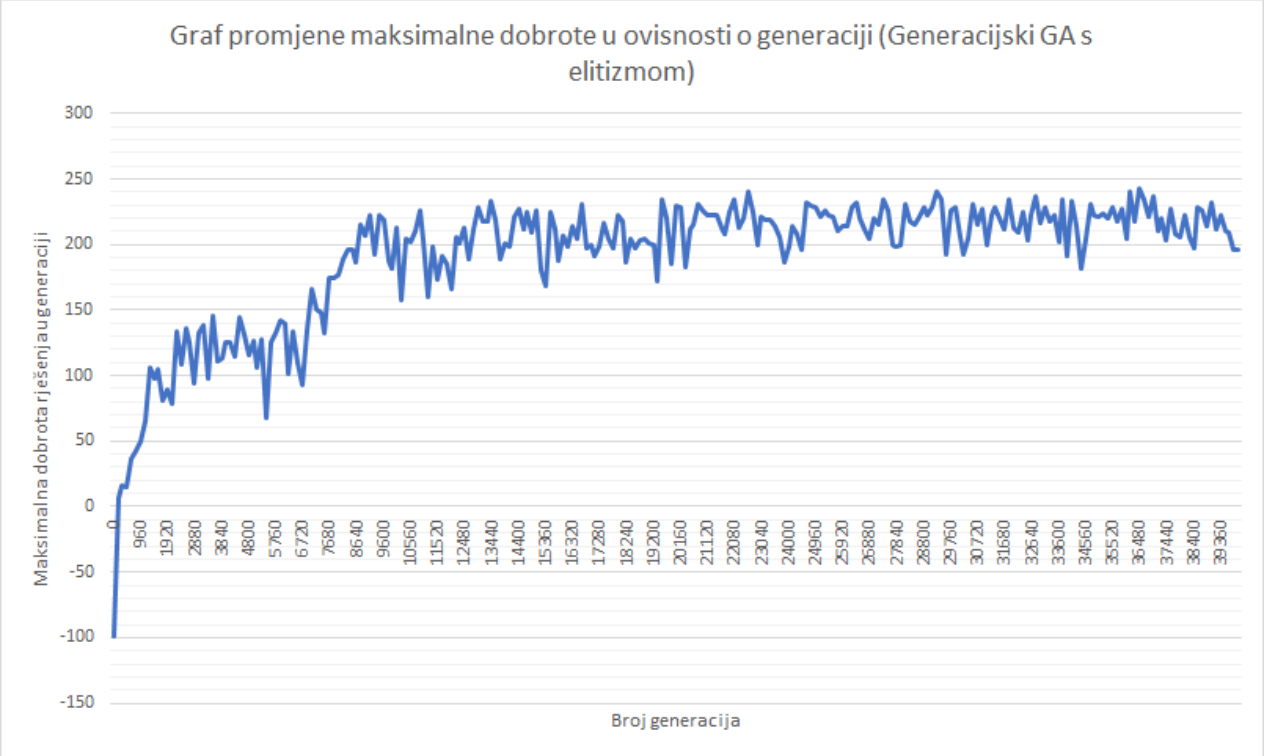
Redni broj pokretanja	Broj potrebnih generacija	Najveća dobrota - kraj	Najveća dobrota	Vrijeme izvođenja [s]
1.	40.000	20,0	41,0	306,7
2.	40.000	30,0	54,0	305,5
3.	40.000	38,0	52,0	347,3
4.	40.000	96,0	148,0	424,7
5.	40.000	35,0	55,0	295,8
6.	40.000	68,0	162,0	371,1
7.	40.000	72,0	110,0	430,4
8.	40.000	35,0	58,0	377,0
9.	40.000	40,0	52,0	376,3
10.	40.000	123,0	83,0	393,1

Budući da se većinu vremena izvođenja pojedine evolucije najbolja dobrota giba oko neke vrijednosti i s obzirom na to da je u svakoj evoluciji dosegnut maksimalan broj generacija, može se zaključiti da je došlo do preuranjene konvergencije.
S obzirom na veliku varijaciju između konačnih rješenja može se zaključiti da je najbolja dobrota rješenja na kraju uvelike uvjetovana početnom, nasumce generiranom populacijom rješenja.

Valja skrenuti pozornost da su nagle varijacije u dobroti u svakoj od evolucija uzrokovane dvama faktorima. Jednim dijelom je to uvjetovano time što se nakon svake generacije mijenjaju i svjetovi na kojima se rješenja evaluiraju. Drugim dijelom, to je uvjetovano i svojstvom generacijskog GA da prošla generacija izumire u trenutku kad nastane nova, a ta nova generacija može biti lošija od prethodne.

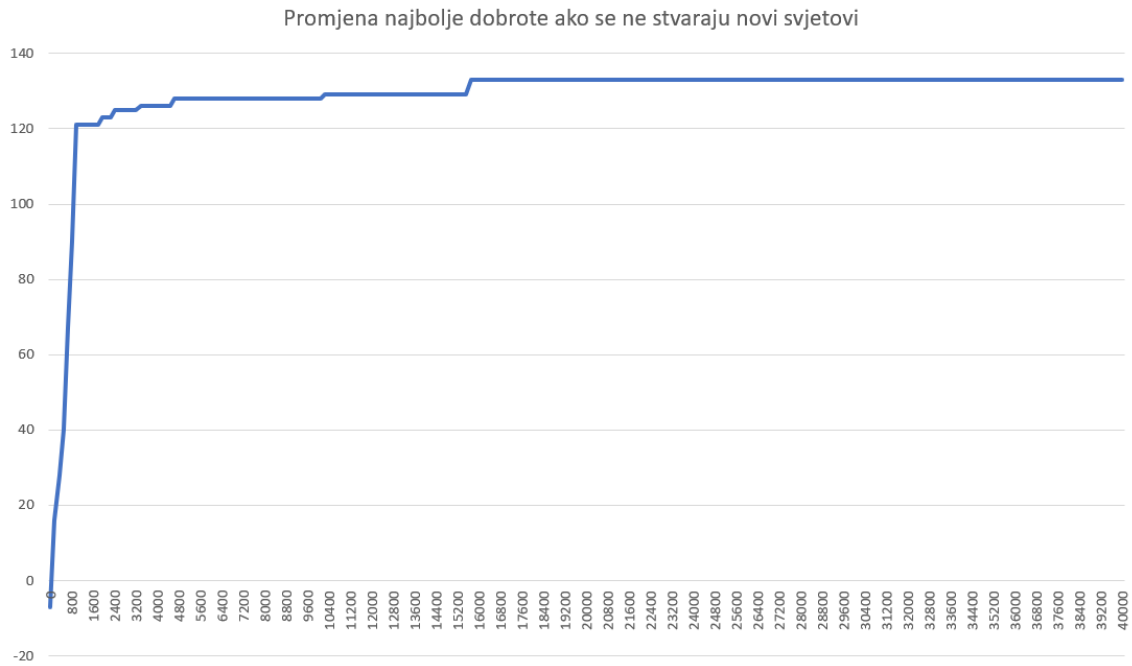
4.3.2. Generacijski genetski algoritam s elitizmom

Kod generacijskog GA s elitizmom parametri pokretanja su identični klasičnom generacijskom GA. Elitom se proglašava najbolja jedinka trenutne generacije i ona se smješta u iduću. Rezultati jednog eksperimenta prikazani su na slici 4.3.



Slika 4.3. Graf promjene dobrote rješenja u ovisnosti o generaciji kod generacijskog GA s elitizmom

Iako generacijski GA s elitizmom čuva najbolje rješenje iz prošle generacije, na grafu se javljaju pilasta nazubljenja. Razlog tomu je taj što se nakon svake generacije prethodnih 10 svjetova odbacuje i generira se novih 10 kako se rješenja ne bi adaptirala na inicijalno stvorene svjetove. Nakon generiranja novih 10 svjetova, čitava populacija se ponovno evaluira i može se dogoditi da rješenje koje je bilo najbolje na prethodnim svjetovima, na novim svjetovima više ne bude najbolje. Ako bi se taj uvjet uklonio, tada bi bio vidljiv progresivni rast dobrote najboljeg rješenja, kako je prikazano na slici 4.4.



Slika 4.4. Graf promjene dobrote rješenja u ovisnosti o generaciji kod generacijskog GA s elitizmom ako se rješenje evoluira na početno generiranim svjetovima

Primjer je pokrenut deset puta. Prethodni graf je iscrtan na temelju posljednje od deset evolucija. Rezultati svih pokretanja prikazani su u tablici 4.2.
Stupac „Broj potrebnih generacija“ označava koliko je generacija bilo potrebno do kraja izvođenja. Stupac „Najveća dobrota - kraj“ označava koja je najveća dobrota neke od jedinki zadnje generacije. Stupac „Najveća dobrota“ označava ukupno najveću nađenu dobrotu tijekom izvođenja. Stupac „Vrijeme izvođenja“ označava vremensko trajanje izvođenja programa, izraženo u sekundama i zaokruženo na jednu decimalu.

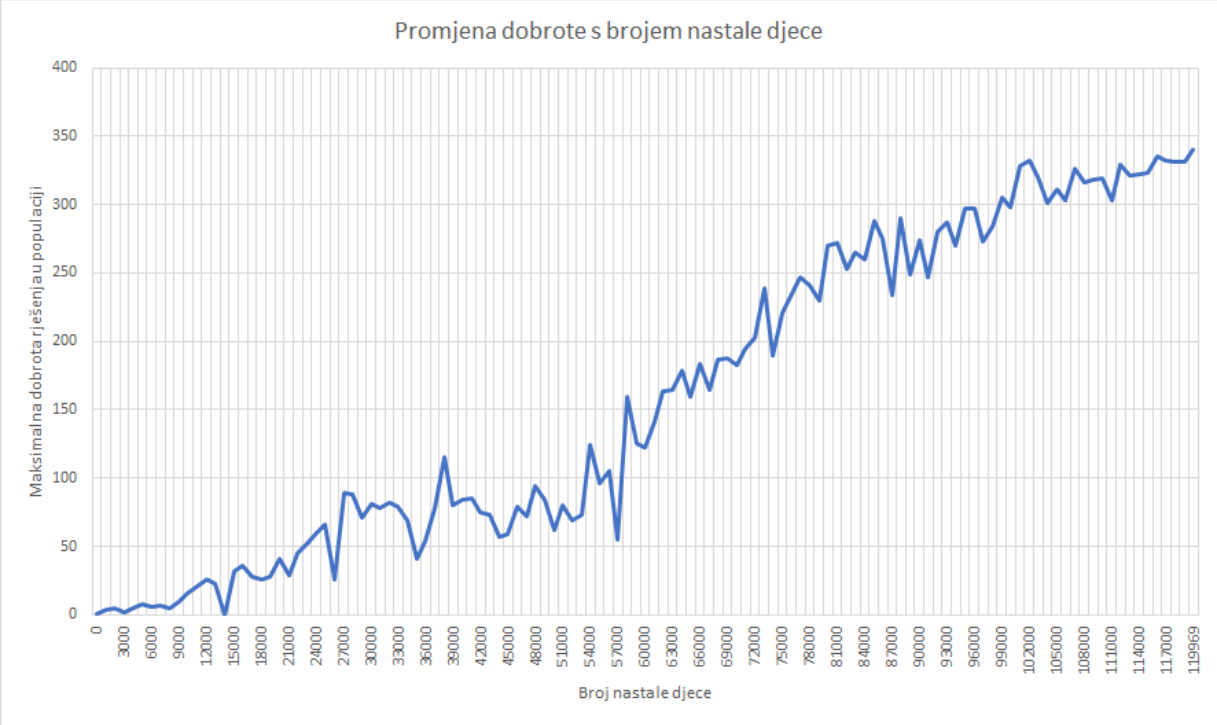
Tablica 4.2. Rezultati pokretanja implementacije generacijskog GA s elitizmom

Redni broj pokretanja	Broj potrebnih generacija	Najveća dobrota - kraj	Najveća dobrota	Vrijeme izvođenja [s]
1.	40.000	257,0	292,0	497,7
2.	40.000	35,0	52,0	424,2
3.	40.000	31,0	54,0	372,6
4.	40.000	44,0	54,0	363,3
5.	40.000	33,0	54,0	404,9
6.	40.000	132,0	177,0	408,3
7.	40.000	35,0	55,0	381,0
8.	40.000	172,0	247,0	368,7
9.	40.000	99,0	186,0	332,7
10	40.000	196,0	242,0	402,7

Analiza preuranjene konvergencije je analogna generacijskom GA. Unatoč tome, za razliku od generacijskog GA, varijacije u dobroti rješenja tijekom pojedine evolucije su nešto manja i ona su uvjetovana time što se nakon svake generacije mijenjaju svjetovi na kojima se rješenja evaluiraju.
Iako generacijski GA s elitizmom u prosjeku daje nešto bolja rješenja u odnosu na klasični generacijski GA, u dobrom dijelu slučajeva razlika je minimalna. Razlog tome je to što tada postoji mnogo relativno loših rješenja tako da se najbolje rješenje utopi među njima.

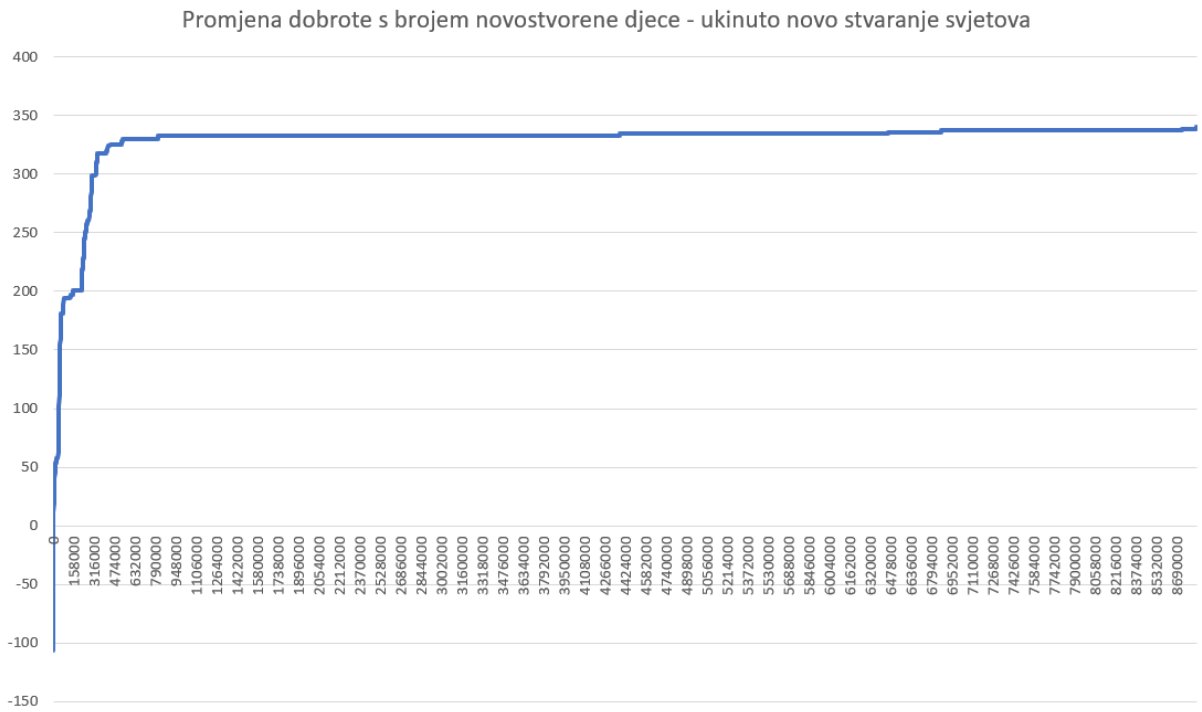
4.3.3. Eliminacijski genetski algoritam

Za razliku od prethodne dvije izvedbe GA, kod eliminacijskog nije došlo do preuranjene konvergencije. Tijek promjene dobrote u odnosu na broj novonastale djece prikazan je na slici 4.5.



Slika 4.5. Graf promjene dobrote rješenja u ovisnosti o broju nastale djece kod eliminacijskog GA

Ovdje se također, kao i kod elitističkog GA primjećuje pilasta nazubljenost koja je također posljedica generiranja novih svjetova nakon svakih 1000 novostvorene djece. Tijek evolucije kad bi se taj uvjet ukinuo prikazan je na slici 4.6.



Slika 4.6. Graf promjene dobrote rješenja u ovisnosti o broju nastale djece kod eliminacijskog GA ako se ukine novo stvaranje svjetova

Primjer je pokrenut deset puta. Prethodni graf je iscrtan na temelju posljednje od deset evolucija. Rezultati svih pokretanja prikazani su u tablici 4.3.
Stupac „Broj potrebnih operacija“ označava koliko je djece bilo potrebno nanovo stvoriti do kraja izvođenja. Stupac „Najveća dobrota“ označava ukupno najveću nađenu dobrotu tijekom izvođenja te ujedno i najveću dobrotu na kraju izvođenja jer se uvjet za prekid provjerava nakon svakog novostvorenog djeteta. Stupac „Vrijeme izvođenja“ označava vremensko trajanje izvođenja programa, izraženo u sekundama i zaokruženo na jednu decimalu.

Tablica 4.3. Rezultati pokretanja implementacije eliminacijskog GA

Redni broj pokretanja	Broj potrebnih operacija	Najveća dobrota	Vrijeme izvođenja [s]
1.	526142	344,0	29,1
2.	240942	341,0	14,5
3.	113128	340,0	5,8
4.	2090293	341,0	120,0
5.	525022	340,0	25,7
6.	549000	344,0	25,6
7.	897970	340,0	44,1
8.	81784	340,0	2,8
9.	109748	340,0	4,5

10	119969	341,0	6,2

Kao što je iz rezultata vidljivo, u deset pokretanja algoritma niti jednom nije došlo do preuranjene konvergencije. Nadalje, svih deset eksperimenata završilo je u prilično kratkom vremenu.

4.3.4. Samoadaptivni segregacijski genetski algoritam s aspektima simuliranog kaljenja (SASEGASA)

Strategija SASEGASA je pokrenuta s 50 jedinki po potpopulaciji i 10 potpopulacija. Donja granica prihvatljive dobrote nije propisana već je algoritam implementiran prema kanonskoj definiciji. Propisano je 100.000 iteracija/generacija. Algoritam selekcije potomaka kao uvjet prekida koristi dosezanje maksimalnog selekcijskog pritiska i maksimalni broj generacija.
Zbog složenosti grafa i velikog broja potrebnih parametara, tijek evolucije neće biti prikazan grafički.

Primjer je pokrenut deset puta. Rezultati svih pokretanja prikazani su u tablici 4.4. Stupac „Najveća dobrota“ označava ukupno najveću nađenu dobrotu tijekom izvođenja. Stupac „Vrijeme izvođenja“ označava koliko je vremenski trajalo izvođenje programa, izraženo u formatu minute:sekunde i zaokruženo na najbližu cijelu sekundu.

Tablica 4.4. Rezultati pokretanja implementacije strategije SASEGASA

Redni broj pokretanja	Najveća dobrota	Vrijeme izvođenja [mm:ss]
1.	348,0	61:37
2.	349,0	25:43
3.	344,0	42:23
4.	345,0	75:46
5.	347,0	55:42
6.	311,0	105:37
7.	348,0	69:22
8.	341,0	40:17
9.	329,0	122:12
10	340,0	90:53

Kao što je iz tablice vidljivo, od deset pokretanja dva nisu dosegla razinu koja bi se smatrala prihvatljivim rješenjem (tj. dobrotu od 340 bodova). Preostalih osam rješenja je u prosjeku bliže globalnom optimumu od rješenja dobivenih eliminacijskih GA. Unatoč tome, robusnost kod SASEGASA strategije ima i svoju cijenu – najkraća

optimizacija trajala je čak 25 minuta i 43 sekunde, dok je najdulja trajala više od dva puna sata.

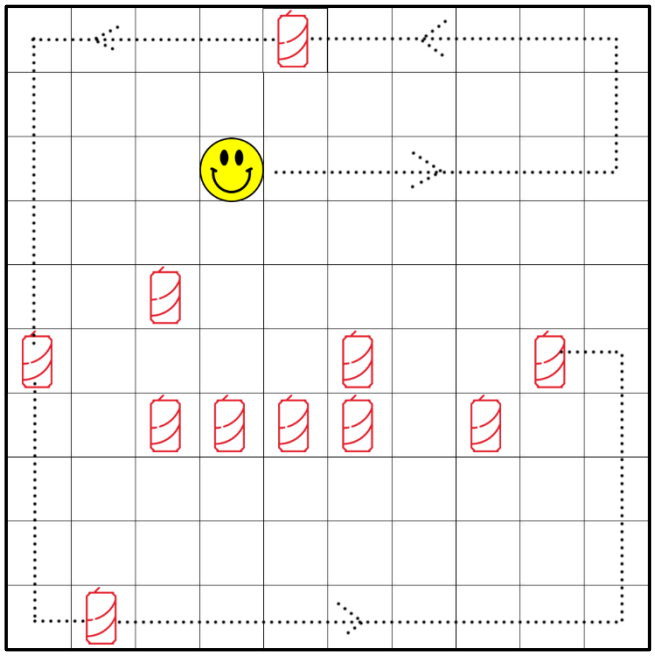
4.4. Primjedbe i zaključci

Kao što je izloženo u prethodnom poglavlju, problem preuranjene konvergencije je čest kod GA. Na njemu su pali generacijski i elitistički GA. Unatoč tome, eliminacijski GA se pokazao kao prilično imun na preuranjenu konvergenciju i sposoban dati kvalitetna rješenja u prilično kratkom vremenskom roku. Nadalje, strategija usporavanja preuranjene konvergencije SASEGASA je u 8 od 10 pokretanja omogućila uspješno izbjegavanje preuranjene konvergencije. Iako bolja od generacijskog i elitističkog GA, ta robusnost po pitanju izbjegavanja preuranjene konvergencije ima i svoju cijenu – strategija SASEGASA se pokazala kao vremenski i prema broju operacija koje treba izvršiti vrlo skupa te stoga nije idealna za problem kojeg se u ovom radu izlaže.

Znače li ovi podaci da je strategija SASEGASA loša i da je bolje koristiti eliminacijski GA? Naravno da ne. Strategija za ovaj problem nije dala povoljnija rješenja u odnosu na eliminacijski GA, ali to ne znači da je eliminacijski GA superiorniji za svaki problem kojeg je potrebno riješiti te će se možda za neke druge probleme generacijski GA i SASEGASA pokazati boljima i otpornijima na preuranjenu konvergenciju u odnosu na eliminacijski GA.

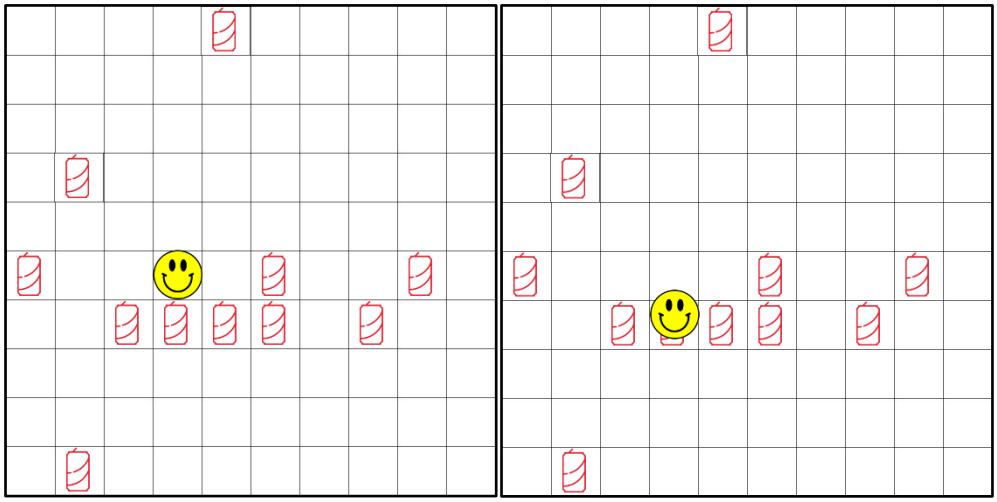
Ta je tvrdnja dokazana u sklopu „*No free lunch*“ teorema koji matematički dokazuje da ne postoji općenito najbolji optimizacijski algoritam već da za svaki od algoritama postoje problemi koji će se najbolje riješiti upravo tim algoritmom. „*All algorithms that search for an extremum of a cost function perform exactly the same, according to any performance measure, when averaged over all possible cost functions. In particular, if algorithm A outperforms algorithm B on some cost functions, then loosely speaking there must exist exactly as many other functions where B outperforms A*“. Citat je preuzet iz [18]. Za više informacija čitatelja se upućuje na [18, 19].

Valja se još osvrnuti i na dvije zanimljive primjedbe po pitanju toga što je GA evoluirao. Prvo, kada Robby ne vidi limenke za pokupiti u blizini – on neće odabrati slučajne kretnje već će se kružno kretati po rubu svog svijeta (pazeći da pritom ne udari u zid), bilo u smjeru kazaljke na satu ili suprotno. Primjer takvih kretnji prikazan je na slici 4.7. Limenke koje su se u vom primjeru našle na rubu ne igraju nikakvu ulogu te Robby na taj način pretražuje i onda kad ih nema.

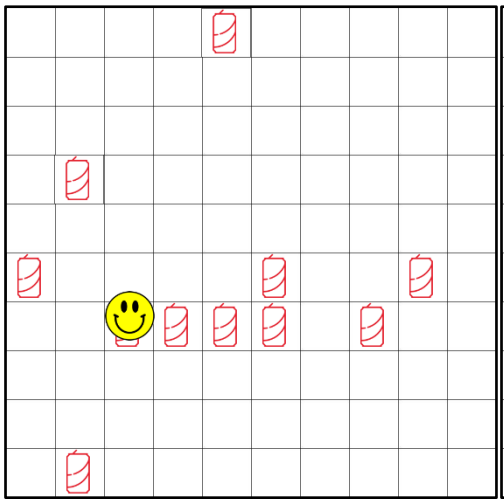


Slika 4.7. Primjer kretnji Robby-ja kad na okolnim poljima ne vidi limenke

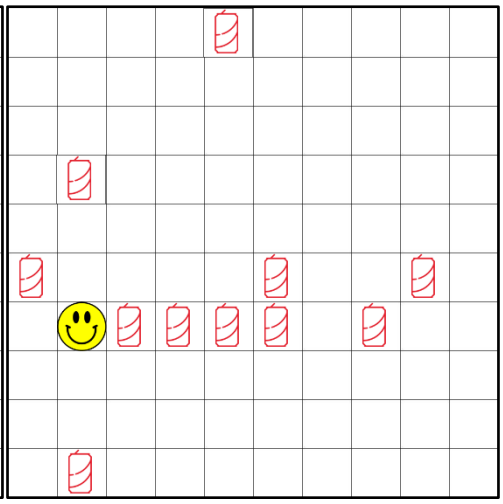
Druga zanimljiva primjedba je ta da Robby, kad naiđe na niz limenki, neće pokupiti prvu i nastaviti dalje sa sakupljanjem. Umjesto toga, on će pronaći početak tog niza limenki i tek kad ga pronađe početi kupiti. Primjer takvog ponašanja prikazan je na nizu slika 4.8.



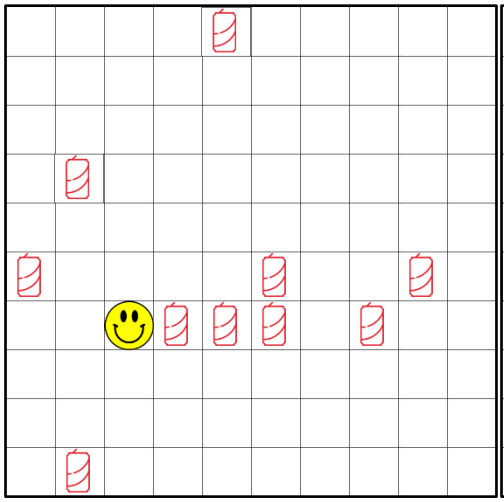
a) b)



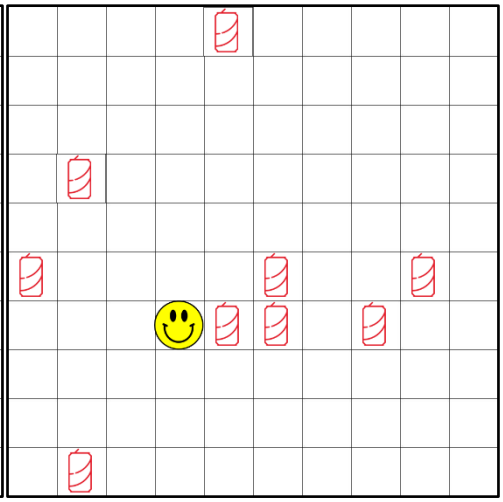
c)



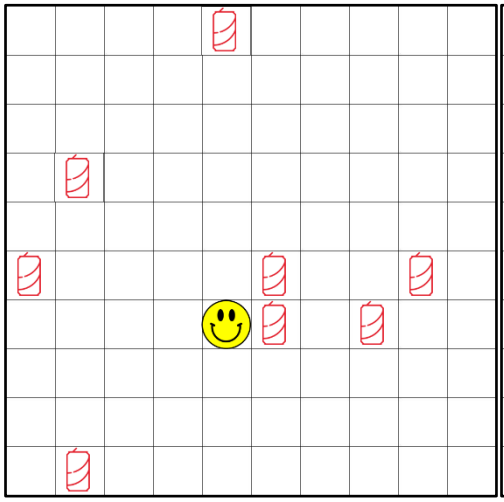
d)



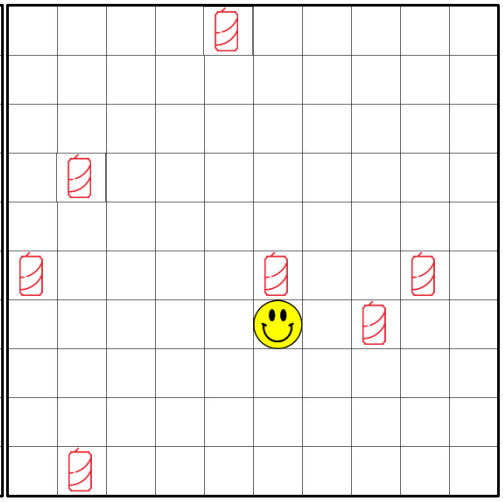
e)



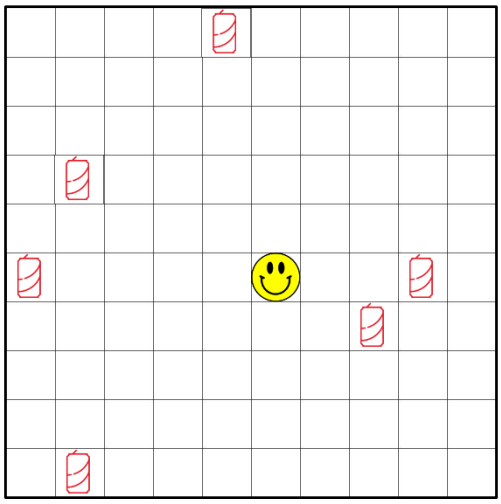
f)



g)



h)



i)

Slika 4.8. Primjer ponašanja robota Robby kad naiđe na niz (engl. *cluster*) boca

4.5. Potencijalna poboljšanja postupka optimizacije

Svi spomenuti algoritmi implementirani su u slijednoj verziji te se mogu i ubrzati paralelizacijom izvođenja postupaka optimizacija. Kod klasičnog GA to je moguće tako da se paralelizira evaluacija ili pak čitav postupak selekcije, križanja i mutiranja čime se dobiva tzv. otočni model paralelizacije.

SASEGASA je sam po sebi zamišljen kao paralelni tip GA. Kod te strategije prilično poboljšanje performansi izvođenja donesla bi paralelizacija algoritma odabira potomaka (engl. *offspring selection*) tako da svaka dretva vrti taj algoritam nad jednom potpopulacijom.

Za sve paralelizacije generalno vrijedi pravilo da se one ne smiju sinkronizirati, budući da bi tada performanse algoritama približno bile jednake kao da nema paralelizacije. Međutim, ako se paralelizacija bez sinkronizacije radi nad jednom populacijom, tada se može dogoditi da jedna dretva prepiše preko memorijskog prostora u koji je druga dretva već nešto zapisala, ali u takvom slučaju beskoristan rad koji se također očituje manjim gubitkom performansi drastično manje utječe na brzinu rada algoritma u odnosu na rad sa sinkronizacijama. Za više detalja o tome, zainteresiranog čitatelja se upućuje na [16].

Nadalje, umjesto stvaranja velikog broja dretvi čijim se stvaranjem operacijskom sustavu daje mnogo dodatnog posla, bolje je rješenje koristiti bazen dretvi fiksne veličine. Ideja bazena dretvi jest da se stvori fiksni broj dretvi i sve stave u bazen. U trenutku kad dođe posao, dretve iz bazena su već stvorene i spremne preuzeti taj posao, te se nakon dovršetka posla vraćaju natrag u bazen.
Za kraj, postavlja se pitanje koliko dretvi stvoriti. Ako se stvori premalo dretvi tada se u pogledu vremena izvođenja algoritma neće puno osjetiti vremensko poboljšanje.

Stvaranje previše dretvi, s druge strane, može još više usporiti izvođenje programa budući da će procesor morati balansirati između velikog broja dretvi konstantno radeći „kućanske poslove“ pohrane konteksta, spremanja konteksta i dr[7]. Pokazuje se da je optimalna mjera broja dretvi jednaka broju jezgri procesora računala na kojem se algoritam vrti. Ta se informacija u programskom jeziku Java dobiva naredbom `Runtime.getRuntime().availableProcessors()`. Tada će se svaka od dretvi moći smjestiti na jednu od jezgri procesora i gotovo bez odvraćanja drugim poslovima, svaka će dretva moći raditi svoj posao.

[7] Za više informacija, čitatelja se upućuje na [17]

5. Zaključak

U ovom radu izloženi su rezultati primjene tri tipa genetskog algoritma i jedne strategije usporavanja preuranjene konvergencije na problemu učenja robota Robby. Generacijski GA i GA s elitizmom su se pokazali vrlo ranjivi na problem preuranjene konvergencije. Eliminacijski GA se pokazao vrlo imun na preuranjenu konvergenciju, te je pritom dao rješenje problema u prilično kratkom vremenskom roku. Strategija SASEGASA se pokazala prilično imunom na problem preuranjene konvergencije završivši u njoj dva od ukupno deset pokretanja. Usto robusnost takve strategije ima i svoju cijenu u pogledu vremena izvođenja budući da se ona izvršavala višestruko u dulje od eliminacijskog GA. Upravo iz tog razloga može se zaključiti da strategija SASEGASA nije idealna za rješavanje problema kojeg se u radu izlaže.

Iako je učenje robota Robby jedan „školski" primjer korištenja optimizacije koristeći genetski algoritam, njegova složenost je na razini složenih problema na kojima se GA primjenjuje i u stvarnosti. Ovakav primjer, iako je „školski", daje uvida u to da GA neke stvari „vidi" i pretražuje drugačije nego čovjek, a najbolji primjer toga su kretnje Robbyja po rubu svijeta kad u njegovoj okolini nema limenki i način na koji on skuplja niz limenki kad naiđe na njega. Kako ovdje, tako i u stvarnom svijetu, GA će često razviti rješenja koja rade, ali teško je vidjeti zašto ona rade. Nekadašnji NASA-in stručnjak za genetske algoritme Jason Lohn naglašava ovu pojavu: „*Evolutionary algorithms are a great tool for exploring the dark corners of design space. You show [your designs] to people with 25 years' experience in the industry and they say 'Wow, does that really work?' [...] We frequently see evolved designs that are completely unintelligible*."[8]
I doista, ta rješenja rade – Lohn i njegov tim stručnjaka su 2005. dobili nagradu „*Human Competitive*" za njihov dizajn nove antene za NASA-ine letjelice – dizajn koji je dobiven upravo njihovim genetskim algoritmom[9].

[8] Citat preuzet iz [2]
[9] Za više informacija, znatiželjnog čitatelja se upućuje na [20]

Korištena literatura

[1] M. Čupić: *Prirodom inspirirani optimizacijski algoritmi. Metaheuristike.* (radna inačica udžbenika). Zagreb, 2013.

[2] M. Mitchell: *Complexity: a guided tour*. New York: Oxford University Press, 2009.

[3] L. J. Fogel, A. J. Owens, M. J. Walsh. *Artificial Intelligence through Simulated Evolution.* New York: John Wiley, 1966.

[4] I. Rechenberg: *Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart: Frommann-Holzboog, 1973.

[5] R. Fisher: *The Genetical Theory of Natural Selection, 1st edition.* Oxford: The Clarendon Press, 1930.

[6] C. Darwin: *The Origin of Species*. London: John Murray, 1859.

[7] J. H. Holland: *Adaption in Natural and Artifical Systems*. Michigan: University of Michigan Press, 1975.

[8] B. Borković, I. Šubek, T. Fiolić, A. Ivanković: *Posavski konj, naslijeđe za budućnost*. Konferencija o izvornim pasminama i sortama kao dijelu prirodne i kulturne baštine, Šibenik, (2007.), str. 33 – 34

[9] M. Steinhausz: *Uzgoj konja u Posavini Savske Banovine*. Zagreb: Izdanje kr. banske uprave Savske banovine u Zagrebu, 1934.

[10] M. Affenzeller, S. Winkler, S. Wagner, A. Beham: *Genetic algorithms and Genetic programming. Modern Concepts and Practical Applications*. Boca Raton: CRC Press, 2009.

[11] E. Schöneburg, F. Heinzmann, and S. Feddersen: *Genetische Algorithmen und Evolutionsstrategien*. Addison-Wesley, 1994.

[12] M. Srinivas and L. Patnaik: *Adaptive probabilities of crossover and mutation in genetic algorithms.* Objavljeno u: IEEE Transactions on Systems, Man, and Cybernetics, volume 24 (1994.), str. 656–667

[13] M. Affenzeller, S. Wagner: *SASEGASA: A New Generic Parallel Evolutionary Algorithm for Achieving Highest Quality Results.* Journal of Heuristics, Volume 10, Issue 3 (May 2004.), str. 243 – 267

[14] D. L. Hartl, A. G. Clark: *Principles of Population Genetics, 2nd edition*. Massachusetts: Sinauer Associates Inc., 1989.

[15] *Starost Svemira,* datum nastanka: 7.7.2018., https://hr.wikipedia.org/wiki/Starost_Svemira, datum pristupanja: 28.5.2019.

[16] M. Golub: *Poboljšavanje djelotvornosti paralelnih genetskih algoritama,* doktorska disertacija. Zagreb: Fakultet elektrotehnike i računarstva, 2001.

[17] L. Budin, M. Golub, D. Jakobović, L. Jelenković: *Operacijskli sustavi*. Zagreb: Element Zagreb, 2013.

[18] D. H. Wolpert, W. G. Macready: No free lunch theorems for search. Technical report, Santa Fe Institute, Sante Fe, NM, USA, 1995. Technical Report SFI-TR-95-02-010.

[19] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. IEEE Transactions On Evolutionary Computation, 1(1):67_82, 1997.

[20] J. D. Lohn, G. S. Hornby, D. S. Linden: *An Evolved Antenna for Deployment on Nasa's Space Technology 5 Mission.* Objavljeno u: Genetic Programming Theory and Practice II. Springer Science+Business Media, 2005., str. 301 - 315

Primjena samoadaptivnog segregacijskog genetskog algoritma s aspektima simuliranog kaljenja na primjeru učenja robota Robby

Sažetak

Problem učenja robota Robby je problem čiji je cilj demonstrirati rad genetskog algoritma. U ovom radu problem se rješava koristeći generacijski genetski algoritam, elitistički genetski algoritam, eliminacijski genetski algoritam i strategiju usporavanja preuranjene konvergencije samoadaptivni segregacijski genetski algoritam s aspektima simuliranog kaljenja. Spomenuti algoritmi su opisani, izrađena je programska implementacija opisanih algoritama, izloženi su eksperimentalni rezultati te dani zaključci i primjedbe na temelju eksperimentalnih rezultata.

Ključne riječi: genetski algoritam (GA), učenje robota Robby, preuranjena konvergencija, usporavanje preuranjene konvergencije

Application of Self-Adaptive Segregative Genetic Algorithm with Simulated Annealing Aspects to Learning of Robby the Robot

Abstract

Robby the soda-can-collecting robot is problem designed to demonstrate the application of genetic algorithm. In this paper, it's solved using generation genetic algorithm, generation genetic algorithm with elitism, elimination genetic algorithm and strategy for retarding premature convergence called Self-Adaptive Segregative Genetic Algorithm with Simulated Annealing Aspects. Mentioned algorithms are described and implemented in programming language and experimental results, conclusions and observations are given.

Key words: genetic algorithm (GA), Robby the Robot, premature convergence, retarding premature convergence