SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Evolucijski algoritmi

Juraj Petrović

Voditelj: *Domagoj Jakobović*

Zagreb, svibanj, 2007.

Sadržaj:

1. UVOD ... 3

2. GENETSKI ALGORITMI ... 4

Uvod ... 4
Rješavanje problema pomoću genetskih algoritama ... 4
Predstavljanje rješenja ... 5
Postupci selekcije ... 6
Genetski operatori ... 8
Križanje ... 8
Mutacija ... 9
Primjer – Problem trgovačkog putnika (TSP) ... 10

3. GENETSKO PROGRAMIRANJE ... 12

Uvod ... 12
Sličnosti i razlike genetskog programiranja i genetskih algoritama ... 12
Parametri genetskog programa ... 13
Primjer genetskog programa ... 14

4. EVOLUCIJSKE STRATEGIJE ... 16

Općenito o evolucijskim strategijama ... 16

5. EVOLUCIJSKO PROGRAMIRANJE ... 18

Općenito o evolucijskom programiranju ... 18

6. ZAKLJUČAK ... 19

7. POPIS LITERATURE ... 20

8. SAŽETAK ... 21

1.Uvod

Evolucijski algoritmi definiraju se kao postupci optimiranja, učenja i modeliranja, koji se temelje na mehanizmu evolucije u prirodi. Njihov nastanak uzrokovali su prvenstveno pokušaji primjene načela evolucije u prirodi pri rješavanju nekih problema, ali i nastojanja za boljim razumijevanjem same evolucije u prirodi.

Prve ideje o mogućoj primjeni ovakvih modela javljaju se još pedesetih godina 20. stoljeća, ali zbog skromnih mogućnosti tadašnjih računala ostaju nerealizirane i uglavnom nepoznate široj znanstvenoj javnosti. Šezdesetih godina neovisno su razvijena tri postupka zasnovana na načelima evolucije u prirodi: evolucijsko programiranje (L. J. Fogel), evolucijske strategije (Rechenberg, Schwefel) i genetski algoritmi (J. H. Holland). Nešto kasnije počinje i razvoj genetskog programiranja (J. Koza). Evolucijski algoritmi spadaju u šire područje znanosti o spoznaji (*cognitive science*), a uže u područje inteligentnih algoritama (*computational intelligence*) [11].

Ovaj seminarski rad nastoji proanalizirati sve tipove evolucijskih algoritama, ali prvenstveno genetske algoritme i genetsko programiranje, te u dva primjera demonstrirati njihovu uspješnu primjenu.

2.Genetski algoritmi

Uvod

Genetski algoritmi čine jedan je od četiri tipa evolucijskih algoritama. To je heuristička (pronalazačka, otkrivačka) metoda optimiranja, koja imitira proces evolucije u prirodi.

Takvi algoritmi predloženi su prvi put još u ranim sedamdesetim godinama 20. stoljeća od strane Amerikanca Johna Henryja Hollanda. On je prvi pretpostavio postojanje snažne povezanosti svijeta računala i prirode, te je iznio ideju kako bi se i računala mogla prilagođavati okolišu, tj. zadatku ili problemu na sličan način kako to čine i životinje u prirodi.

Evolucija u prirodi (bar na prvi pogled) dosta je jasna; u nekoj populaciji životinjskih jedinki kroz dulji period vremena održali su se oni geni koji su bili odgovorni za neke pozitivne osobine. Do toga je došlo upravo zato jer su ti geni omogućavali lakše preživljavanje, a time i veću vjerojatnost parenja i prijenosa istoga genetskog materijala na potomstvo. Na analogan je način Holland definirao genetske algoritme. Prema njegovim riječima, genetski algoritam je metoda analize koja se temelji na Darwinovoj teoriji evolucije. On započinje sa nekim skupom jedinki koje posjeduju slučajno generirana svojstva. Za svaku od jedinki pomoću neke metode možemo procijeniti koliko su nam njena svojstva poželjna. One jedinke koje imaju poželjnija svojstva određenim metodama kombiniramo nastojeći im time dodatno poboljšati svojstva.

Genetski algoritmi su iterativni, što znači da im se izvođenje sastoji od nekog broja ponavljanja, pri čemu se u svakoj iteraciji dolazi sve bliže i bliže poželjnom rješenju. Pri svakoj iteraciji promatra se trenutno postojeći skup jedinki (generacija). Nad jedinkama se tada primjenjuju genetski operatori i metode selekcije pomoću čega se stvara nova generacija.

Najčešću primjenu genetski algoritmi pronalaze na području neuronskih mreža, pri problemima traženju najkraćeg puta, kod problema trgovačkog putnika, problema transporta i raspoređivanja procesa, optimiranja upita nad bazom podataka itd.

U NASA-i su genetski algoritmi korišteni npr. pri konstrukciji antena. Raspored elemenata kod antene Yugi-Oda (popularno zvana «riblja kost») dobiven genetskim algoritmom omogućio je bar 7.8% bolju sposobnost prihvata signala. Slično je bilo i sa UHF antenom svemirske letjelice Mars Odyssey; inžinjeri su pri konstrukciji previdjeli da bi solarne ploče uz antenu mogle utjecati na njene performanse. Genetskim algoritmom dobivena je nova antena, funkcionalna kao što je u početku bilo planirano [2].

Rješavanje problema pomoću genetskih algoritama

Rješavanju određenog problema pomoću genetskog algoritma možemo pristupiti na dva načina: ili prilagođavanjem problema genetskom algoritmu ili prilagođavanjem genetskog algoritma specifičnosti problema [1]. Prilagođavanje genetskog algoritma problemu znači da moramo modificirati njegov rad tako da radi s veličinama svojstvenim za određeni zadatak. To se najčešće odnosi na definiranje drugačijih struktura podataka i operatora. S obzirom na to da to nije nimalo jednostavan zadatak, često se pribjegava i drugom rješenju – pravljenju genetskog algoritma koji je

moguće primjeniti na neku čitavu klasu problema (takvi genetski algoritmi nazivaju se i evolucijski programi).

Potencijalna rješenja našeg problema predstavljena su populacijom jedinki koju promatramo u određenoj iteraciji genetskog algoritma. Ta su rješenja u računalu predstavljena preko neke podatkovne strukture i nazivaju se *kromosomi*, a mjera u kojoj su nam ta rješenja prihvatljiva naziva se *dobrota* i računa se *funkcijom dobrote* [1].

Na početku genetskog algoritma generira se početna populacija jedinki i to najčešće slučajnim odabirom rješenja iz domene. Početna populacija može se sastojati i od identičnih jedinki, ali je taj postupak manje učinkovit, ili kao početno rješenje možemo staviti optimalno rješenje dobiveno nekom drugom metodom. Nakon toga slijedi proces koji se ponavlja određeni broj puta, određeno vrijeme ili dok se ne zadovolji neki uvjet. Proces se sastoji od djelovanja genetskih operatora (selekcija, križanje i mutacija) nad populacijom jedinki.

Uobičajeno je da se tijekom evolucije održava stalna veličina populacije, a budući da je trajanje izvođenja pojedine iteracije približno konstantno, time je zadano i vrijeme izvođenja. Osim veličine populacije i vremena izvođenja algoritma na početku se zadaje i vjerojatnost mutacije i eventualno vjerojatnost križanja. Nerijetko se ti parametri tokom izvođenja algoritma mijenjaju, a moguće je i kreirati posebni genetski algoritam, koji se izvršava usporedno s glavnim i služi za određivanje vjerojatnosti križanja i mutacije. U ovisnosti o svim navedenim parametrima algoritam će brže ili sporije davati bolja ili lošija rješenja.

Predstavljanje rješenja

Podatkovna struktura preko koje se u računalu predstavlja potencijalno rješenje problema naziva se, dakle, *kromosom*. Kako podatke spremamo u kromosom ovisno je o tome kakvim problemom se bavimo. Na primjer, za problem najkraćeg puta kromosom možemo formirati kao polje cijelih brojeva, gdje svaki broj označava redni broj mjesta, a njihov redosljed predstavlja rješenje dolaska od početne točke do cilja. U praksi se često koristi i prikaz rješenja pomoću prirodnoga binarnog koda ili Grayevog koda [1]. Važno je pritom da kromosom predstavlja moguće rješenje zadanog problema, te da su genetski operatori definirani na način da ne stvaraju nove kromosome koji predstavljaju nemoguća rješenja, jer se time bespotrebno smanjuje učinkovitost algoritma. U nastavku slijedi prikaz kromosoma korištenjem prirodnog binarnog koda (slika 2.1), prikaz kromosoma zapisanog kao permutacija (slika 2.2), te prikaz kromosoma kao vektora s realnim komponentama (slika 2.3):

Kromosom 1	0110100101110

Slika 2.1: Zapis kromosoma prirodnim binarnim kodom

Kromosom 2	1 3 2 6 4 5

Slika 2.2: Zapis kromosoma kao permutacije

Kromosom 3	1.23 3.75 3.83 9.33 4.45

Slika 2.3: Zapis kromosoma kao vektor s realnim komponentama

Primjer problema u kojem ćemo koristiti kromosome u obliku binarnog koda je *problem ruksaka* (knapsack problem). U tom problemu zadani su nam broj predmeta, njihova veličina i vrijednosti. Cilj nam je ograničeni kapacitet ruksaka napuniti tako da je vrijednost predmeta u njemu najveća moguća. Nula na nekom mjestu u kromosomu tada bi značila da neki predmet nije u ruksaku, a jedinica da jest. Zapis kromosoma kao permutacije koristi se primjerice kod problema trgovačkog putnika (vidi poglavlje 2.6) [3].

Postupci selekcije

Svrha postupaka selekcije je očuvanje dobrih svojstava tj. gena i njihovo prenošenje na iduću generaciju jedinki. S obzirom na vrstu selekcije genetske algoritme dijelimo na generacijske i eliminacijske [1]. Generacijski genetski algoritmi u svakoj pojedinoj iteraciji raspolažu s dvije generacije: starom generacijom te novom, sastavljenom od izabranih jedinki iz stare generacije koje će onda sudjelovati u procesu reprodukcije. Generacijski genetski algoritmi tipično koriste jednostavnu i turnirsku selekciju, dok eliminacijski genetski algoritmi koriste jednostavnu eliminacijsku ili turnirsku selekciju. Pogreška koju bi valjalo izbjeći u genetskom algoritmu je da prilikom selekcije samo odaberemo određeni broj najboljih jedinki koje ćemo prenijeti u iduću generaciju. Taj je postupak loš jer dovodi do prerane konvergencije rješenja genetskog algoritma. Važno je zato da prilikom selekcije postoji mogućnost očuvanja i dobroga genetskog materijala koji nose lošije jedinke pa i one trebaju imati neku (manju) vjerojatnost preživljavanja i sudjelovanja u procesu reprodukcije.

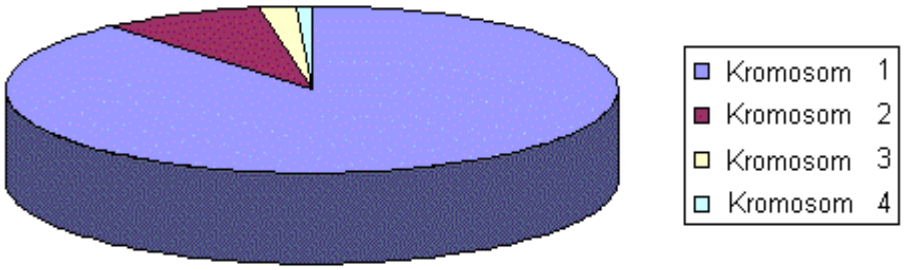
Selekcijom se, dakle, odabiru roditeljski kromosomi kombiniranjem čijega će genetskog materijala nastati novi kromosomi.

Jednostavna selekcija (*roulette wheel parent selection*) generira novu jednako brojnu populaciju jedinki. Što kromosom ima bolje gene, to je veća vjerojatnost da će biti izabran za roditeljski kromosom. Ime ovog postupaka selekcije dolazi od načina na koji ovu situaciju možemo lako predočiti. Zamislimo kotač za *roulette*. Na njemu nisu brojevi nego oznake svakog kromosoma, a veličina prostora na kotaču koji zauzima pojedini kromosom proporcionalna je njegovoj dobroti. Kada zavrtimo kotač i bacimo u njega kuglicu, veća je vjerojatnost da će se kuglica zaustaviti na području kromosoma s većom dobrotom [3].

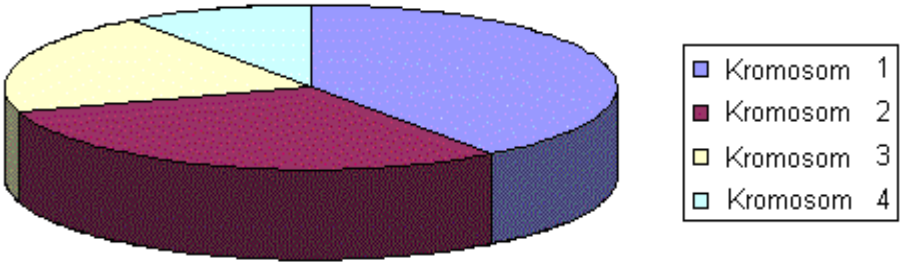
Nije teško primjetiti da je pri gore opisanoj metodi selekcije moguće da, ako je neki kromosom dosta bolji od drugih, on bude izabran znatno veći broj puta. Takvo pojavljivanje duplikata kromosoma u sljedećoj generaciji pokazalo se kao veliki nedostatak genetskih algoritama [1]. Broj duplikata može narasti i na 50% populacije, što nepotrebno usporava izvršavanje algoritma.

Taj problem nastoji se riješiti uvođenjem rangiranja kromosoma. Kromosomi se rangiraju s obzirom na dobrotu, tako da je na prvom mjestu najlošiji kromosom a na poslijednjem, n-tom mjestu, najbolji. Selekcija se sada izvodi kao u prethodnom primjeru, ali naš zamišljeni kotač *roulettea* sada organiziramo tako da pojedini kromosom u njemu zauzima površinu proporcionalnu svom rangu. Tako smo postigli smanjenje razlike u vjerojatnosti odabira dva susjedno rangirana kromosoma.

Drugim riječima, postigli smo manje duplikata i veću vjerojatnost očuvanja raznolikosti genetskog materijala. Ovaj se postupak naziva selekcija po rangu (*rank selection*) [3]. Vizualizacija jedonstavne selekcije prikazana je na slici 2.4, a selekcija po rangu na slici 2.5.



Slika 2.4: ***Roulette wheel selection roulette wheel***



Slika 2.5: ***Rank selection roulette wheel***

Turnirska selekcija (*tournament selection*) kod genetskog algoritma znači u svakom koraku generiranje nove populacije jedinki tako da onoliko puta koliko nam populacija ima jedinki odaberemo iz nje slučajnih *k* jedinki [1]. Najbolju od svakih tih *k* izabranih jedinki izdvajamo, kako bi na nju kasnije djelovali genetskim operatorima.

Eliminacijska selekcija (*steady-state selection*) funkcionira tako da ne bira dobar, nego loš genetski materijal [1]. U svakoj generaciji izabire se nekoliko najboljih kromosoma od kojih se djelovanjem genetskih operatora stvaraju novi, te se bira nekoliko najlošijih kromosoma koji se gube. Ostali kromosomi preživljavaju do iduće generacije.

Dobro rješenje koje smo dobili kroz neki broj iteracija može se izgubiti djelovanjem genetskih operatora. Kako bi se to izbjeglo često se pribjegava mehanizmu zaštite najbolje jedinke od bilo kakvih izmjena. Taj mehanizam naziva se *elitizam* [3].

Genetski operatori

Genetski operatori koje koriste genetski algoritmi dijele se na unarne, koji stvaraju novi kromosom mijenjajući manji dio genetskog materijala zadanog kromosoma, te na operatore višeg reda, koji stvaraju novi kromosom kombinirajući osobine nekoliko postojećih kromosoma [1].

Križanje

U procesu križanja sudjeluju dvije jedinke koje nazivamo *roditelji*. Rezultat križanja su jedna ili dvije nove jedinke – *djeca*. Djeca nasljeđuju neke gene svojih roditelja, a imajući na umu da roditelji, kako bi prošli selekcijski proces moraju imati bar neke dobre gene, zaključujemo kako će se ti dobri geni (barem pokušati) prenijeti na djecu. Cilj bi nam, naravno, bio dobiti djecu bolju od svojih roditelja.

Križanje se može definirati proizvoljnim brojem prekidnih točaka. Pogledajmo nekoliko primjera: na slici 2.6 nalazi se križanje sa samo jednom točkom prekida, na slici 2.7 prikazano je križanje s dvije točkje prekida. Na slici 2.8 odabrali smo 7 slučajnih točaka prekida. Slika 2.9 prikazuje takozvano *aritmetičko* križanje.

Slika 2.6: *Single point crossover* – križanje s jednom prekidnom točkom

Slika 2.7: *Two point crossover* – definiraju se dvije točke prekida

Slika 2.8: *Uniform crossover* – uzimaju se slučajno odabrani dijelovi prvog ili drugog roditelja za zadani broj točaka prekida

Slika 2.9: *Arithmetic crossover* – dijete je rezultat neke aritmetičke operacije između roditelja

U prvom slučaju križanja definirali smo jednu točku prekida. Do te točke, dijete je naslijedilo genetski materijal prvog roditelja, a nakon nje genetski materijal drugog roditelja. U drugom slučaju dodali smo još jednu točku prekida, nakon koje dijete opet nasljeđuje gene prvog roditelja.

Vjerojatnost je nasljeđivanja genetskog materijala od pojedinog roditelja obično jednaka i iznosi 0.5. Ukoliko je vjerojatnost da će dijete naslijediti gene jednog roditelja veća nego da će naslijediti gene drugog roditelja, tada govorimo o p-uniformnom križanju [1].

Poseban slučaj na koji valja obratiti pažnju je kada su oba roditelja jednaka. Raditi križanje tada je očito suvišno, jer je umjesto toga dovoljno samo kopirati jednog od roditelja. S druge strane, tako dobivamo samo još jedan duplikat nekog kromosoma, što nam nije u interesu. U praksi se zato često prije križanja provjerava različitost roditelja. Ukoliko ona nije prisutna jedan od roditelja se mutira pa se potom obavlja križanje.

Mutacija

Mutacija je slučajna promjena jednog ili više gena. To je unarna operacija, to jest djeluje nad samo jednom jedinkom. Vjerojatnost da će neki kromosom mutirati definira se u algoritmu. Ako zadana vjerojatnost teži u jedinicu, algoritam će se uglavnom svesti na slučajno pretraživanje prostora rješenja, a ako je pak vjerojatnost mutacije vrlo mala, realno je za očekivati da će skup rješenja brzo konvergirati prema nekom lokalnom minimumu ili maksimumu.

Razlikujemo nekoliko vrsta mutacija: *jednostavna mutacija*, *miješajuća mutacija* i *invertirajuća miješajuća mutacija* [1]. Kod jednostavne mutacije svaki bit kromosoma mijenja se s početno definiranom vjerojatnošću. Miješajuća mutacija odabire slučajni kromosom, prvu i drugu granicu te unutar nje ili izmiješa gene ili ih slučajno generira. Slično tome, invertirajuća mutacija također slučajno odabire granice dijela kromosoma nad kojim će djelovati, a potom invertira sve bitove unutar tih granica.

Mutacija kao genetski operator važna je iz već spomenutog razloga moguće konvergencije prema nekom neoptimalnom rješenju. Ona omogućuje da, čak i ako je cijela generacija zaglavila u području lokalnog minimuma ili maksimuma, pojava mutirane jedinke s boljim svojstvima pokrene daljnju potragu za optimalnim rješenjem.

Još je jedna uloga mutacije u obnavljanju izgubljenoga genetskog materijala [1]. Ako bi sve jedinke u generaciji imale jednaku vrijednost nekoga određenoga gena, on se ne bi mogao promijeniti samo križanjem. Na primjer, ako kromosome zapisujemo binarno, jedan bit na određenom bitnom mjestu koji bi bio jednak u svakom kromosomu značio bi smanjenje prostora pretraživanja za polovinu.

Primjer – Problem trgovačkog putnika (TSP)

Problem trgovačkog putnika jedan je od najčešće korištenih problema za demonstriranje (ne)efikasnosti nekog postupka optimizacije. Problem je zadan na sljedeći način: poznate su udanjenosti između svaka dva od ukupno N gradova koje trgovački putnik treba obići najkraćim mogućim putem (najkraći Hamiltonovski ciklus). Traži se redoslijed obilaska.

Najprije moramo pronaći najpraktičniji način definiranja kromosoma. U ovom slučaju tako ćemo, kao što je prikazano na slici 2.10, kromosom definirati kao polje cijelih brojeva, koje će predstavljati redoslijed obilaska gradova.

Kromosom 1	1 3 2 6 4 5

Slika 2.10: Kromosom za TSP

Ovaj kromosom će nam dakle predstavljati potencijalno rješenje obilaska gradova od prvog preko trećeg, drugog, šestog, četvrtog pa konačno do petog. Radi očuvanja konzistentnosti kromosoma sada ćemo genetski operator križanja morati modificirati. Nepotrebno je naime provjeravati rješenja koja ne sadrže u sebi sve gradove (a koja lako možemo dobiti standardnim križanjem), a do iste pogreške može doći i mutacijom. Zato uvodimo sljedeće preinake [3]:

1. Križanje sa jednom točkom prekida vršimo tako da do točke prekida kopiramo polje iz prvog roditelja, a nakon toga dodajemo brojeve gradova kojih nema redom kako se pojavljuju u drugom roditelju. Križanje sa dvije točke prekida vršimo tako da dva dijela prvog roditelja kopiramo, a između njih dolaze brojevi gradova kojih još nema, redom kojim se javljaju u drugom roditelju. Primjer križanja sa jednom točkom prekida na trećem mjestu dan je na slici 2.11.

Roditelj 1	1 3 2 6 4 5
Roditelj2	4 2 1 5 6 3
Dijete	1 3 4 2 5 6

Slika 2.11: Križanje s točkom prekida na trećem mjestu

2. Jednostavnu mutaciju sada ćemo obavljati tako da izmiješamo redoslijed nekoliko slučajno odabranih brojeva gradova. Druga inačica mutacije je jednostavna poboljšavajuća mutacija, koja znači isto što i jednostavna, ali će se obaviti samo ako je novonastali raspored bolji od starog. Primjer jednostavne mutacije dan je na slici 2.12.

Kromosom 1	1 3 2 6 4 5
Kromosom 1 (nakon mutacije)	2 3 5 6 4 1

Slika 2.12: Kromosom prije i nakon jednostavne mutacije koja djeluje na gradove 1, 2 i 5

Funkcija dobrote je trivijalna. Možemo ju na primjer računati kao sumu N najvećih udaljenosti između gradova umanjenu za sumu udaljenosti koja proizlazi iz zapisa u kromosomu.

Na primjer, neka je zadana tablica udaljenosti gradova 2.1.

Tablica 2.1: Tablica udaljenosti gradova za TSP

	1.	**2.**	**3.**	**4.**	**5.**	**6.**
1.	0	5	2	17	9	8
2.	5	0	6	6	10	2
3.	2	6	0	5	11	1
4.	17	6	5	0	2	6
5.	9	10	11	2	0	15
6.	8	2	1	6	15	0

Za šest zadanih gradova suma maksimalnih šest udaljenosti je:

$$S = 17 + 15 + 11 + 10 + 9 + 8 = 70$$

Za kromosom sa slike 10 funkcija dobrote tada će poprimit vrijednost:

$$F = S - (2 + 6 + 2 + 6 + 2 + 8) = 70 - 26 = 44$$

Broj generacija i veličinu populacije biramo u ovisnosti o N-u. Na primjer, za manje N-ove veličine do 10 zadovoljavajuće rješenje trebali bi uz petnaestak jedinki dobiti u oko 50 generacija.

Struktura našeg algoritma sada bi bila:

1. Kreiraj početnu populaciju jedinki slučajnim generiranjem nizova brojeva gradova;
2. Dok se ne ostvari zadani broj iteracija učini

   {
   - najbolju jedinku preslikaj u novu generaciju;
   - postupkom jednostavne selekcije odaberi roditelje i obavi križanje;
   - djeluj operatorom mutacije uz zadanu vjerojatnost;

   }
3. Kraj;

Implementaciju ovog algoritma uz samostalno zadavanje parametara moguće je isprobati na internet adresi http://cs.felk.cvut.cz/~xobitko/ga/tspexample.html.

3.Genetsko programiranje

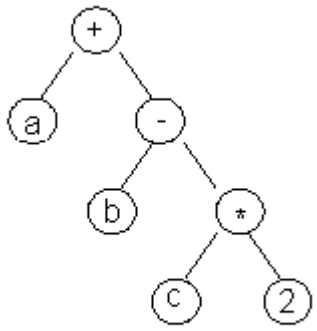
Uvod

Genetsko programiranje proširuje genetski model učenja na programe. Osnovna je ideja postići da računalo riješi neki problem, a da mu zapravo ne kažemo kako. Vjerojatno najznačajniji doprinos genetskom programiranju dao je Amerikanac John Koza, počevši od 70-ih godina 20. stoljeća. Trenutno je poznato najmanje 36 problema [4] kod kojih je genetskim programiranjem postignuto bolje rješenje od do tada najboljeg poznatog rješenja, od kojih je većina vezana za područja električnih krugova, molekularne biologije, raspoređivanja itd.

Postoje brojne sličnosti, ali i neke bitne razlike od genetskih algoritama. Temeljna je ideja ista: stvoriti neku populaciju početnih rješenja (u ovom slučaju programa za rješavanje problema). Kroz određeni broj generacija pronaći optimalno rješenje, djelovanjem genetskih operatora i provjeravanjem u kojoj nas mjeri pojedino rješenje zadovoljava.

Sličnosti i razlike genetskog programiranja i genetskih algoritama

Kod genetskih programa jedinke koje čine generaciju nisu više nizovi znakova ili strukture podataka definirane duljine, nego programi koji čine potencijalna rješenja problema. Radi jednostavnosti ti se programi vizualno ne predstavljaju kao linije koda, nego strukturirani kao stabla i to ne nužno jednako velika. Na primjer, jednostavan program koji računa vrijednost izraza $a + b - c * 2$ prikazuje se kako to pokazuje slika 3.1:



Slika 3.13: Prikaz programa pomoću stabla

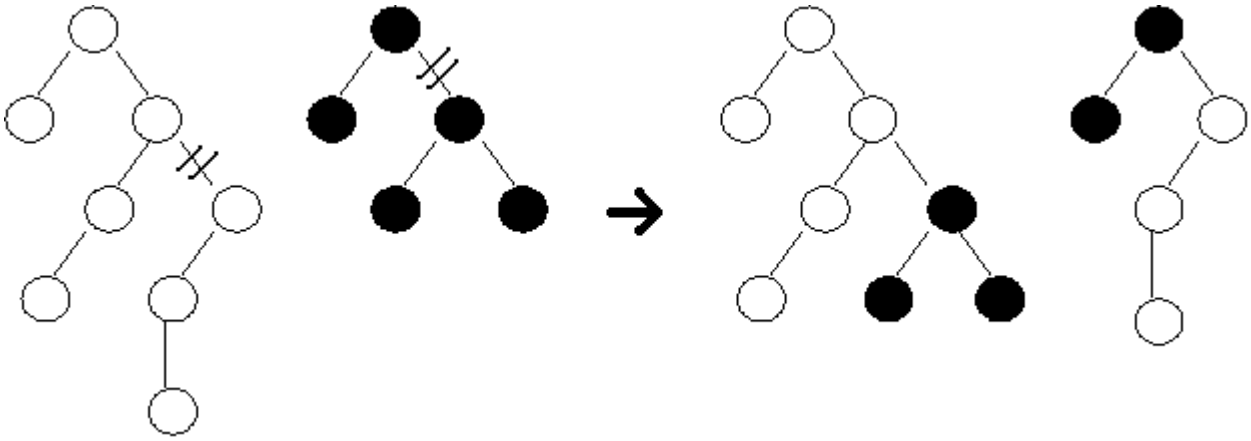
Unutarnji čvorovi stabla nazivaju se funkcije (*functions*) a listovi ulazi (*terminals*). Definiranjem ulaza i funkcija dobivamo skup komponenti od kojih se kreira program.

Početna populacija generira se na sljedeći način: najprije slučajno odabiremo korijen stabla iz skupa funkcija. Na njega nadodajemo slučajne daljnje elemente iz skupa funkcija ili ulaza. Taj proces nastavljamo sve dok svi listovi stabla nisu ulazi. Kod genetskog programiranja sve su funkcije i

ulazi u pravilu istog tipa podataka, što znači da će dobiveni program sigurno biti sintaksno ispravan. Uobičajeno je prilikom generiranja početne populacije ograničiti maksimalnu veličinu stabla. To je jednostavno ostvariti tako da, nakon što stablo postigne odgovarajuću maksimalnu dubinu, dalje na njega ne dodajemo još funkcija, nego samo ulaze. To je potpuna metoda (*full method*) [5] za stvaranje stabla koja također osigurava da su sve grane stabla jednake dubine. Druga često korištena metoda za određivanje dubine stabla je metoda rasta (*grow method*) [5]. Kod nje na korijen stabla dodajemo ili funkcije ili ulaze, ali nam je zadana maksimalna dubina stabla. U ovom slučaju sve generirane grane stabla neće nužno biti iste dubine. Kombinacija ove dvije metode koja započinje izgradnju stabla na određenoj dubini i nastavlja o nekog zadanog maksimuma naziva se *ramped half-and-half* [5]. Često se koristi jer predstavlja dobar izbor za generiranje raznolike početne populacije.

Nakon što je generirana početna populacija programa procjenjuje se uspješnost primjene programa na zadani problem (fukcija dobrote), te se ovisno o tome primjenjuju postupci selekcije. Selekcija se najčešće obavlja prethodno objašnjenim postupcima jednostavne ili turnirske selekcije.

Genetski operatori koje koristimo kod genetskog programiranja su, isto kao i kod genetskih algoritama, reprodukcija, križanje i mutacija. Križanje znači da u dva različita stabla izaberemo po jedno podstablo te im zamijenimo mjesta. Proces križanja prikazan je na slici 3.2. Mutacija se obavlja tako da se u slučajno odabranom stablu slučajno odabere neki čvor. Podstablo koje izrasta iz tog čvora se briše i na njegovo mjesto dolazi novo slučajno generirano podstablo. Reprodukcija znači kopiranje nepromijenjenog programa u novu generaciju a nerijetko se primjenjuje i mehanizam elitizma (objašnjeno u drugom poglavlju) [5].



Slika 3.14: Križanje kod genetskog programiranja

Parametri genetskog programa

Kao i kod genetskih algoritama i kod genetskog programiranja postoje parametri koji bitno utječu na izvođenje programa i u njemu se zadaju [5]. To su:

1. SKUP FUNKCIJA I ULAZA
   Skup ulaza uglavnom se sastoji od ulaza karakterističnih za promatrani problem i numeričkih konstanti, a skup funkcija od funkcija koje želimo obavljati nad tim ulazima. Sve su funkcije definirane nad istim tipom podataka, to jest ulaza.

2. POČETNA VELIČINA POPULACIJE
   Veličina populacije također spada među ključne parametre za izvođenje programa, a kreće se od 50 pa do više od nekoliko desetaka milijuna jedinki, opet ovisno o problemu koji promatramo.

3. BROJ GENERACIJA
   Zadaje se kao i kod genetskih algoritama: brojem iteracija ili duljinom vremena izvođenja.

4. VJEROJATNOST MUTACIJE I SELEKCIJE
   Identično odgovarajućim vjerojatnostima kod genetskih algoritama.

5. MAKSIMALNA POČETNA I KONAČNA VELIČINA STABLA
   Velika dozvoljena dubina ili uopće izuzimanje granice veličine stabla može rezultirati nerealno velikim stablima koja će oduzimati puno vremena za izvođenje. Premala stabla opet previše ograničavaju pretragu prostora programskih rješenja.

Primjer genetskog programa

Zanimljiv primjer genetskog programiranja je program koji za zadani graf funkcije na nekom intervalu aproksimira tu funkciju pomoću elementarnih funkcija. Demo-verzija ovog programa dostupna je na http://alphard.ethz.ch/Hafner/ggp/gp.htm.

Skup se funkcija ovog programa sastoji od osnovnih operatora zbrajanja, oduzimanja, množenja i dijeljenja ali i još od nekih prijeko potrebnih funkcija, kao što su trigonometrijske (kosinus) i logaritamska funkcija. Skup ulaza sastoji se, naravno, od varijable x i slučajno generiranih realnih konstanti. Iako nam takve funkcije nisu posebno zanimljiv objekt promatranja, ovakav program mogao bi imati poteškoća pri aproksimiranju funkcija koje su konstantne po cijeloj kodomeni. Za rješavanje tog problema obično se uvodi još jedan nelinearan ulaz p, čija se vrijednost određuje nekim algoritmom optimizacije [6].

Ovaj se program izvodi ili do pronalaska funkcije koja savršeno odgovara onoj zadanoj na grafu ili do prekida od strane korisnika. U ovom drugom slučaju program nakon prekida prikazuje najbolja rješenja koja je pronašao. U demo-verziji, nažalost, nije moguće vidjeti matematičku formulu funkcije kojom se aproksimira zadani graf.

Jedna od primjena ovog programa, kao što i sam autor predlaže, može biti i predviđanje kretanja cijena dionica na tržištu [6]. Naravno, kretanje cijena dionica u stvarnosti ovisi o brojnim parametrima od kojih su mnogi «nepredvidljivi». Program bi u ovom slučaju odredio funkciju po kojoj se cijena neke dionice ponašala u vremenskom razdoblju prikazanom na učitanom grafu. Daljnje kretanje cijene iste dionice dobili bi jednostavno iscrtavajući dobivenu funkciju na proizvoljno velikom intervalu. Ovom metodom, ako se ona primjenjuje uz dobro poznavanje i praćenje stvarnih čimbenika koji utječu na cijenu dionica na tržištu, zaista je moguće ostvariti neke zamjetnije rezultate.

Uz to, primjene ove metode (symbolic regression) mogu biti još mnoge: ekonometrijsko modeliranje i predviđanje, empirijsko otkrivanje fizikalnih zakona, provedba simboličke integracije ili derivacije funkcija, otkrivanje simboličkih rješenja polinoma, simboličko predstavljanje

vremenskih nizova, otkrivanje funkcija prioriteta aktivnosti u raspoređivanju, otkrivanje funkcija sažimanja multimedijalnih podataka itd.

4.Evolucijske strategije

Općenito o evolucijskim strategijama

Evolucijske strategije još su jedna od tehnika optimizacije iz područja evolucijskih algoritama. Njihov razvoj započinju I. Rechenberg i H. Schwefel u Njemačkoj 60-ih godina 20. stoljeća na berlinskom Tehničkom fakultetu [8]. Prvu su primjenu pronašle u rješavanju problema najboljega aerodinamičnog dizajna, a danas se primjenjuju i primjerice pri određivanju parametara prema kojima međudjeluju pojedine galaksije u svemiru [10].

Evolucijske strategije potencijalna rješenja prikazuju kao konstantno velike vektore realnih brojeva. Broj na određenu mjestu unutar vektora opisuje neku karakteristiku rješenja [7]. Kroz određeni broj generacija opet se pokušava pronaći najbolje rješenje problema korištenjem operatora selekcije i mutacije.

Za razliku od genetskih algoritama, koji nerijetko rade s populacijama veličine nekoliko tisuća jedinki, u ovom će nam slučaju veličina populacije tipično biti dva: trenutni roditelj i novi kromosom koji je rezultat mutacije nad njim [8]. Suvremene evolucijske strategije ipak ponekad koriste više roditelja i dodatno operator križanja kako bi se izbjeglo eventualno zaglavljivanje rješenja u području nekoga lokalnog optimuma [7].

Mutacija kod evolucijskih strategija naziva se tipično *Gausijanska mutacija* i obavlja se tako da svakoj komponenti vektora dodamo slučajnu varijablu koja poprima vrijednosti prema Gaussovoj ili normalnoj razdiobi [8]. Normalna razdioba zadaje se u potpunosti s dva parametra: μ i σ. Za μ se obično postavlja vrijednost nula, dok se vrijednost σ mijenja iz generacije u generaciju.

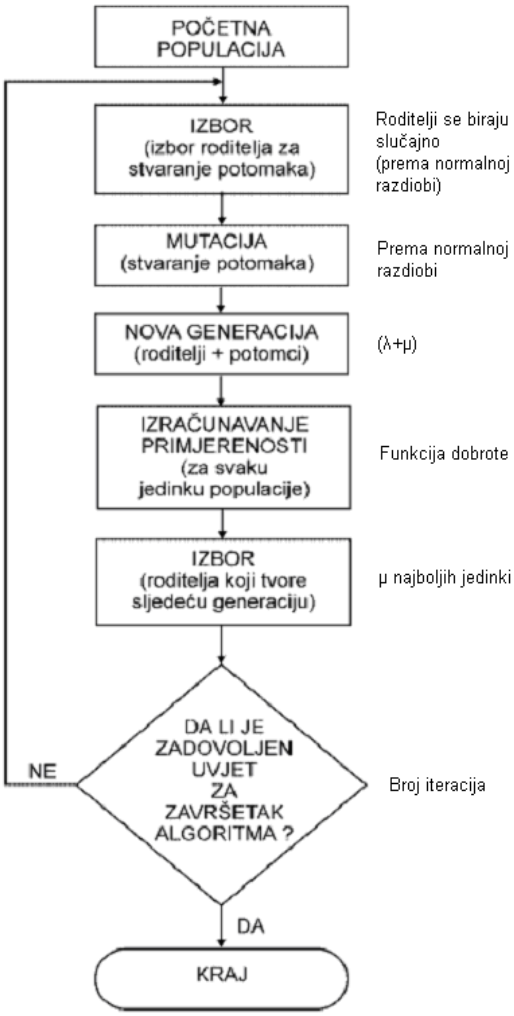
Općenito se može stvoriti λ mutacija μ roditelja (nema veze s prethodnim parametrom normalne razdiobe μ) od kojih biramo μ najboljih mutiranih jedinki i prenosimo ih u iduću generaciju. Tada govorimo o (λ,μ)-selekciji, ili u prethodno navedenom slučaju (1,1)-selekciji [8]. Ako jedinke nove generacije biramo i između roditelja i djece (mutiranih roditelja), tada je riječ o $(\lambda+\mu)$ selekciji.

Postupak kreiranja nove generacije možemo opisati kroz sljedeće korake:

1. Stvori novih λ jedinki tako da
   1) Odaberi slučajnih $\varsigma \leq \mu$ roditelja
   2) Križaj ς roditelja kako bi dobio dijete
   3) Odredi salučajnu novu vrijednost parametra *p* prema normalnoj razdiobi
   4) Mutiraj dijete pomoću nove vrijednosti parametra *p*

2. Odaberi novu populaciju na temelju funkcije dobrote iz
   1) populacije djece - (λ,μ)-selekcija
   2) populacije djece i roditelja - $(\lambda+\mu)$-selekciji

Dijagram tijeka evolucijske strategije prikazan je na slici 4.1.



Slika 4.15: Dijagram tijeka evolucijske strategije

5.Evolucijsko programiranje

Općenito o evolucijskom programiranju

Evolucijsko programiranje prvi je počeo razvijati J. L. Fogel 70-ih godina 20. stoljeća. Njegov interes je bio usredotočen na razvijanje umjetne inteligencije, te je uspio napraviti matematički automat za predviđanje binarnih vremenskih nizova [9].

Glavna razlika između evolucijskog programiranja i ostalih evolucijskih algoritama je u tome da kod evolucijskog programiranja ne postoji razmjena genetskog materijala između jedinki, odnosno, ne koristi se operator križanja već samo mutacije. Ako to zanemarimo, evolucijsko je programiranje vrlo slično evolucijskim strategijama.

Uobičajena metoda selekcije pri evolucijskom programiranju je da svaku od N jedinki iz generacije mutiramo te dobijemo ukupno 2N jedinki od kojih N najboljih ostavimo za iduću generaciju. Mutacija koja se primjenjuje obično je gausijanska kao i kod evolucijskih strategija [9].

Evolucijsko programiranje najčešće se primjenjuje na traženje minimuma ili maksimuma funkcija realnih varijabli.

6.Zaključak

Osnovne ideje evolucijskih algoritama nije teško shvatiti. U konačnici se oni uvijek svode na pronalaženje najboljeg rješenja od velikog broja pretraživanih, korištenjem nekih metoda koje bi nam trebale omogućiti brži dolazak do cilja. Evolucijski algoritmi zato i jesu pronašli primjenu u brojnim područjima, od traženja minimuma ili maksimuma zadane funkcije, problema raspoređivanja, pronalaska najkraćeg puta, preko idealnog aerodinamičnog dizajna i dizajna uopće, pa sve do predviđanja kretanja cijena dionica na burzi. Postupak dobivanja rješenja je ponovljiv tako da uvijek možemo provjeriti postojeće ili čak dobiti još bolje rješenje, a čak i ako ne dobijemo idealno, dobit ćemo neko dobro rješenje. S druge strane, napraviti dobar program koji koristi evolucijske algoritme nije jednostavan posao. Često je potrebno i promijeniti neke karakteristike genetskih operatora, pronaći način za definiranje funkcije dobrote, a s obzirom na to da je prostor na kojem pretražujemo rješenja iznimno velik, potrebna nam je velika procesorska snaga i dulje vrijeme izvođenja programa.

Unatoč navedenim poteškoćama u realizaciji, evolucijski algoritmi dosad su pokazali veliku uspješnost i nastavljaju se razvijati i primjenjivati i dalje.

7.Popis literature

[1] Golub, M.. Golub, M., Genetski algoritam, Prvi dio, 27. rujna 2004., verzija 2.3

[2] Automated Antenna Design**,** *Automated Design and Optimization for Increased Performance and Reliability on NASA Missions*, http://ic.arc.nasa.gov/projects/esg/research/antenna.htm, 19. ožujka 2007.

[3] Obitko, M., IX. Selection, X. Encoding, 1998., *Genetic algotithms*, http://cs.felk.cvut.cz/~xobitko/ga/selection.html, http://cs.felk.cvut.cz/~xobitko/ga/encoding.html, 20. ožujka 2007.

[4] Koza, R. J., What is Genetic Programming (GP)?, How Genetic Programming Works, 1. kolovoza 2004., http://www.genetic-programming.com/, 20. ožujka 2007.

[5] Loveard, T., Genetic Programming For Clasification Learning Problems, doktorski rad, Royal Melbourne Institute of Technology, 2003.

[6] Hafner, Ch., Ch. Hafner's Generalized Genetic Program (GGP), 24. siječnja 2006., http://alphard.ethz.ch/Hafner/ggp/gp.htm, 20. ožujka 2007.

[7] Wikipedia, the free encyclopedia, Evolution strategy, 2. siječnja 2007., http://en.wikipedia.org/wiki/Evolution_strategy, 22. ožujka 2007.

[8] Eiben A.E., Smith J.E., Evolution strategies, *Evolutionary computing*, http://www.cs.vu.nl/~gusz/ecbook/slides/Evolution_strategies.ppt, 25. ožujka 2007.

[9] Dorronsoro, B., Evolution Sstrategies, Evolutionary Programming, studeni 2004., *What Cellular Evolutionary Algorithms (cEAs) are?*, http://neo.lcc.uma.es/cEA-web/ES.htm, http://neo.lcc.uma.es/cEA-web/EP.htm, 25. ožujka 2007.

[10]Gómez, J.C., Fuentes, O., Athanassoula L., Bosna A., Using Evolution Strategies to Find a Dynamical Model of the M81 Triplet, http://www.cyber.brad.ac.uk/egso/KES2004/gomez.pdf, 25. ožujka 2007.

[11]Grundler, D., Evolucijski algoritmi (I), Pobude i načela, 13. svibnja 2007., www.hrcak.srce.hr/index.php?show=clanak_download&id_clanak_jezik=8743, 10. svibnja 2007.

8.Sažetak

Razvoj evolucijskih algoritama započinje u drugoj polovici 20. stoljeća. Njihovo funkcioniranje temelji se na preslikavanju modela evolucije iz prirode koji je prvi uspostavio Charles Darwin još sredinom 19. stoljeća. Evolucijski algoritmi dijele se u četiri skupine, a to su genetski algoritmi, genetsko programiranje, evolucijske strategije i evolucijsko programiranje. Evolucijski algoritmi korištenjem genetskih operatora pretražuju prostor rješenja u potrazi za najboljim.

Genetski algoritmi tako kreiraju početnu populaciju rješenja od kojih se bolja prenose u sljedeću generaciju ili križaju i mutiraju. Genetski programi stvaraju populaciju programa koji se testiraju na zadanom problemu i ovisno o uspješnosti križaju i prenose u iduću generaciju. Karakterističnost evolucijskih strategija i evolucijskog programiranja je manji broj jedinki u prvoj generaciji, te mutacije kod kojih se jedinka ne mijenja slučajno, nego joj se najčešće svim komponentama dodaje parametar s normalnom razdiobom. Nešto dublji uvid u genetske operatore i način funkcioniranja evolucijskih algoritama, prvenstveno genetskih algoritama i programa, tema je ovog seminarskog rada.