Sadržaj

1. Uvod

Evolucijsko programiranje je jedna od stohastičkih optimizacijskih strategija evolucijskih agoritama. Iako kao i ostali evolucijski algoritmi ne daje uvijek potpuno točno rješenje, evolucijsko programiranje je našlo široku primjenu u farmaceutskoj industriji, vojnom planiranju, sustavima identifikacije, epidemiologiji, upravljanju prometom pa čak i pri tehnikama otkrivanja raka.

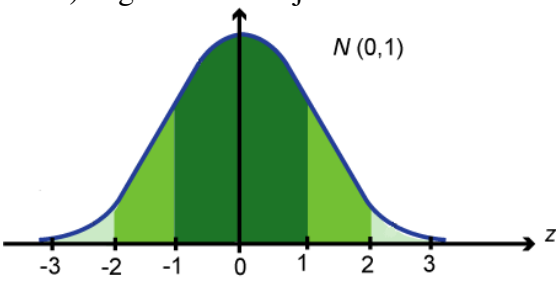
U drugom poglavlju se opisuje sami algoritam. Odjeljak 2.1 služi za upoznavanje sa algoritmom, dok se u odjeljku 2.2 ozbiljnije razmatra rad algoritma. Potom se u odjeljku 2.3 nalazi pseudokod kojeg zadovoljava svaki algoritam evolucijskog programiranja. Slijede primjeri primjene, sličnosti sa ostalim evolucijskim algoritmima te zaključak u poglavljima 3, 4 i 5. Konačno, popis korištene literature se nalazi u posljednjem poglavlju.

2. Opis algoritma

U ovom su poglavlju prikazane karakteristike evolucijskog programiranja. Prilikom upoznavanja sa algoritmom je potrebno znati da se često, zbog boljih rezultata, u evolucijskim algoritmima odstupa od nekih svojstava koje karakteriziraju određenu skupinu algoritama.

2.1 Osnovni koraci

Evolucijsko programiranje za svoj rad koristi genetske operatore selekcije i mutacije. Križanje, kao genetski operator, se ne koristi. Selekcija je proces kojim genetski algoritam čuva dobre, a odbacuje loše jedinke iz populacije rješenja. Selekcijom se odabiru jedinke koje će sudjelovati u reprodukciji, te tako prenijeti svoj genetski materijal (značajke jedinke) na slijedeću generaciju. Kako i loše jedinke mogu sadržavati dobre i korisne gene, potrebno je i njima omogućiti barem minimalnu vjerojatnost za razmnožavanje. Mutacija mijenja vrijednost pojedinim genima. Intenzitet mutacije se za ovaj algoritam odvija prema gaussovoj jediničnoj normalnoj razdiobi. Na temelju toga se zaključuje da su vjerojatnije male mutacije (djeluju na manje gena jedinke) nego one snažnije.



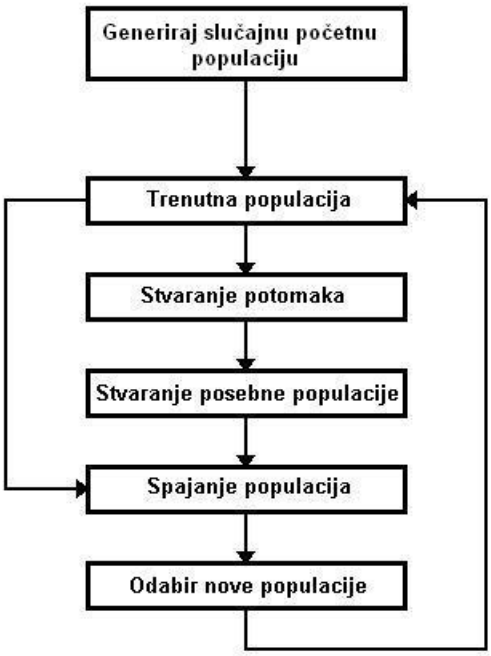
Slika 2-1 Gaussova jedinična normalna razdioba

Za Evolucijsko programiranje postoji početna pretpostavka da se prostor problema zadatka može okarakterizirati pomoću varijabli i da postoji optimalno rješenje dobiveno uz pomoć tih istih varijabli. Ako uzmemo za primjer problem trgovačkog putnika, kod njega bi svi mogući putovi sačinjavali prostor problema zadatka. Svakom putu bi se mogla pridodati brojčana vrijednost koja bi ocjenjivala korisnost puta(na temelju onog puta koji tražimo, odnosno u ovom slučaju najkraćeg puta). Za cilj možemo postaviti traženje najkraćeg puta u cijelom prostoru ili kod zahtijevnijih prostora, gdje nam je važna brzina pronalaska, lokalno najkraćih puteva.
Rad algoritma evolucijskog programiranja se odvija u tri osnovna koraka:

1) Slučajnim se odabirom generira početna populacija te se evaluiraju sve jedinke. Veličina populacije ovisi o vrsti problema. Ne postoji univerzalni način za određivanje najefikasnijeg broja jedinki, ali o njemu ovisi brzina optimizacije.
2) svaka jedinka se kopira u novu populaciju, gdje se mutira prema zadanoj distribuciji mutacije. Jačina mutacije ovisi o funkcionalnim promjenama nametnutim od strane roditelja.
3) Stare jedinke i novi potomci se evaluiraju te prema tome odabiru za povratak u prvotnu populaciju kako bi ponovno prošli proces selekcije i mutacije.

2.2 Rad algoritma

Rad algoritma prikazan je na slici 2-2:



Slika 2-2 Prikaz rada algoritma evolucijskog programiranja

2.2.1 Generiranje početne populacije

Na veličinu početne populacije utječu:

1. zahtjevnost problema i veličina prostora rješenja
2. dostupna količina memorije
3. vremensko ograničenje unutar kojega problem mora biti riješen

Prvi čimbenik nije uvijek poznat na početku stvaranja populacije, tako da je najbolje mijenjati veličinu populacije i na temelju dobivenog rješenja, koje se ne mijenja znakovito sa daljnjim povećanjem, zaključiti koja veličina je najprikladnija. Drugi čimbenik je važan s obzirom da generirana populacija i svi potomci moraju biti pohranjeni u memoriji računala. To postaje problem ukoliko se populacija sastoji od vrlo velikog broja kompleksnih rješenja. Na treći čimbenik utječu želje korisnika gdje se pri zahtjevnijim zadacima mora raditi kompromis između sporog i kvalitetnog te brzog i ne toliko kvalitetnog rješenja.

Imajući navedeno na umu, slučajno se generira početna populacija jedinki veličine μ. Svaka se jedinka predstavlja kao par realnih vrijednosti $(x_i, \eta_i), \forall i \in \{1,...,\mu\}$, gdje je x_i vektor elemenata rješenja, a η_i pripadajuća varijanca kojom određuemo intenzitet mutacije. Varijabla k označava vrijednost brojača iteracija i ona se zasada postavlja na nulu.

2.2.2 Trenutna populacija

Trenutna populacija je skupina jedinki pohranjena od strane programa u trenutku kada započinje nova iteracija algoritma. Nju se koristi za stvaranje potomaka. Jedinke se evaluiraju na temelju njihove sličnosti sa optimalnim rješenjima funkcijom $f(x_i)$.

2.2.3 Stvaranje potomaka

Potomci se stvaraju mutacijom jedinki na temelju gaussove normalne jedinične razdiobe. Jedinke koje sudjeluju u stvaranju potomaka nazivamo roditeljima. Svaki roditelj (x_i, η_i), i = 1,...,μ stvara jednog potomka (x_i', η_i') prema izrazima:

$$x_i'(j) = x_i(j) + \eta_i(j)N(0,1)$$

$$\eta_i'(j) = \eta_i(j)\exp\left(\left(\sqrt{2n}\right)^{-1} N(0,1) + \left(\sqrt{2\sqrt{n}}\right)^{-1} N_j(0,1)\right)$$

Gdje j predstavlja oznaku mjesta komponente unutar jedinke, odnosno izraz ($x_i(j)$, $\eta_i(j)$) predstavlja jedan gen jedinke (x_i, η_i). Varijabla j može poprimiti vrijednost od 1 do n (n je broj gena unutar jedinke).
N(0,1) označava jednodimenzionalni slučajni broj dobiven na temelju gaussove jedinične normalne razdiobe. $N_j(0,1)$ iznova generira slučajni broj za svaku vrijednost j.

2.2.4 Stvaranje posebne populacije i spajanje sa trenutnom

Svaki potomak se pohranjuje u posebnu populaciju. Kada se stvore svi potomci nastaje spajanje posebne i trenutne populacije, a potom se primjenjuje postupak selekcije čiji će rezultati sačinjavati novu trenutnu populaciju. Selekcija se obavlja turnirski i opisana je u slijedećem odjeljku.

2.2.5 Odabir unutar trenutne populacije

Iz unije posebne i trenutne populacije se odabire onoliko rješenja koliko ih je bilo unutar početne populacije(μ). Kada se evaluiraju sve jedinke na temelju funkcije $f(x_i)$, pristupa se turnirskoj selekciji. Za svaku jedinku se slučajno bira q protivnika od svih jedinki. Ako prilikom usporedbe jedinka ima veću dobrotu od svog protivnika, onda se njoj bilježi „pobjeda“. Odabire se μ jedinki u populaciji (x_i, η_i) i (x_i', η_i'), $\forall i \in \{1,...,\mu\}$, koje imaju najviše pobjeda.
Algoritam prestaje s radom ako je odabranim skupom rješenja dosegnut zadani kriterij. U suprotnom se uvećava varijabla k, odabrani skup rješenja postaje nova trenutna populacija i algoritam se ponavlja od koraka provođenja mutacije nad trenutnom populacijom.

2.3 Pseudokod

```
Evolucijsko programiranje{

k = 0;

slučajnim postupkom generiraj početnu populaciju potencijalnih
rješenja P_k veličine n;

evaluiraj P_k;

sve dok nije zadovoljen uvjet završetka evolucijskog procesa

     {

     i = 0;

     dok je i < μ

          {

          P'_k [i]= mutiraj P_k[i];

          i++

          }

     P''_k = P_k ∪ P'_k;

     evaluiraj(P''_k);

     i = 0;

     dok je i < μ

          {

          P_k+1[i]= selektiraj(P''_k[i]);

          i++;

          }

     k = k + 1;

     }

}
```

3. Primjeri primjene

3.1 Primjer evolucijskog programiranja na temelju ispitivanja različitih okusa

Ovim primjerom se opisuje jednostavni algoritam kojim se određuje optimalna mješavina 15 tekućina. Funkciju kojom se evaluiraju jedinke(mješavine) predstavlja subjektivno mišljenje kušača. Konvergencija prema rješenju ovog algoritma je dovoljno brza kako bi se mogla iskoristiti za demonstraciju evolucijskog programiranja.
Algoritam se sastoji od tri djela:

1) Stvara se početna populacija rješenja. Za ovaj primjer je odabrano 15 tekućina koje predstavljaju lepezu različitih, ujedno i uobičajenih, okusa. Vektor početnog rješenja je generiran slučajnim odabirom 15 vrijednosti između nule u jedinice. Potom je svaka vrijednost normalizirana djeljenjem sa zbrojem svih vrijednosti unutar vektora. Sada svaka vrijednost predstavlja postotak pojedine tekućine u mješavini. Potom je na isti način generirano još devet vektora rješenja.

Zašećerena voda	Sok od đumbira	Nektar jagode
Zasoljena voda	Sok od ananasa	Čaj
Sok od maline	Nektar breskve	Mlijeko
Sok od grejpa	Sok od brusnice	Kava
Sok od jabuke	Sok od grožđa	Čokolada

Tablica 3.1 Okusi korišteni u primjeru

2) Evaluira se dobrota svakog rješenja. Kako bi se dobila jedna šalica tekućine pomnožene su vrijednosti vektora sa 48 malih žličica za čaj. Dobivena tekućina prenesena u posudu. Nakon kušanja svih 10 mješavina, kušač je odabrao najbolje 3 prema svom subjektivnom mišljenju. Te su tekućine postale roditelji za slijedeću generaciju potomaka.

3) Roditelji su mutirani prema statističkoj distribuciji koja favorizira manje promjene nego one snažnije.

Najbolje tri mješavine predstavljamo kao 3 vektora $a_1(i)$, $a_2(i)$ te $a_3(i)$. Najbolji izbor iz prijašnje generacije se definira sa $a_0(i)$. Iz ta 4 vektora stavramo 3 vektora smjera koji će usmjeriti nova rješenja da budu sve prihvatljivija.

$$\Delta_1(i) = a_1(i) - a_0(i)$$
$$\Delta_2(i) = a_2(i) - a_0(i)$$
$$\Delta_3(i) = a_3(i) - a_0(i)$$

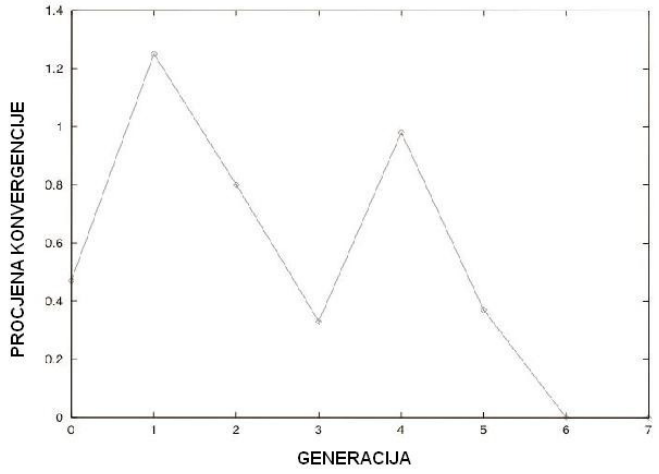
Deset novih rješenja (potomaka) koji će sačinjavati novu generaciju se definiraju na slijedeći način.

$$v_1(i) = a_1(i)$$
$$v_2(i) = a_1(i) + \Delta_1(i)$$
$$v_3(i) = a_1(i) + 2\Delta_1(i)$$
$$v_4(i) = a_1(i) + \Delta_1(i) + r_1(i)$$
$$v_5(i) = a_1(i) + \Delta_1(i) + r_2(i)$$
$$v_6(i) = a_1(i) + \Delta_2(i)$$
$$v_7(i) = a_1(i) + 2\Delta_2(i)$$
$$v_8(i) = a_1(i) + \Delta_2(i) + r_3(i)$$
$$v_9(i) = a_1(i) + \Delta_2(i) + r_4(i)$$
$$v_{10}(i) = a_1(i) + \Delta_3(i)$$

Gdje je $r_n(i)$ vektor koji sadržava slučajne vrijednosti između -1 i 1. Na kraju ovog koraka se normalizira svaki novi vektor rješenja. Uključivanjem najbolje mješavine prošle generacije u trenutnu htjelo se napraviti referencu prema kojoj će se stvarati dobra rješenja, odnosno ostvariti elitizam.
Prema navodima kušača, početna generacija mješavina je imala odvratan okus. Odabrana su ona rješenja koja su se nekako mogli popiti. U drugoj generaciji su bili slični dojmovi za većinu mješavina. Slijedeća generacija je izbacila ipak malo ukusnije napitke. U većini su počeli dominirati određeni okusi ali i kombinacije okusa su bile „smislenije“. Sada je pri odabiru počeo utjecati i okus koji ostaje nakon isprobavanja. U šestoj generaciji se najbolji napitak nije promijenio u odnosu na prošlu generaciju, a razlike između najbolja tri su bile vrlo male. Kada je i u sedmoj generaciji najbolji napitak ostao isti, isprobavanje je prekinuto.

Kako bi se izračunala konvergencija algoritma računamo apsolutnu vrijednost razlike između trenutnog najboljeg rješenja i najboljeg rješenja prošle generacije: $\Delta c(i) = \sum \left| a_1(i) - a_0(i) \right|$

Iako njime ne možemo osjetiti okus mješavina koje su nastajale, ipak nam dočarava kako su se mijenjali omjeri svakog okusa unutar tekućine. Konvergencija tokom cjelog ispitivanja je prikazana slikom 3.1

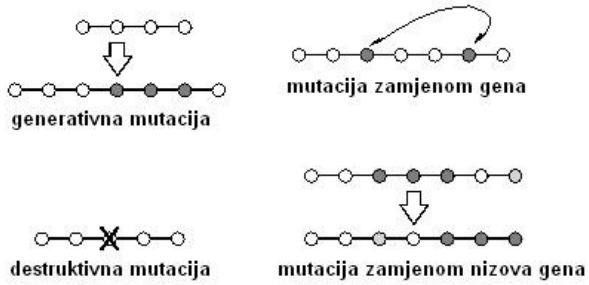


Slika 3-1 Prikaz konvergencije tokom ispitivanja mješavina tekućina

Većina tradicionalnih metoda optimacije ne bi bila prikladna u ovom eksperimentu zbog subjektivne prirode evaluacije i velikog broja mogućih rješenja (15^{49}). Evolucijski algoritam (evolucijsko programiranje) jednostavno i brzo nalazi rješenje zadanog problema.

3.2 Evolutionary Programming Toolkit (EPTK)

Evolutionary Programming Toolkit je primjer programskog ostvarenja algoritma evolucijskog programiranja. Koncept je zamišljen tako da se mapiraju djelovi slijeda DNA koji se sastoje od znakova A, T, C i G prema nazivima dušičnih baza (adenin, timin, citozin i gvanin), prema nekom zadanom algoritmu. Nizovi se evaluiraju te se oni najbolji koriste za reprodukciju tako što ih se kopira te naprave dvije mutirane kopije. Mutacija može uključivati mijenjanje, brisanje i umetanje znakova kako je prikazano slikom 3-2.



Slika 3-2 Prikaz mutacija implementiranih u programskom rješenju

Manje vrijedni nizove se brišu slučajnim odabirom kroz generacije. Vrijedniji niz može mutacijom postati još „snažniji“, ili može postati slabiji te odumrijeti. U svakom slučaju ostaju najjači generirani nizovi.
Programsko ostvarenje je implementirano uz pomoć dviju glavnih funkcija.

1. `double rank (string dna);`
Ova funkcija predstavlja funkciju cilja. Ona vraća vrijednost (rang) DNA niza, s tim da manji broj predstavlja veću vrijednost. Funkcija služi za eksperiment u kojem želimo dobiti nizove sa što više A dušičnih baza.

```
public double rank (string dna)
{
   double ret = 100;
   for (int i=0; i<&ltdna.Length; i++)
      if ( dna[i] == 'A' )
         ret--;
   return ret;
}
```

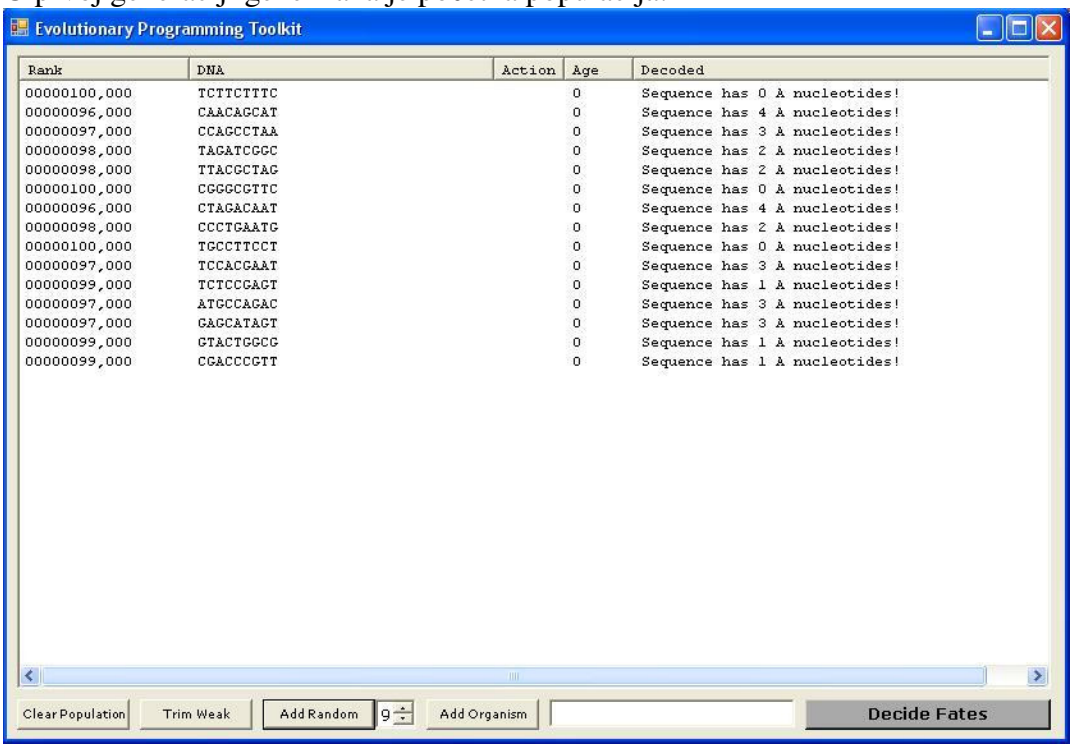
Sa svakim novodobivenim znakom A vrijednost se povećava (rang se smanjuje) i time dobivamo „jači“ niz.

2. `string decode (string dna);`
U svrhu bolje predodžbe koristimo slijedeću funkciju. Ona vraća string u kojem je ispisan broj znakova A za pojedini niz.

```
public string decode (string dna)
{
   return "Sequence has " + (100 - rank(dna)) + " A nucleotides!";
}
```

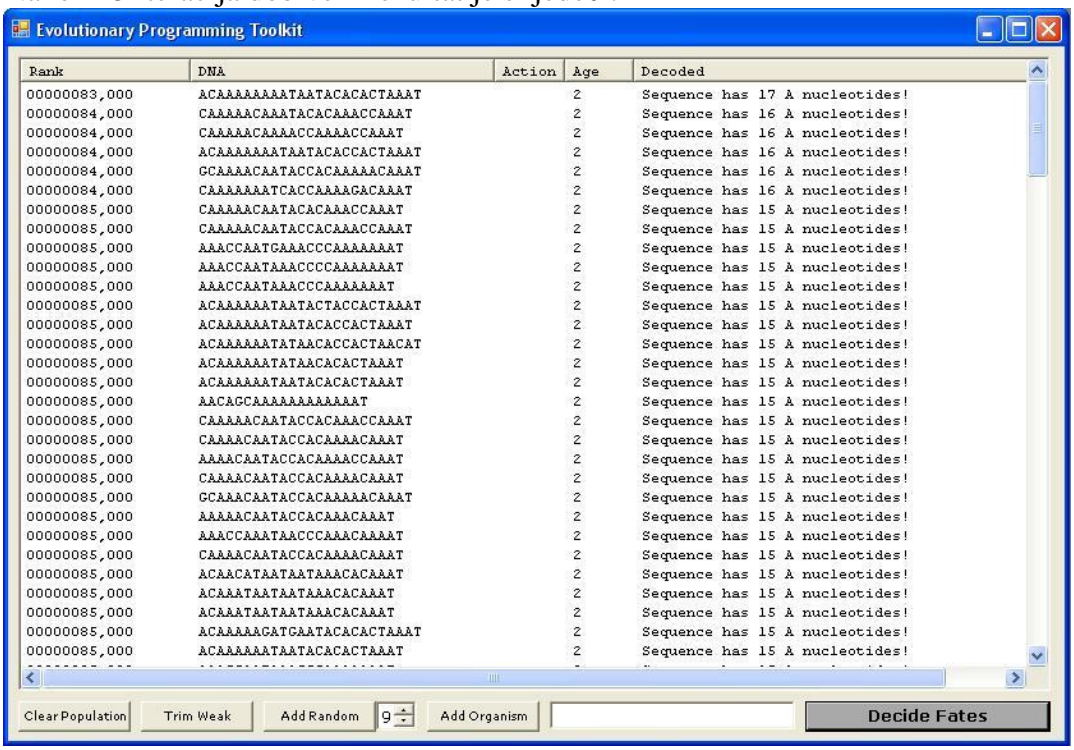
Primjer rada

U prvoj generaciji generirana je početna populacija:



Slika 3-3 Slučajno generirani nizovi

Nakon 25 iteracija dobiveni rezultat je slijedeći:



Slika 3-4 Prikaz dobivenih nizova nakon 25 iteracija

Pošto je određeno da više znakova A donosi veću vrijednost, takvi nizovi postaju dominantni. Ponavljajući proces nastaju nizovi sa većom vrijednosti sve dok se ne dođe do ranga 0 odnosno niz(ov)a sa 100 A znakova.

4. Sličnosti sa ostalim evolucijskim algoritmima

4.1 Evolucijsko programiranje i genetski algoritmi

Evolucijsko programiranje je usko povezano s genetskim algoritmima. Obje tehnike inicijaliziraju početnu populaciju jedinki te ih potom iterativno mijenjaju u svrhu približavanja rješenju određenog problema. Glavna razlika je u tome da se metoda genetskih algoritama oslanja na genetski operator križanja, dok evolucijsko programiranje stavlja naglasak na provođenju mutacije. Genetskim algoritmima se vrši selekcija jedinki prema njihovoj dobroti kako bi križanjem nastali što vrijedniji potomci. Posljedica ovog postupka je da jedinke imaju sve sličniji genetski sastav. Takva uniformnost može dovesti do konvergencije rješenja prema lokalnom optimumu. Stoga, u svrhu nalaženja globalno najboljeg rješenja, se primjenjuje operator mutacije kojim čuvamo raznolikost jedinki tokom iteracija. Takav problem ne postoji u slučaju evolucijskog programiranja s obzirom da se u radu algoritma ne koristi operator križanja.

4.2 Evolucijsko programiranje i genetsko programiranje

Za genetsko programiranje se može ustvrditi da ono, u odnosu na ostale algoritme, izraženije prilagođava rješenje zadatku. To je evolucijski algoritam kojim se traži kompjutorski program koji će izvršavati određeni zadatak. Jedinke unutar početne populacije predstavljaju programi sastavljeni slučajnim odabirom dozovljenih naredbi, izraza i funkcija. Genetskim operacijama se ti programi „prilagođavaju“ kako bi rješavali zadani problem. Metoda evolucijskog programiranja može koristiti istu vrstu reprezentacije jedinki s razlikom da je struktura programa koji predstavljaju jedinke populacije fiksna. Genetskim operacijama se mijenjaju samo numerički parametri.

4.3 Evolucijsko programiranje i evolucijske strategije

Ove dvije tehnike imaju začuđujuće malo razlika s obzirom na njihov nezavisni razvoj. Evolucijske strategije uvijek koriste operator mutacije, dok se, ovisno o problemu, može i ne mora koristiti križanje. Može se reći da evolucijske strategije predstavljaju apstrakciju evolucije na razini individue, odnosno genotipski pristup, i u skladu s tim koriste operator križanja za razliku od evolucijskog programiranja koje predstavlja evoluciju na razini vrste, odnosno fenotipski pristup. Genotip kao biološki pojam označava niz informacija pohranjenih u slijedu DNA. Fenotip predstavlja osobine koje proizlaze iz genotipa poput fizičkih osobina (visina, težina, boja kose), ponašanje, razvoj.
Evolucijsko programiranje tijekom selekcije uobičajeno koristi metodu turnirske selekcije, dok evolucijske strategije koriste deterministčku selekciju u kojoj se najgora rješenja brišu iz populacije izravno prema dobivenoj vrijednosti pri evaluaciji.

5. Zaključak

Svaki problem ima svoje specifičnosti u skladu s kojima se bira algoritam koji će ga najefikasnije riješiti. Zbog takvog prilagođavanja problemu ponekad je teško odrediti granice gdje prestaje jedan algoritam, a započinje drugi. Te granice, na kraju, niti nisu strogo određene. Tako je dozvoljeno raditi brojne preinake s ciljem dobivanja kvalitetnog i brzog rješenja. Evolucijsko programiranje je nastalo na temelju preinaka genetskih algoritama. Na temelju prikazanih informacija nemožemo zaključiti da li je algoritam evolucijskog programiranja bolji ili lošiji od ostalih evolucijskih algoritama. On se jednostavno upotrebljava tamo gdje ga se najbolje može iskoristiti.

6. Literatura

[1)] The Hitch-Hiker's Guide to Evolutionary Computation, A Guide to Frequently Asked Questions, http://faqs.org/faqs/ai-faq/genetic/part2/section-3.html, 2. 11. 2007
[2] Overview of Evolutionary Programming Methods, http://members.aol.com/btluke/evprog.htm, 13.11.2007.
[3] Wikipedia, http://en.wikipedia.org/wiki/Evolutionary_programming, 24.11.2007.
[4] Wikipedia, http://en.wikipedia.org/wiki/Evolution_strategy, 24. 11. 2007.
[5] Wikipedia, http://en.wikipedia.org/wiki/Genetic_algorithms, 24.11. 2007.
[6]Wikipedia http://en.wikipedia.org/wiki/Genetic_programming, 24. 11. 2007.
[4] Intro to GAs, http://lancet.mit.edu/~mbwall/presentations/IntroToGAs/P014.html, 25.11.2007.
[5] Golub, M. Genetski algoritam, prvi dio, http://www.zemris.fer.hr/~golub/ga/ga_skripta1.pdf, 25.11.2007
[6] Duan, Povineli: „Nonlinear modelling: Genetic programming vs. fast evolutionary programming“, Sveučilište Milwuakee, Wisconsin, 1999.
[7] Bell, Alexander: „A Tasteful Example of Evolutionary Programming“, Sveučilište Georgetown, 2007.
[8]Xin Yao, Yong Liu: „Fast evolutionary programming“ Sveučilište New South Wales, 1997.
[9] Technical Difficulties - Hacks - Evolutionary Programming Toolkit, http://www.technical-difficulties.com/hacks/eptk/, 25. 11. 2007.
[11] Julian F. Miller, Peter Thomson: „Cartesian Genetic Programming“, Sveučilište u Birminghamu, 2001.
[12] Lockhead Martin Corporation: „Genetic algorithms and evolutionary programming“, Orlando, 1997.
[13] Bäck, Rudolph, Schwefel: „Evolutionary programming and evolutionary strategies: Similarities and differences“, Sveučilište u Dortmundu, 1993.
[14] Genetic programming Homepage, http://www.genetic-programming.org/, 25. 11. 2007.