SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Objektno orijentirano razvojno okruženje za evolucijsko računanje

Krešimir Đuretec

Voditelj: *doc.dr.sc. Marin Golub*

Zagreb, travanj, 2008.

Sadržaj

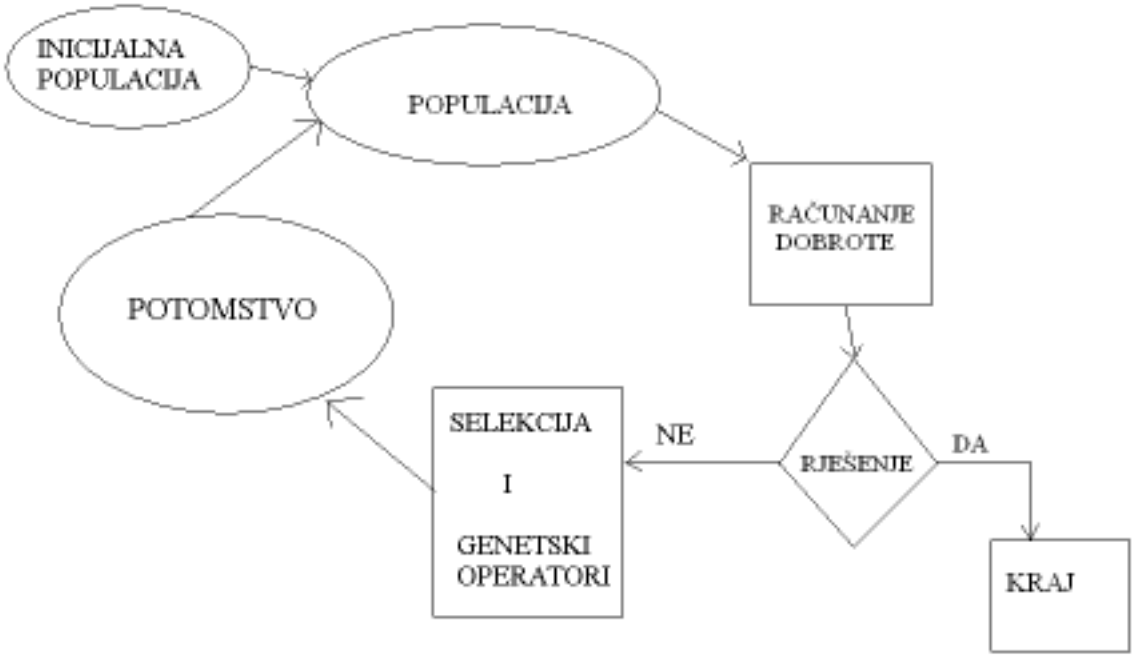
1. Uvod

Računalna znanost , iako mlada znanost , do danas je razvila vrlo veliki broj algoritama koji i na sporijim računalima rješavaju neke složene probleme u vrlo kratkom vremenu. No kako se računalna znanost razvija tako i problemi postaju sve složeniji i ponekad nije vrlo jednostavno pronaći algoritam koji će dati rješenje u realnom vremenu. Za rješavanje takvih problema , kao i mnogo puta do sada, znanstvenici su potražili inspiraciju u prirodi.
1859. godine Charles Darwin je uzdrmao svijet svojom knjigom "O Porijeklu vrsta" (On the Origin of Species). U njoj je objasnio kako prirodne populacije evoluiraju i tako se prilagođavaju životnim uvjetima koje im nameće priroda.
Nekih 120 godina kasnije John Holland je predložio (popularizirao) ,kroz svoj rad "Adaptation in Natural and Artificial Systems'', genetske algoritme. Ovo nije prva pojava primjene evolucije iz prirode u računarstvu. Tako je već Alan Turing pokušao upotrijebiti genetske algoritme u neuronskim mrežama. Također su ranije razvijene i slične metode kao što su evolucijske strategije i evolucijsko programiranje.
Genetski algoritmi se primjenjuju kod neuronskih mreža, optimiranju upita nad bazama podataka, financijama, itd.
Danas postoji vrlo veliki broj metoda koje temelje svoje postupke na evoluciji iz prirode, a genetski algoritmi su jedna od njih.

2. Genetski algoritmi

2.1 Što su genetski algoritmi

Genetski algoritmi su heuristička metoda traženja rješenja. To znači da neće prvo rješenje koje će dati biti točno, nego će procjenjivati moguća rješenja te na temelju procjene odabrati ono najbolje. Da bi generirao moguća rješenja algoritam koristi evolucijske tehnike koje postoje i u prirodi (selekcija, mutacija, križanje).
Rad genetskog algoritma može se jednostavno predočiti slikom :



slika 1. rad genetskog algoritma

Kao što se vidi, algoritam iz populacije traži rješenje računajući dobrotu jedinke. Ako niti jedna jedinka nije dovoljno dobra, nad populacijom se obavljaju genetski operatori i selekcija koji će generirati novu populaciju.

2.2 Jedinka i populacija

U računalu jedinka se može predstaviti kao niz bitova , niz cijelih ili realnih brojeva. Jedinke predstavljaju moguća rješenja zadanog problema.

Populacija je skup svih jedinki. Najčešće je tijekom izvođenja algoritma veličina populacije konstantna. Inicijalna populacija je prva populacija tj. populacija s kojom genetski algoritam počinje rad. Ona je najčešće slučajno generirana.

2.3 Funkcija dobrote

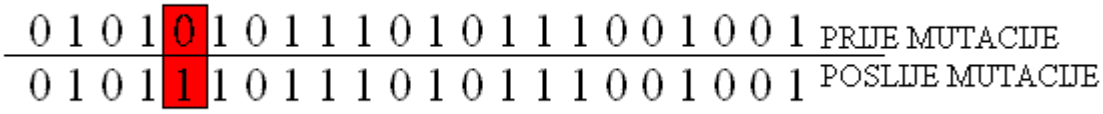
Najvažniji dio genetskih algoritama je funkcija dobrote (funkcija cilja), jer predstavlja ključ selekcije (određuje koje se jedinke odbacuju , a koje ostaju u populaciji) te također predstavlja sami problem koji se rješava. Zadatak funkcije dobrote je odrediti koliko je dobra pojedina jedinka. Što će dobrota biti veća to će jedinka imati veću vjerojatnost preživljavanja. Jedinke s manjom vrijednošću dobrote imati će manju vjerojatnost preživljavanja. Vjerojatnost će biti različita od 0 jer je time omogućeno da se algoritam ne zaustavi u nekom lokalnom maksimumu.

2.4 Unarni operatori (mutacija)

Mutacija je operator koji djeluje nad jednom jedinkom i kao rezultat daje izmijenjenu jedinku. Svrha mutacije je da ubaci novi genetski "materijal" u populaciju i tako onemogući zaustavljanje u prvom lokalnom optimumu koji se nađe. Vjerojatnost mutacije se postavlja na vrlo malu vrijednost (npr. 0.01) jer bi se u suprotnom genetski algoritam pretvorio u algoritam slučajne pretrage prostora rješenja. Postoji više vrsta mutacija ovisno o tipu jedinke nad kojom se mogu izvršavaju.

2.4.1 Mutacija nad bitovima (jednostavna mutacija)

Izvršava se nad jedinkom koja ima binarni prikaz. Parametar mutacije je vjerojatnost *p* da se određeni bit promijeni.



slika 2. jednostavna mutacija

2.4.2 Mutacija nad realnim brojevima

Kada bi se jednostavna mutacija primjenjivala na jedinku koja je predstavljena kao realna vrijednost promjena jednog bita mogla bi dovesti do vrlo velike promjene vrijednosti jedinke. Budući da jedinka nosi informaciju kao realni broj takav slučaj nije koristan. Postoje mutacije koje rade na drugačijem principu nad realnim brojevima, nego jednostavna mutacija nad binarnim prikazom.

2.4.2.1 Uniformna mutacija

Neka je jedinka definirana kao niz realnih brojeva $\langle x_1,...,x_n \rangle$. Primjenom uniformne mutacije jedinka će prijeći u $\langle y_1,...,y_n \rangle$ gdje su x_i i y_i elementi iz $[L_i, U_i]$. Element y_i je uniformno izabran iz danog intervala tj. vjerojatnost biranja bilo kojeg broja iz tog intervala je konstantna.

2.4.2.2 Normalna mutacija

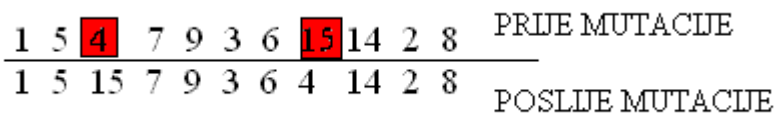
Normalna mutacija je slična uniformnoj ali, biranje broja iz intervala $[L_i, U_i]$ ne podliježe uniformnoj, nego normalnoj (Gaussovoj) razdiobi pa je tako vjerojatnost pojavljivanja broja bliže rubu intervala manja od onog bliže sredini.

2.4.3 Mutacija nad permutacijama

Ponekad je jedinka predstavljena kao permutacija određenog niza brojeva. Primjenom neke od gore navedenih mutacija moglo bi se uništiti to svojstvo (dobije se broj koji ne pripada zadanom nizu). Da se to ne bi dogodilo postoji posebna mutacija koja se primjenjuje nad ovako definiranim jedinkama.

2.4.3.1 Miješajuća (swap) mutacija

Mutacija je vrlo jednostavna i kao parametar ima broj zamjena u kromosomu , a radi na način da slučajno odabere dva gena i zamijeni im mjesta.



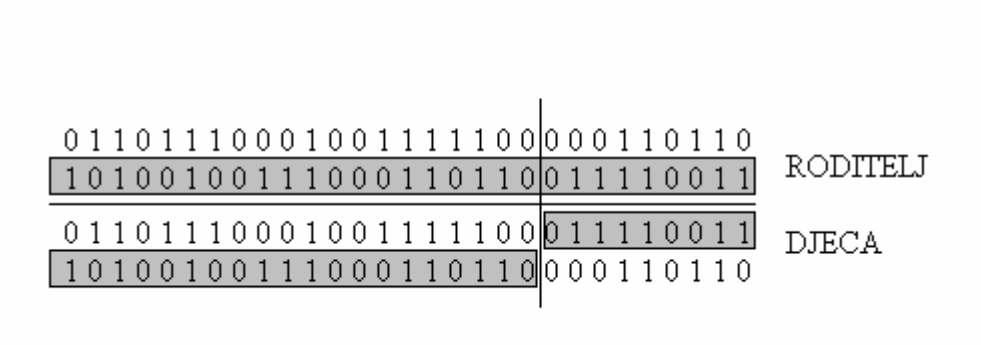
slika 3. miješajuća mutacija

2.5 Binarni operatori (križanje)

Križanje je operator koji uzima dva roditelja te njihovim križanjem nastaju nove jedinke (jedna ili dvije).

2.5.1 Križanje s jednom točkom prekida

Križanje s jednom točkom prekida je najjednostavniji oblik križanja. Uzimaju se dva roditelja te križanjem nastaje jedno ili dvoje djece. Algoritam križanja radi na vrlo jednostavnom principu : izabire se broj između [0, b-1] (b – broj bitova) te se na tom mjestu zamijene repovi roditelja (rep – niz bitova desno od izabranog mjesta).



slika 4. križanje s jednom točkom prekida

2.5.2 Križanje s više točaka prekida

Križanje s N točaka prekida je prošireno križanje s jednom točkom prekida. Umjesto jedne točke izabire s N točaka te se na tim mjestima obavlja zamjena genetskog materijala.

2.5.3 Uniformno križanje

Uniformno križanje je križanje s b-1 točaka prekida. Vjerojatnost da dijete naslijedi svojstvo od jednog roditelja je 50 % (p=0.5).Također je moguće definirati poseban vektor koji će sadržavati za svaki bit posebnu vjerojatnost. Takva vrsta križanja se naziva p-uniformno križanje. U ovom slučaju vjerojatnost p (koja može biti različita za pojedine bitove) govori kolika je vjerojatnost da bude naslijeđen bit od prvog roditelja. Ako prilikom križanja nastaju dva djeteta tada drugo dijete nastaje na suprotan način od prvog (ovdje se ne može reći da je drugo dijete negacija prvog jer ako oba roditelja imaju isti bit tada će i djeca na tom mjestu imati isti bit).

2.5.4 Segmentno križanje

Segmentno križanje je križanje kod kojeg se slučajno bira broj točaka i mjesto prekida .

2.6 Selekcija

Uloga selekcije u algoritmu je da sačuva dobre jedinke unutar populacije (i omogući operatorima da stvore još bolje) i da odbaci loše jedinke.

2.6.1 Jednostavna selekcija

Selekcija pojedine jedinke ovisi o njenoj dobroti tj. koliko je dobra pojedina jedinka u usporedbi s cijelom populacijom.

$$p_i = \frac{dobrota(jedinka_i)}{\sum_{j=1}^{VEL\_POP} dobrota(jedinka_j)}$$

Nedostatci:
Ako postoje jedinke koje su puno bolje od ostatka , može se desiti da će preuzeti cijelu populaciju (prerana konvergencija).
Kad sve jedinke imaju približno jednaku dobrotu selekcija se pretvara u uniformnu selekciju i dobrota u tom slučaju nema veliki značaj.
Ako je potrebno normalizirati dobrotu jedinki tada će se promijeniti vrijednosti vjerojatnosti svih jedinki i više neće biti isti uvjeti selekcije kao prije.

2.6.2 Turnirska selekcija

Parametar turnirske selekcije je broj *k* (*k*>2) . Radi tako da iz populacije uzima slučajno *k* jedinki i od njih bira najbolju te ju seli u novu populaciju gdje će se vršiti genetski operatori. Selekciju određuje i vjerojatnost (p) da najbolja jedinka bude izabrana. Ako je p = 1 tada se govori o deterministčkoj turnirskoj selekciji , a ako je p< 1 tada o stohastičkoj turnirskoj selekciji.

2.6.3 Slučajna (random) selekcija

Turnirska selekcija s vrijednošću *k*=1 se pretvara u slučajnu selekciju. Takav tip selekcije daje najlošije rezultate i uglavnom se ne koristi.

2.7 Uvijet završetka

Uvjet završetka definira kraj izvođenja genetskog algoritma. Uvjet je najčešće maksimalan broj generacija koje algoritam smije dostići, no radi optimizacije izvođenja mogući su i drugačiji uvjeti , npr. kad dobrota jedinke dostigne određenu razinu ili kada ne dolazi do promjene u populaciji nakon nekoliko generacija.

3. EO radno okruženje za evolucijske algoritme

EO (Evolving Objects) radno okruženje je objektno orijentirano okruženje za evolucijsko računanje napisano u C++ jeziku. Okruženje podržava niz objekata za evolucijsko računanje. Upotreba objekata je vrlo jednostavna.
EO okruženje je besplatno i ako trenutno ne podržava nešto nama potrebno, nakon što upoznamo njegovu strukturu, možemo izvesti novi objekt koji će rješavati naš problem i tako doprinijeti razvoju radnog okruženja.

3.1 Jedinka

EO razvojno okruženje pruža podršku za sve tri vrste jedinki(niz bitova, niz cijelih ili realnih brojeva) .
Prilikom definiranja jedinke potrebno je definirati i tip podatka koji predstavlja njezinu dobrotu.

```
// definira se jedinka koja se sastoji od  niza bitova i ima
//vrijednost dobrote tipa double (realni broj)
eoBit<double> jedinka ;


// jedinka tipa int i dobrote tipa int – ovo trenutno nije
// podržano
eoInt<int> jedinka;

//jedinka tip real (realni broj) i dobrote tipa double
eoReal<double> jedinka;
```

Zbog jednostavnosti programiranja biti će uputno definirati novi tip podatka npr.

```
typedef eoBit<double> tip_jedinke;
```

3.2 Populacija

Da bi smo objedinili naše jedinke u populaciju moramo definirati populaciju :

```
eoPop<tip_jedinke> ime_populacije;
```

Ovo je zapravo razred koji će u vektoru čuvati sve jedinke, no uz to sadrži veliki broj metoda koje nam mogu dati korisne informacije o populaciji (npr. i-ta najbolja jedinka). Kako je razred *eoPop* naslijedio razred *vector* iz standardne biblioteke predložaka na populaciju je moguće primijeniti i metode razreda *vector.*

Primjeri nekoliko operacija nad populacijom :

```
//dodavanje jedinke u populaciju
ime_populacije.push_back(jedinka);

//sortiranje populacije
ime_populacije.sort( );

//ispis populacije
cout<<ime_populacije;
```

3.3 Funkcija dobrote

Računanje dobrote ovisi o problemu koji se rješava i to će programer uvijek sam morat implementirati. Najčešće je to neka kraća funkcija koja kao parametar ima jedinku, a vraća dobrotu te jedinke (napomena: vraća onaj tip podatka koji je definiran kao dobrota u deklaraciji jedinke). Argument funkcije mora biti referenca na konstantni objekt i funkcija dobrote ne dozvoljava mijenjanje istog.
npr. funkcija bi mogla izgledati ovako

```
double  dobrota( const jedinka & x) {
. . .

    // ovdje se nešto računa

. . .
return varijabla;
}
```

Kako EO razvojno okruženje ne radi direktno s funkcijama nego s funkcijskim objektima funkciju dobrote se mora pretvoriti (ubaciti) u funkcijski objekt koji će se kasnije koristiti.

```
//ubacivanje funkcije u funkcijski objekt
eoEvalFuncPtr<tip_jedinke> ime_funkcijskog_objekta(
pokazivač_na_funkciju);
```

3.4 Generiranje slučajne (random) populacije

Da bi se generirala početna populacija potrebno je stvoriti jedinke. Kod populacije se zahtijeva određena slučajna razdioba. Razvojno okruženje nudi "random number generator" (rng) objekt koji je već ugrađen u okruženje.
Tako ako se želi napuniti jedinku npr. u binarnom prikazu s 1 i 0 gdje je vjerojatnost pojave 1 ili 0 50% koristit će se rng.flip( ) metoda pa bi sljedeći programski odsječak napravio upravo željeno :

```
// definira se dobrota kao double tip ,iako se ovdje ne
// koristi
eoBit<double> jedinka ;

// definira se broj prema kojem će generator generirati slučajne
// brojeve
rng.reseed(time(NULL));

for (int i=0; i<VELICINA_JEDINKE; i++) {

//napuni jedinku s 1 i 0
   jedinka.push_back(rng.flip( ))
}
```

Deklaracije nekih razdiobi brojeva :

flip

bool eoRng::flip (double vjerojatnost=0.5)

vjerojatnost -predstavlja vjerojatnost pojavljivanja 1 i ako se ne navede pretpostavljena je 0.5.

uniform

double eoRng::uniform (double m = 1.0)

m – predstavlja interval nad kojim će se obaviti uniformna razdioba brojeva $[0, m)$. Pretpostavljena vrijednost je 1.0.

Generiranje slučajne populacije je moguće obaviti i na drugi način. Razvojno okruženje podržava objekte koji mogu stvoriti čitavu slučajnu populaciju.

Programski odsječak koji će generirati jednu random populaciju :

```
// definira se tip jedinke
typedef eoReal <double> jedinka;

//...

rng.reseed(time(NULL));

eoEvalFuncPtr<jedinka> eval( funkcija_dobrote );

// definira se generator koji će generirati uniformno brojeve
// -10 i 10
eoUniformGenerator<double> generator(-10.0,10.0);

// određuje se veličina jedinki u našem slučaju 5
eoInitFixedLength<jedinka> random (5, generator);

// stvara se populacija veličine 10
eoPop<jedinka> pop(10,random);

// računa se funkcija dobrote pojedinih jedinki
apply <jedinka>(eval,pop);
```

3.5 Unarni operator (mutacija)

3.5.1 Jednostavna mutacija

Parametar koji određuje mutaciju je *vjerojatnost* promjene određenog bita (možda bi bolje bilo rečeno da se mijenja svaki x-ti bit).

Definiranje jednostavne mutacije nad bitovima :

```
// definira se mutacija s parametrom vjerojatnost
eoBitMutation<tip_jedinke> mutacija( vjerojatnost )
```

konstruktor klase

```
template < class Chrom >
eoBitMutation < Chrom >::eoBitMutation(const double & _rate=0.01, bool _normalize = false )
```

_rate - vjerojatnost promjene bita (0.01 pretpostavljena vrijednost) npr. 0.01 znači da se će se promijeniti svaki 100-ti
_normalize - true ili false - ako je true tada će _rate biti promijenjen u - rate/jedinka.size() inače ostaje _rate

Također je definiran operator () pa je moguće obaviti mutaciju nad jednom jedinkom

```
// definira se tip podatka jedinka
typedef eoBit<double> jedinka ;

//.....

// definira se nova jedinka v
jedinka v;

//.....

// definira se mutacija
eoBitMutation<jedinka> mutacija(0.1);

// primjenjuje se mutacija, vraća vrijednost true ako se je nešto
// promijenilo inače false
mutacija(v);
```

3.5.2 Uniformna mutacija

Uniformna mutacija radi mutaciju nad jedinkom u granicama [jedinka(i) - epsilon, jedinka(i) + epsilon]

konstruktor

```
template < class EOT >
eoUniformMutation< EOT >::eoUniformMutation(const double &_epsilon, const double
&_p_change=1.0)
```

_epsilon - zadane granice
_p_change - vjerojatnost promjene (pretpostavljena vrijednost =1.0)

3.5.3 Normalna mutacija

konstruktor

```
template < class EOT >
eoNormalMutation< EOT >::eoNormalMutation(const double &_sigma, const double &_p_change=1.0)
```

_sigma – granice mutacije (kao i za uniformnu mutaciju)
_p_change - vjerojatnost promjene (pretpostavljena vrijednost = 1.0)

3.6 Binarni operator (križanje)

3.6.1 Križanje s jednom točkom prekida (jednostavno križanje)

Definiranje jednostavnog križanja :

```
eo1PtBitXover<tip_jedinke> krizanje;
```

3.6.2 Križanje s N točaka prekida

Omogućuje križanje dvije jedinke na dva ili više mjesta

```
// definira se križanje koje će križati dvije jedinke na 4 mjesta
eoNPtsBitXover<tip_jedinke> krizanje(4);
```

3.6.3 Uniformno križanje

```
eoUBitXover<tip_jedinke> krizanje
```

3.6.4 Segmentno križanje

```
eoSegmentCrossover<tip_jedinke> krizanje;
```

3.7 Selekcija

3.7.1 Jednostavna selekcija (roullete wheel)

```
eoProportionalSelect<tip_jedinke> selekcija;
```

3.7.2 Deterministička turnirska selekcija

Definira se turnirska selekcija gdje je ***veličina*** veličina turnirske selekcije.

```
eoDetTournamentSelect<tip_jedinke> selekcija(veličina);
```

Napomena : veličina ≥ 2 .

3.7.3 Stohastička turnirska selekcija

```
eoStochTournamentSelect<tip_jedinke> selekcija(postotak);
```

Napomena :postotak $\in [0.5,1.0]$.

3.7.4 Slučajna selekcija (random select)

```
eoRandomSelect<tip_jedinke> selekcija;
```

3.8 Trajanje evolucije

Da bi se definiralo koliko dugo će se izvršavati evolucijske operacije može se definirati maksimalan broj generacija prije završetka algoritma.

```
eoGenContinue<tip_jedinke> trajanje(MAX_BROJ_GENERACIJA)
```

3.9 Algoritmi

3.9.1 SGA (Simple genetic algorithm)

Jednostavni genetski algoritam predložen od J.Hollanda .
Algoritmu se kao parametar zadaje selekcija, križanje, postotak(učestalost) križanja, mutacija , postotak (učestalost) mutacije, funkcija dobrote (pretvorena u funkcijski objekt) i objekt koji definira uvjet završetka rada algoritma.

```
eoSGA< tip_jedinke > algoritam (selekcija, krizanje, CROSS_RATE,
mutacija, MUT_RATE, ime_funkcijskog_objekta,trajanje) ;

// izvršavanje algoritma nad populacijom
algoritam (populacija);
```

3.9.2 EasyEA

Da bismo omogućili ugradnju neke druge metode zamjene (osim generacijske zamjene koja je ugrađene u SGA) koristit ćemo EasyEA . Postoji veći broj konstruktora pa su i parametri različiti. Jedan od konstruktora prihvaća sljedeće parametre: uvjet nastavka (ili kontrolna točka , objašnjeno kasnije) , funkcija dobrote (objekt) ,selekcija, transformacija (križanje i mutacija spojeni u jedno) i zamjena.

```
eoEasyEA<tip_jedinke> algoritam (checkpoint,
ime_funkcijskog_objekta, selekcija, transformacija, zamjena);
```

4. Napredne mogućnosti EO radnog okruženja

4.1 Kombiniranje više operatora

Unutar jednog programa moguće je koristiti više istovrsnih operatora (mutacija, križanja).

4.1.1 Kombiniranje više mutacija

Da bi se uključilo više mutacija unutar jednog programa koristit će se zato definirani objekt koji će te mutacije objediniti u jedan objekt i po principu "roullette wheel" birati određenu mutaciju koju će izvršiti u određenom trenutku.

Konstruktor

```
template < class EOT >
eoPropCombinedMonOp::eoPropCombinedMonOp(eoMonOp &operator, const double _vjerojatnost)
```

operator - vrsta mutacije koju se želi ugraditi u objekt
_vjerojatnost - vjerojatnost da ta mutacija bude odabrana prilikom izvođenja mutacije nad jedinkom

Za dodavanje više mutacija postoji metoda :

```
template < class EOT >
eoPropCombinedMonOp::add(eoMonOp &operator, const double _vjerojatnost, bool _kontrola=false)
```

operator - vrsta mutacije koju se želi ugraditi u objekt
_vjerojatnost - vjerojatnost da ta mutacija bude odabrana prilikom izvođenja mutacije nad jedinkom
_kontrola - ako je false tada ostavlja vjerojatnosti onakve kakve smo ih zadali , a ako je true tada računa relativnu vjerojatnost

Definiranje objekta koji će sadržavati dvije mutacije :

```
// definira se mutacija jednog bita
eoBitMutation <tip_jedinke> mutacija1(0.01);

//definira se mutacija točno 5 bitova
eoDetBitFlip<tip_jedinke> mutacija2(5);

// definira se objekt mutacija
eoPropCombinedMonOp< tip_jedinke > mutacija(mutacija1,
vjerojatnost1);

// dodaje se druga mutacija , parametar true je dovoljno dodati
//  prilikom dodavanja zadnje mutacije
mutacija.add(mutacija2, vjerojatnost2, true);
```

4.1.2 Kombiniranje više križanja

Postupak uključivanja više križanja unutar jednog programa je identičan postupku uključivanja više mutacija. Razlikuju se samo u tipu objekta koji sadrži operatore i odlučuje koji će koristiti.

Konstruktor :

template < class EOT >
eoPropCombinedQuadOp::eoPropCombinedQuadOp(eoMonOp &operator, const double _vjerojatnost)

operator - vrsta križanja koju se želi ugraditi u objekt
_vjerojatnost - vjerojatnost da to križanje bude odabrano prilikom izvođenja križanja

Za dodavanje više križanja postoji metoda :

template < class EOT >
eoPropCombinedQuadOp::add(eoQuadOp &operator, const double _vjerojatnost, bool _kontrola=false)

operator - vrsta križanja koju se želi ugraditi u objekt
_vjerojatnost - vjerojatnost da to križanje bude odabrano prilikom izvođenja križanja
_kontrola - ako je false tada ostavlja vjerojatnosti onakve kakve smo ih zadali , a ako je true tada računa relativnu vjerojatnost

Definiranje objekta koji će sadržavati tri križanja :

```
eo1PtBitXover <tip_jedinke> krizanje1;

eoUBitXover <tip_jedinke> krizanje2;

eoNPtsBitXover <tip_jedinke> krizanje3(5);

eoPropCombinedQuadOp <tip_jedinke> krizanje(krizanje1,
vjerojatnost1);

krizanje.add(krizanje2, vjerojatnost2);

krizanje.add(krizanje3, vjerojatnost3, true);
```

4.1.3 Spajanje križanja i mutacije u jedan objekt

Kako će se vidjeti kasnije križanje i mutaciju je potrebno objediniti unutar jednog objekta.

Konstruktor:

template<class EOT>
eoSGATransform < EOT >::eoSGATransform(eoQuadOp <EOT> &krizanje, double _vjerojatnost_kr, eoMonOp<EOT> & mutacija, double vjerojatnost_mut)

Križanje i mutacija se može objediniti na sljedeći način :

```
eoSGATransform<tip_jedinke > operator(krizanje,
VJEROJATNOST_KRIZANJA, mutacija, VJEROJATNOST_MUTACIJE);
```

4.2 Kombiniranje više kriterija zaustavljanja

Uvjet zaustavljanja može biti maksimalan broj generacija. Također može se definirati još neke uvjete zaustavljanja kao što su : zaustavljanje kada se ne pojavljuje poboljšanje nakon određenog broja generacija , ili dok se ne pojavi jedinka s maksimalnom vrijednošću dobrote.

```
// već objašnjeni uvjet zaustavljanja kod dostizanja max broja
// generacija
eoGenContinue <tip_jedinke> genCont(MAX_GEN);

// prolazak kroz MIN_GEN broj generacija , a nakon toga
// zaustavljanje kada prođe STEADY_GEN broj generacija bez
// promjene
eoSteadyFitContinue <tip_jedinke> steadyCont(MIN_GEN, STEADY_GEN);

// zaustavljanje kad jedinka dosegne zadanu dobrotu
eoFitContinue <tip_jedinke> fitCont(0);

// kombiniranje zadanih uvjeta
eoCombinedContinue <tip_jedinke> continuator(genCont);
continuator.add(steadyCont);
continuator.add(fitCont);
```

4.3 Definiranje selekcije i zamjene

Da bi se dobila potpuna selekcija određenu selekciju (definiranu u poglavlju 3.7) mora se ugraditi u objekt koji će kao parametar (uz selekciju) imati i postotak populacije koji se selektira.

```
// definiramo našu selekcija
eoDetTournamentSelect<tip_jedinke> selekcija1(3);

//ugrađujemo ju u objekt s 100% izborom populacije
eoSelectPerc<tip_jedinke> selekcija(selekcija1);
```

Ako se ne želi ograničiti samo na generacijsku zamjenu (ugrađena u SGA) može se definirati i neka druga zamjena (ponuđena u okruženju). Također može se definirati i elitizam (čuvanje najboljih jedinki) , objediniti zamjenu i elitizam u jedan objekt te tako dobiti potpunu zamjenu koja će čuvati najbolje jedinke.

```
// definira se zamjena
eoGenerationalReplacement<tip_jedinke> genzamjena;

//definira se elitizam i u njega ugrađuje generacijska zamjena
eoWeakElitistReplacement<tip_jedinke> zamjena(genzamjena);
```

4.4 Parser

Da se ne bi moralo uvijek prilikom mijenjanja jednog ili više parametara algoritma prevoditi program, razvojno okruženje podržava vlastiti parser. Upotreba parsera je vrlo jednostavna i nadasve korisna jer nakon što se napiše program iz jedne datoteke mogu se učitavati parametri. Što je još zanimljivije imena varijabli parametara definira programer te si na taj način dodatno olakšava korištenje parsera.

Definiranje pravila parsiranja datoteke :

Konstruktor klase eoValueParam:

template < class Value Type >
eoValueParam < Value Type >::eoValueParam(ValueType _defaultValue, std::string _longName, std::string _description="No description", char _shortHand=0, bool _required=false)

_defaultValue – pretpostavljena vrijednost parametra , biti će upotrijebljena ako se ne navede druga
_longName - dugo ime varijable , omogućuje navođenje ime_varijable = vrijednost
_description - opis varijable , biti će naveden ako se zatraži pomoć
_shortHand - kratko ime varijable , omogućuje navođenje v= vrijednost
_required - ako je true tad ta varijabla mora postojati u našoj datoteci

Jednostavan programski odsječak :

```
// definira se parser eoParser parser(argc, argv);

// definira se objekt koji će sadržavati veličinu jedinke ,
// njegovu default vrijednost dugo i kratko ime
eoValueParam vecVelicina(8, "vecVel", "Veličina jedinke",'V');

// stvara se par vrijednost – ime
// prilikom korištenja help naredbe omogućuje grupni ispis
parser.processParam( vecVelicina, "Predstavljanje jedinke" );

// dodjeljuje se vrijdnost određenoj varijabli koja će se
// koristiti
unsigned vecSize = vecVelicina.value();
```

4.5 Kontrolne točke

Kontrolne točke omogućuju obavljanje određenih akcija (računanje statistike) unutar svake generacije. Kontrolna točka je implementirana kao objekt koji sadrži uvjete zaustavljanja, statistiku, ispis na ekran i ažuriranje. Te operacije (koje su definirane) izvršava za svaku generaciju .

Da bi se generirala određena statistika mora se definirati potrebne objekte :

```
// definira se kontrolna točka i ugrađuju se uvjeti završetka
eoCheckPoint<tip_jedinke> kontrolna_tocka(continuator);

// izdvajat će vrijednost najbolje jedinke svake generacije
eoBestFitnessStat<tip_jedinke> najbolji;

// računat će prosjek i standardnu devijaciju svake generacije
eoSecondMomentStats<tip_jedinke> par;

//dodavanje u kontrolnu točku
kontrolna_tocka.add(najbolji);
kontrolna_tocka.add(par);
```

5. Programski primjer (Problem ruksaka)

5.1 Definicija problema

Pljačkaš pljačka dućan. U dućanu je N proizvoda i svaki proizvod ima svoju cijenu i volumen. Pljačkaš ima ruksak u koji stane volumen V. Želja pljačkaša je da ukrade što vredniju robu , a da mu ona pritom stane u ruksak.
Budući da je u pitanju maksimizacija (vrijednosti ukradene robe) za rješavanje ovog problema može se koristiti genetički algoritam.

5.2 Razrada problema

Neka su proizvodi u dućanu predstavljeni kao niz elemenata. Svaki element se sastoji od dvije vrijednosti cijene i volumena proizvoda .

PROIZVOD	1.	2.	3.	4.
VOLUMEN	50	30	65	15
VRIJEDNOST	40	20	45	35

slika 5. proizvodi u dućanu

Odmah se uočava da će se kao prikaz rješenja odabrati niz bitova duljine N (broj elemenata u dućanu) . 1 će predstavljati da je taj element stavljen u ruksak ,a 0 da nije.

5.3 Kod

```
#include <fstream>
#include <iostream>
#include <stdexcept>
#include<cstdio>
#include<vector>
#include <eo>

#include<ga.h>
using namespace std;

vector<pair <int,int> > niz(100);
int volumen_ruksak;
unsigned broj;
typedef eoBit<double>  jedinka;




double dobrota(const jedinka & _niz)
{
  int ukupna_cijena=0;
  int ukupni_volumen=0;
  int pomocni;
  for (int i = 0; i < broj; i++) {
      if (_niz[i]) {
            ukupna_cijena+=niz[i].first;
            ukupni_volumen+=niz[i].second;
```

```
            }

      }

  if (ukupni_volumen<=volumen_ruksak)

      return (double)ukupna_cijena;

  else

      return 0;

}



void main_function(int argc, char **argv)

{

eoParser parser (argc,argv);



unsigned velicina_jedinke=broj;



eoValueParam<unsigned int> velicinapop(10,"velicina_populacije","Velicina
populacije",'P');

parser.processParam(velicinapop,"Populacija");

unsigned velicina_populacije =velicinapop.value();



eoValueParam<unsigned int>
velicinaturnir(3,"turnirska_selekcija","Velicina turnirske
selekcije",'T');

parser.processParam(velicinaturnir,"Selekcija");

unsigned turnirska_selekcija=velicinaturnir.value();



eoValueParam<unsigned int> velicinaN(2,"velicina_N","Broj tocaka N
krizanja",'N');

parser.processParam(velicinaN,"Krizanje");

unsigned Nkrizanje =velicinaN.value();
```

```
eoValueParam<double>
vjerojatnost1kriz(0.8,"vjerojatnost_1_krizanja","Vjerojatnost 1
krizanja",'V');

parser.processParam(vjerojatnost1kriz,"Krizanje");

double vjer1krizanje =vjerojatnost1kriz.value();



eoValueParam<double>
vjerojatnostUkriz(0.2,"vjerojatnost_U_krizanja","Vjerojatnost uniformnog
krizanja",'U');

parser.processParam(vjerojatnostUkriz,"Krizanje");

double vjerUkrizanje=vjerojatnostUkriz.value();



eoValueParam<double>
vjerojatnostNkriz(0.4,"vjerojatnost_N_krizanja","Vjerojatnost_N_krizanja"
,'K');

parser.processParam(vjerojatnostNkriz,"Krizanje");

double vjerNkrizanje =vjerojatnostNkriz.value();



eoValueParam<double>
vjerojatnostBitmut(0.01,"vjerojatnost_bit_mutacije","Vjerojatnost
jedonstavne mutacije",'M');

parser.processParam(vjerojatnostBitmut,"Mutacija");

double vjerBitMutacije =vjerojatnostBitmut.value();



eoValueParam<double>
vjerojatnostpojBitmut(0.8,"vjerojatnost_pojave_bit_mutacije","Vjerojatnos
t pojave jedonstavne mutacije",'A');

parser.processParam(vjerojatnostpojBitmut,"Mutacija");

double vjerBmutacije =vjerojatnostpojBitmut.value();



eoValueParam<double>
vjerojatnostpoj1mut(0.2,"vjerojatnost_pojave_1_mutacije","Vjerojatnost
pojave mutacije tocno jednog bita",'B');

parser.processParam(vjerojatnostpoj1mut,"Mutacija");

double vjer1mutacije =vjerojatnostpoj1mut.value();
```

```
eoValueParam<double>
vjerojatnostpojkriz(0.6,"vjerojatnost_pojave_krizanja","Vjerojatnost
pojave opcenitog krizanja",'C');

parser.processParam(vjerojatnostpojkriz,"Krizanje");

double vjer_krizanje=vjerojatnostpojkriz.value();



eoValueParam<double>
vjerojatnostpojmut(0.2,"vjerojatnost_mutacije","Vjerojatnost pojave
opcenite muatacije",'D');

parser.processParam(vjerojatnostpojmut,"Mutacija");

double vjer_mutacije=vjerojatnostpojmut.value();



eoValueParam<unsigned int> velicinamaxgen(80,"max_generacija","Veličina
max generacija",'E');

parser.processParam(velicinamaxgen,"Generacije");

unsigned max_generacija = velicinamaxgen.value();



eoValueParam<unsigned int> velicinamingen(10,"min_generacija","Veličina
min generacija",'F');

parser.processParam(velicinamingen,"Generacije");

unsigned min_generacija = velicinamingen.value();



eoValueParam<unsigned int> velicinamirno(10,"mirnoca","Broj isti
nepoboljšanih generacija",'G');

parser.processParam(velicinamirno,"Generacije");

unsigned mirnoca = velicinamirno.value();



eoValueParam<double> velicinadobrota(1000,"dobrota","Maksimalna moguća
dobrota",'H');

parser.processParam(velicinadobrota,"Generacije");

double naj_dobrota =velicinadobrota.value();
```

```
    if (parser.userNeedsHelp())
      {
        parser.printHelp(cout);
        exit(1);
      }



rng.reseed(time(NULL));
eoEvalFuncPtr<jedinka> eval1( dobrota );
eoEvalFuncCounter<jedinka> eval(eval1);
eoUniformGenerator<bool> generator;
eoInitFixedLength<jedinka> random(velicina_jedinke,generator);
eoPop<jedinka> populacija(velicina_populacije,random);
apply<jedinka> (eval,populacija);
populacija.sort();
cout<<populacija;

eoDetTournamentSelect<jedinka> turnir(turnirska_selekcija);
eoSelectPerc<jedinka> selekcija(turnir);
eoGenerationalReplacement<jedinka> zamjena;
eo1PtBitXover<jedinka> xover1;
eoUBitXover<jedinka> xoverU;
eoNPtsBitXover<jedinka> xover2(Nkrizanje);
eoPropCombinedQuadOp<jedinka> xover(xover1, vjer1krizanje);
xover.add(xoverU, vjerUkrizanje);
xover.add(xover2, vjerNkrizanje, true);



 eoBitMutation<jedinka>  mutationBit(vjerBitMutacije);
```

```
 eoDetBitFlip<jedinka> mutationOneBit;

 eoPropCombinedMonOp<jedinka> mutation(mutationBit, vjerBmutacije);

 mutation.add(mutationOneBit, vjer1mutacije, true);


 eoSGATransform<jedinka> transform(xover, vjer_krizanje, mutation,
vjer_mutacije);




 eoGenContinue<jedinka> genCont(max_generacija);

 eoSteadyFitContinue<jedinka> steadyCont(min_generacija, mirnoca);

 eoFitContinue<jedinka> fitCont(naj_dobrota);

 eoCombinedContinue<jedinka> continuator(genCont);

 continuator.add(steadyCont);

 continuator.add(fitCont);


 eoCheckPoint<jedinka> checkpoint(continuator);

 eoValueParam<unsigned> generationCounter(0, "Gen.");

 eoIncrementor<unsigned> increment(generationCounter.value());

 checkpoint.add(increment);

 eoBestFitnessStat<jedinka> bestStat;

 eoSecondMomentStats<jedinka> SecondStat;

 checkpoint.add(bestStat);

 checkpoint.add(SecondStat);

 eoStdoutMonitor monitor(false);

 checkpoint.add(monitor);

 monitor.add(generationCounter);

 monitor.add(eval);

 monitor.add(bestStat);

 monitor.add(SecondStat);
```

```
 eoEasyEA<jedinka> gga(checkpoint, eval, selekcija, transform, zamjena);
 gga(populacija);
 populacija.sort();
 cout<<populacija;
populacija.clear();
}

int main(int argc, char **argv)
{
srand(time(NULL));

int i,vrijednost,volumen,j;
FILE *tok=fopen("podaci.txt","r");

while(1) {
    fscanf(tok,"%d %d",&broj,&volumen_ruksak);
    if (broj==0) break;
    for (i=0; i<broj; i++) {
          fscanf(tok ,"%d %d",&vrijednost,&volumen);
          niz[i].first=vrijednost;
          niz[i].second=volumen;
          }
    try
    {
        main_function(argc, argv);
    }
    catch(exception& e)
    {
```

```
        cout << "Exception: " << e.what() << '\n';

    }

    }

fclose(tok);

return 1;

}
```

Uz kod su potrebne još dodatne dvije datoteke , "podaci.txt" i datoteka proizvoljnog imena recimo ("config.txt") u kojoj se nalaze parametri za naš algoritam.

6. Zaključak

Genetski algoritmi su postali vrlo korisno oruđe za rješavanje niz problema koji su vrlo složeni i ne postoji za njihovo rješavanje efikasan algoritam. Snaga genetskih algoritama je u tome što su vrlo jednostavni i primjenjivi na niz problema . Jednostavnost temelje na tome što je njihove operatore vrlo lagano implementirati (točno se zna kako se koji operator ponaša), a ideja samih algoritama je svima razumljiva. S druge strane genetski algoritmi rade na vrlo velikim populacijama i vrlo velikom broju generacija što će tražiti određenu procesorsku snagu. Zato je tu potrebna određena vještina jer i jedan krivo podešeni parametar može usporiti rad algoritma.
Danas postoje mnoga razvojna okruženja za genetske algoritme. Iako mnogi znanstvenici smatraju da korištenje razvojnih okruženja nije dobro, jer ne tjera programera na razmišljanje i programer nema potpunu kontrolu nad programom, razvojno okruženje može znatno olakšati rad. EO razvojno okruženje je jedno takvo okruženje koje omogućuje implementaciju određenog algoritma u svega nekoliko redaka programa. Tako ako se želi proučavati genetske algoritme s matematičke strane (provoditi statistiku i sl.) ne mora se programirati svaku komponentu algoritma i tako gubiti dragocjeno vrijeme.

7. Literatura

1. Golub, Marin, Genetski algoritam: Prvi dio , verzija 2.3 -27. rujna 2004., Zagreb,2004
2. Eiben A. E. , Smith J. E. , Introduction to Evolutionary Computing , Springer-Verlag,Berlin,2003.
3. *EO Evolutionary Computation Framework* , http://eodev.sourceforge.net/ , 15.travnja 2008.
4. *Tutorial EO ,* http://eodev.sourceforge.net/eo/tutorial/html/eoTutorial.html, 15 travnja 2008
5. *Genetic algorithm – Wikipedia, http://en.wikipedia.org/wiki/Genetic_algorithm,* 15. travnja 2008.

8. Sažetak

Genetski algoritmi su heuristička metoda traženja rješenja, koja imitira evolucijski proces iz prirode.
Algoritam se može podijeliti na nekoliko cjelina koje su neophodne za njegov rad :

- predstavljanje jedinke u računalu
- definiranje funkcije dobrote (ovo je najvažniji dio algoritma)
- definiranje uvjeta završetka algoritma
- definiranje selekcije
- definiranje genetskih operatora (mutacija i križanje)
- definiranje parametara algoritma (veličina populacije, vjerojatnost mutacije, vjerojatnost križanja,…)

Parametri algoritma su također vrlo važna karika algoritma jer koliko su dobro definirani parametri toliko će dobro raditi i sam algoritam.
EO radno okruženje je objektno orijentirano okruženje napisano u C++. Omogućuje pisanje vrlo jednostavnog i razumljivog koda. Jednostavnost temelji na tome što se pisanje programa temelji na pisanju jedne (ili nekoliko naredbi) za svaku cjelinu algoritma. Također podržava parser i generiranje statistike i tako omogućuje pisanje kratkih, ali po mogućnostima vrlo jakih programa.