1. UVOD

Zadnjih desetak godina pojačano je proučavanje i razvoj *genetskih algoritama*. Počelo ih se sve više primjenjivati u raznim područjima kao što su neuronske mreže, pri traženju najkraćeg puta, problemu raspoređivanja procesa, traženju maksimuma funkcija, itd.

Prije deset-ak godina postojali su UNIX© strojevi i PC-i. Unix strojevi su bili relativno skupi ali su imali dobar OS i većina korisnika je radila u komandnom promptu. Porastom snage Intel© procesora i izlaskom 32-bitnih Windowsa©, Wintel platforma polako preuzima dio gdje su se prije koristile Unix radne stanice. Poboljšavanjem alata za vizualno programiranje (Visual Basic©, Borland Delphi©) razvoj programa je postao lakši i brži.

Budućи da su Windowsi grafičko okruženje, prikaz rezultata genetskog algoritma moguće je lakše i intuitivnije predočiti korisniku. Također i promjena parametara je lakša putem izbrnika i *edit boxova* nego da se ručno mijenjaju parametri u ulaznoj datoteci, kao što je to bio slučaj kod verzije programa na Unix-u. Prvo je trebalo u uređivaču teksta promijeniti parametre, zatim pozvati program iz komandne linije i konačno opet u uređivaču teksta proučavati rezultate.

2. GENETSKI ALGORITAM

2.1. Općenito

Genetski algoritam je heuristička metoda optimiranja koja imitira evolucijski proces. Analogija između genetskog algoritma i evolucijskog procesa očituje se u procesu selekcije i genetskim operatorima. U prirodi jedinka koja je najbolje prilagođena ima najveću vjerojatnost preživljavanja i parenja, a time i prenošenja svog genetskog materijala na potomke. Selekcijom se odabiru dobre jedinke koje se prenose u sljedeću populaciju, a manipulacijom genetskog materijala stvaraju se nove jedinke. Takav ciklus selekcije, reprodukcije i manipulacije genetskim materijalom ponavlja se sve dok nije zadovoljen uvjet zaustavljanja evolucijskog procesa.

Snaga genetskog algoritma leži u činjenici da su oni sposobni odrediti položaj globalnog optimuma u prostoru s više lokalnih ekstrema. Klasične deterministčke metode će se kretati uvijek prema lokalnom ekstremu, pri čemu on može biti i globalan, ali to se ne može odrediti iz rezultata. Razlika je također je ta da za klasične metode uvijek možemo locirati ekstrem unutar željene preciznosti. Za rezultat genetskog algoritma ne možemo sa stopostotnom vjerojatnošću reći da li je globalni ili lokalni optimum, te da li je određen sa željenom preciznošću.

```
procedura Evolucijski_program
{
     t = 0
     generiraj početnu populaciju potencijalnih rješenja P(0)
     sve dok nije zadovoljen uvjet završetka evolucijskog procesa
     {
          t = t + 1
          selektiraj P'(t) iz P(t-1)
          generiraj novu generaciju P(t) iz P'(t) koristeći genetske
             operatore
     }
}
```

Slika 1: Struktura evolucijskog programa

Prilikom inicijalizacije generira se početna populacija jedinki. Obično se početna populacija generira slučajnim odabirom rješenja iz domene. Slijedi proces koji se ponavlja sve dok ne istekne vrijeme ili je zadovoljen neki uvjet. Taj proces se sastoji od djelovanja genetskih operatora selekcije, križanja i mutacije nad populacijom jedinki. Tijekom selekcije loše jedinke odumiru, a bolje opstaju te se u slijedećem koraku, križanju, razmnožavaju. Križanjem se prenose svojstva roditelja na djecu. Mutacijom se mijenjaju svojstva jedinke slučajnom promjenom gena. Takvim postupkom se postiže iz generacije u generaciju sve veća i veća prosječna dobrota populacije.

2.2. Dobre strane genetskog algoritma

- funkcija *f* koju treba optimizirati je potpuno proizvoljna, tj. nema posebnih zahtjeva kao što su neprekinutost, derivabilnost i sl.
- primjenjiv je na veliki broj problema
- struktura algoritma nudi velike mogućnosti nadogradnje i povećanja efikasnosti algoritma jednostavnim zahvatima (puno stupnjeva slobode)
- jednostavnim ponavljanjem postupka se može povećati pouzdanost rezultata
- ako već ne nađe rješenje (globalni optimum), daje nekakvo *dobro* rješenje koje može zadovoljiti
- kao rezultat daje skup rješenja, a ne jedno rješenje
- rješava sve probleme koji se mogu predstaviti kao optimizacijski, bez obzira da li funkcija *f* koju treba optimizirati ima za argumente realne brojeve ili bitove ili znakove ili bilo koju vrstu informacije
- vrlo jednostavno je primjenjiv na višemodalnim (višedimenziskim) problemima
- jednostavnost ideje i dostupnost programske podrške

2.3. Loše strane genetskog algoritma

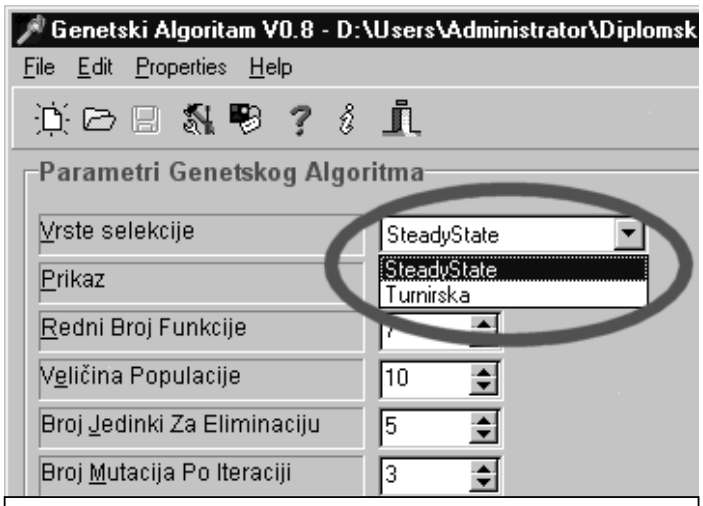
- Teško se definira dobra funkcija dobrote na temelju f
- potrebno je prilagoditi GA zadanim ograničenjima
- problem decepcijske funkcije
- često je potrebna prilagodba problema algoritmu
- teško je postaviti dobre parametre (velik utjecaj parametara na efikasnost)
- ne može se postići 100% pouzdanost rješenja
- konvergencija je znatno sporija od ostalih numeričkih metoda
- potrebno je posebno definirati genetske operatore za posebne vrste prikaza
- zbog stohastičnosti nikad ne znamo prirodu nađenog rješenja
- zbog izvođenja velikog broja računskih operacija GA je spor, traži se velika procesorska snaga

2.4. Pojmovi vezani uz genetski algoritam

2.4.1. Vrste selekcije

Postupak selekcije bi se mogao realizirati sortiranjem i odabirom VEL_POP-M najboljih jedinki (VEL_POP – veličina populacije, M – broj jedinki za eliminaciju), ali to bi dovelo do prerane konvergencije genetskog algoritma. To ima za posljedicu da se izgubi dobar genetski materijal koji mogu sadržavati loše jedinke.

Genetske algoritme dijelimo s obzirom na vrste selekcija na *generacijske* (turnirska) i *eliminacijske* (steady-state).



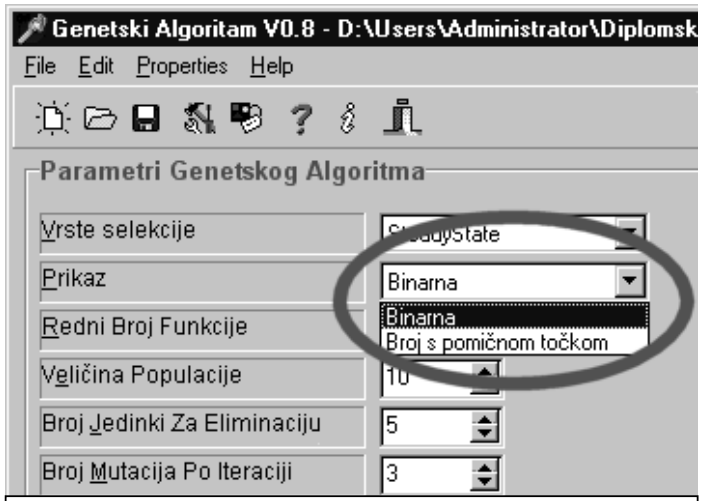
Slika 2: Vrste selekcije genetskog algoritma

Kod generacijskog (turnirska selekcija) genetskog algoritma nova populacija se bira iz stare na osnovu njene dobrote. Broj jedinki nove populacije je jednaka broju jedinki stare populacije. Djeca se biraju od roditelja proporcionalno njihovoj dobroti. Odabir se vrši tako da se VEL_POP puta biraju dvije jedinke iz stare populacije, uspoređuje ih i bolju jedinku kopira u bazen za reprodukciju nad kojim će u sljedećem koraku djelovati genetski operatori.

Eliminacijska selekcija (steady-state) ne bira *dobre* kromosome za sljedeću populaciju, nego *loše* koje treba eliminirati i reprodukcijom zamijeniti novima.

2.4.2. Prikaz kromosoma

Kromosomi se mogu prikazivati na razne načine, ali je kod računanja optimuma funkcija naj lakše kromosome prikazivati binarno ili kao broj s pomičnom točkom.



Slika 3: Prikaz kromosoma

Binarni prikaz dozvoljava elegantne i jednostavne operatore i dobro služi u teoretskoj analizi rada genetskog algoritma. No ima i neke nedostatke. Za problem sa stotinu varijabli u opsegu [-500,500], preciznost na šest decimala duljina binarno kodiranog rješenja iznosi oko 3000 bita. Osim zahtjeva za memorijom, veča duljina značajno usporava rad algoritma. Kod binarnog prikaza postoji i problem da se dvije susjedne točke mogu razlikovati u više od dva bita. Npr. Razlika između 255 i 256 nije jedan nego je to razlika između 011111111 i 100000000, čak u devet bitova koji se trebaju promijeniti križanjem. Zato se ponekad upotrebljava Grayev kod, kod kojeg je razlika između dva susjedna broja upravo jedan bit.

Kod prikaza kromosoma kao broj s pomičnom točkom taj problem ne postoji. Prednost je i da je puno brži, pogotovo kada se traži velika preciznost. Jednostavnije je i održavanje populacije jer ih svi današnji programski jezici podržavaju. No kod kromosoma koji su predstavljeni kao broj s pomičnom točkom potrebno je definirati operatore koji će njima rukovati.

3. O PROGRAMU

Program se sastoji od glavnog programa i DLL-a (*Dynamic-Link Library*). Glavni program je samo sučelje prema DLL-u u kojem se nalazi implementiran genetski algoritam.

3.1. Datoteka dinamičkog povezivanja

Genetski algoritam koji je prije bio implementiran na Unix-u sada je trebalo preraditi da radi kao DLL u Windows-ima. DLL je kao što i samo ime kaže biblioteka procedura prevedih na strojni jezik. No, da bi mogla biti dinamička , odnosno da je više programa može pozivati bez da brinu o varijablama, za svaki se program posebno u memorijskom prostoru rezervira prostor za varijable. Program koji poziva DLL nemože pristupiti i njegovim varijablama. Da bi komunikacija između DLL-a i glavnog programa bila moguća, trebalo je napisati procedure koje vračaju, odnosno postavljaju, varijable koje se nalaze u DLL-u, a koristi ih i glavni program.

```
extern "C" __declspec( dllexport )char* VratiImeDat(void)
{
      return ime_dat;
}
extern "C" __declspec( dllexport )void PostaviImeDat(char *vid)
{
      strcpy(ime_dat,vid);
}
```

Slika 4: Izgled funkcije u DLL-u

Na slici 4 vidimo kakav izgled imaju sve funkcije koje se pozivaju iz glavnog programa. Izraz **extern "C"** znači da se funkcije prevode kao C funkcije bez ukrašavanja imena funkcija koje se događa ako je funkcija kompajlirana kao C++. Dok izraz *__declspec(dllexport)* označava kompajleru da sljedeću funkciju mora kompajlirati za *eksportiranje* iz DLL-a. Dalje slijedi normalan kod funkcije. Sve funkcije koje su potrebne glavnom programu napisane su sa takvim zaglavljem.

Budući da je glavni program pisan u *Object Pascal-u*, da bi on mogao pozivati funkcije koje su u DLL-u, trebalu mu je pokazati gdje se nalaze te funkcije i s kojim parametrima ih treba pozivati.

```
procedure PostaviImeDat(vid:PChar);stdcall;external'GADll.dll'index 6;
function  VratiImeDat:PChar;stdcall;external 'GADll.dll' index 26;
```

Slika 5: Izgled zaglavlja funkcija u Pascal-u

Na slici 6 vidimo kako su napisana zaglavlja u Pascal-u za iste funkcije u DLL-u. Vidimo da Pascal ima tip **PChar** koji je identičan C-ovom **char***. Poslije definicije funkcije nalazi se **external 'GADll.dll' index 6** koji označava kompajleru da se funkcija nalazi u DLL-u po imenu GADll.dll i to je funkcija pod rednim brojem 6.Tako napisano zaglavlje se zove statičko importiranje funkcija iz DLL-a.

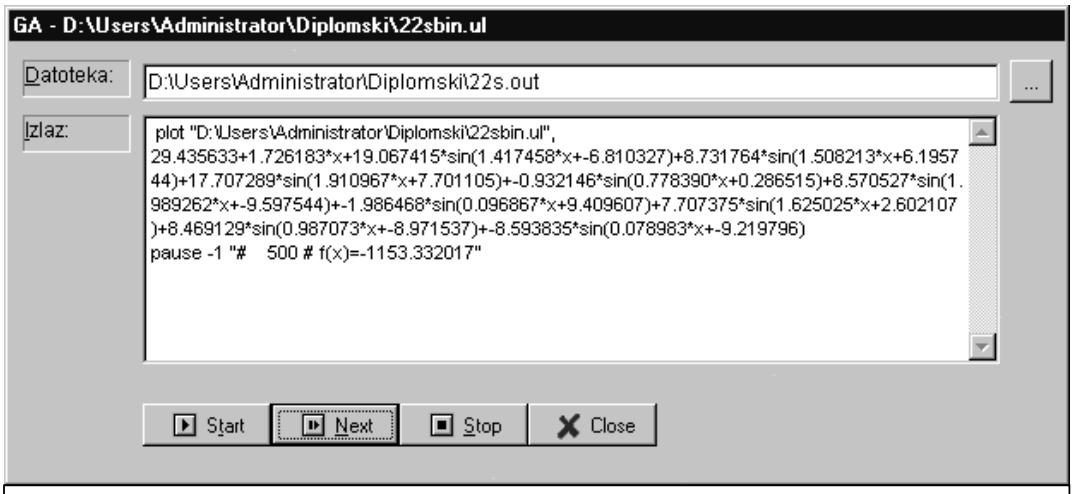
3.2. Glavni program

Glavni program je napisan u Borland Delphi©-u jer je to vizualan alat za razliku od MS Visual C++©. Programiranje u Delphi-u je brže jer postoje *wizard*-i koji ubrzavaju kreiranje glavne *form-e* i ljuske programa. Također postoje i komponente koje se jednostavno spuste na prozor. Ako želimo za neku komponentu napisati *event* (procedura koja se izvrši kada komponenta dobije određeni *message*), u *object inspector-u* izaberemo događaj i Delphi nam otvara prozor u kojem možemo editirati proceduru za koju je već napisano zaglavlje. Sve to ubrzava posao programeru i omogučuje mu da se posveti glavnom problemu.

Slika 6: Izgled glavnog programa

Glavni program sadrži standardni izbornik i toolbar sa najčešće korištenim operacijama. Parametri genetskog algoritma se unose u promijenjivim kućicama. Dok se unos granica ili točaka unosi u posebnom prozoru koji se poziva pritiskom na dugme *Unos Granica(Točaka)*. Kada smo zadovoljni parametrima snimimo ih u datoteku.

Sada možemo pozvati genetski algoritam koji se nalazi u DLL-u. Pritiskom na dugme *GA* otvara se novi prozor.



Slika 7: Izlaz genetskog algoritma

U njemu odaberemo u koju će datoteku biti izlaz genetskog algoritma i pritiskom na *Start* možemo pratiti izlaz genetskog algoritma. Svaki pritisak na dugme *Next* ispisuje trenutnu iteraciju genetskog algoritma, dok dugme *Stop* zaustavlja genetski algoritam. Na slici 7 vidimo krajnji rezultat funkcije broj 7 nakon 500 iteracija. Ako želimo izači iz tog prozora kliknemo na *Close* i ako parametri nisu bili dobri možemo ih ponovo mijenjati.

3.2.1. Funkcije

U genetskom algoritmu su implementirane ove jednostavne funkcije:

```
0 - x*sin(10*π*x)+1.0
1 - sin(10*π*x*x)/x
2 - x*x*sin(10*π*x*x)
3 - pow(x,sin(x))
4 - x*cos(tan(x))
5 - x*cos(tan(pow(x,0.5)))
6 - x*sin(30*(π/2-atan(x))/x)
7 - 0.5-(sin(x)*sin(x)-0.5)/(1+0.001*x*x)/(1+0.001*x*x)
```

Tablica 1: Implementirane funkcije

3.2.2. Implementacija poziva genetskog algoritma

Da bi glavni program mogao normalno dalje raditi, poziv početka rada genetskog algoritma stavljen je u novi *thread*. U Delphi-u postoji klasa TThread koja sakriva od programera pozive API-u. U proceduri TVrsta.Execute nalazi se poziv genetskog algoritma. Kreiranje novog *thread-a* poziva se sa :

```
with TVrsta.Create do ;
```

```
TVrsta = class(TThread)
private
protected
      procedure Execute; override;
public
      constructor Create;
end;

constructor TVrsta.Create;
begin
      FreeOnTerminate := True;
      inherited Create(False);
end;
procedure TVrsta.Execute;
begin
try
      GADllHeader.vrsta();
except
      next:=3;
end;
```

Slika 8: Poziv genetskog algoritma

4. ZAKLJUČAK

Budući da slika vrijedi kao tisuću riječi, vizualizacija samo olakšava ispitivanje i proučavanje genetskog algoritma. Puno je lakše unošenje parametara u vizualnom sučelju nego da se isti parametri moraju unosi u uređivaču teksta. Kod proučavanja genetskog algoritma ciklus promjena, startanje, proučavanje i ponovna promjena traje dugo ako se sve mora raditi ručno. Kod vizualnog sučelja sve je na dohvat miša, lako i brzo. Zbog svega toga ušteda vremena je skoro 2/3 vremena kod nevizualnog sučelja.

4.1. Mogućnosti nadogradnje

Mogućnosti nadogradnje su brojne. Budući da se sada može editirati samo jedna datoteka sa parametrima, moglo bi se ugraditi MDI (Multiple Document Interface) sučelje. U meniju bi se dodala stavka `window` sa pripadnim podmenijima za upravljanje prozorima. Može se poboljšati sučelje prema DLL-u i mogao bi se ubrzati DLL. Budući da se sada izlaz iz genetskog algoritma koristi kao ulaz u GNUPlot koji prikazuje funkcije grafički, moglo bi se dodati prikazivanje funkcija u samom programu. Rješenje postoji i potrebno ga je samo dodati. Da bi program izgledao profesionalnije može mu se dodati i sučelje prema *Registry-u* da svoje informacije zapisuje u njega. Sve što se može podešavati oko samog programa može se dodati u *Opcije programa* koje bi se također zapisivale u Registry.

Također je potrebno napisati dobar Help iz kojeg budući studenti i ostali korisnici mogu puno naučiti o genetskim algoritmima.

Budući da je sada podržano samo 7 jednostavnih funkcija i nekoliko višedimenzinalnih, može se napraviti jedno sučelje prema DLL-u s funkcijama koje bi mogao napisati bilo koji programer. Ako netko nije vičan programiranju može se dodati da se ručno unose funkcije, što je također već riješeno i potrebno je samo dodati u glavni program.

LITERATURA

[1] Golub Marin: Magistarski rad: Genetski algoritam, FER, Zagreb
[2] Jakobović Domagoj: Diplomski rad: Genetski algoritam, FER, Zagreb

SADRŽAJ