

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Genetski algoritmi

Matija Bačić

Voditelj: dr. sc. Mladen Karan

Zagreb, travanj, 2019

Sadržaj

1. Uvod.....	3
2. Prvi primjer : generiranje niza znakova	5
3. Drugi primjer : problem trgovačkog putnika.....	7
4. Zaključak	9
5. Literatura	10

1. Uvod

Veliku prepreku računarstvu oduvijek su stvarali NP problemi (problemi za koje se rješenja mogu provjeriti, ali se ne mogu izračunati u polinomnom vremenu). Takvi su problemi npr. problem raspoređivanja studenata u grupe, problem trgovačkog putnika, problem bojanja grafa i drugi. Razvojem računarstva takvi kompleksni problemi počeli su se rješavati tehnikom grube sile (eng. brute-force). Tehnika se svodi na pretraživanje cjelokupnog prostora rješenja. Međutim, ovo nije prihvatljiva metoda zbog svoje faktorijske složenosti.

Algoritme koje pronalaze dovoljno prihvatljiva rješenja, ali nam ne nude garanciju da će uspjeti pronaći optimalno rješenje te rade u niskoj složenosti nazivamo heuristikama. Jedna od takvih heuristika upravo je i genetski algoritam. On nam nudi način za riješiti probleme za koje ne postoje egzaktni algoritmi.

H. J. Holland 1975. godine stvorio je genetski algoritam. Svoju inspiraciju pronašao je u Darwinovoj teoriji evolucije koja je zasnovana na 5 temeljnih pretpostavki:

1. Populacija ima varijacije (nema identičnih jedinki)
2. Više potomaka je kreirano no što će preživjeti
3. Populacija se mijenja s vremenom
4. Neke varijacije se favoriziraju u odnosu na druge
5. Oni koji opstaju imaju favorizirane osobine

Upravo na tim principima leži genetski algoritam. On ne definira konkretno rješenje za svaki problem, već daje apstraktna pravila koja se različito implementiraju za specifične probleme.

Genetski algoritam radi nad populacijom jedinki koja se na početku generira sa što više varijacija. Također, početni skup jedinki može biti i rezultat nekog drugog algoritma ili prethodne iteracije istog algoritma. Prelazak u drugu generaciju podrazumijeva selekciju jedinki (jedinke sa boljim osobinama opstaju) koja se obavlja korištenjem funkcije dobrote. Faktor dobrote vezan je uz svaku jedinku i govori koliko je vjerojatno da će geni te jedinke biti odabrani za križanje i prelazak u iduću generaciju.

Algoritam također zahtjeva implementaciju operacije križanja jedinki, odnosno način kako se geni prenose iz generacije u generaciju. To može biti riješeno na više načina. Možemo uzeti pola gena od jednog roditelja, a drugu polovicu od drugog roditelja ili odabrati koji postotak gena uzimamo od kojeg roditelja ili uzeti svaki drugi gen od jednog roditelja, a ostale od drugog i sl.

Zadnji zahtjev algoritma jest implementacija operacije mutacije. Naime, u početnoj populaciji možda se ne nalazi dovoljan broj različitih gena da bismo križanjem došli do optimalnog rješenja. Kao rješenje za to uvodimo mutaciju, odnosno vjerojatnost da gen u DNA lancu djeteta ne bude niti od jednog roditelja, već neki slučajno odabrani iz početnog skupa gena.

Upravo su ta tri principa ono što čini genetski algoritam: križanje, mutacija i faktor dobrote. Za svaki se problem mijenja konkretan način implementacije tih pravila upravo zato jer nam se jedinke za svaki problem mogu uvelike razlikovati u strukturi DNA, skupu mogućih gena, a razlikujemo i način na koji izračunavamo dobrotu neke jedinke.

Genetski algoritam možemo opisati sljedećim pseudokodom:

```
void genetic_algorithm() {  
    create_initial_population(); //random population or input from another algorithm  
    do {  
        selection(); //by calculating fitness function for every entity  
        crossover_and_mutation(); //create next generation  
    } while(stop_condition_is_not_met);  
}
```

Postupak izgradnje genetskog algoritma može se na apstraktnoj razini opisati u nekoliko koraka :

- I. Problem postaviti kao optimizacijski problem (npr. problem generiranja određenog unaprijed poznatog niza znakova svesti na problem minimizacije broja netočnih znakova u generiranom nizu)
- II. Odrediti način prikazivanja gena
- III. Odrediti način križanja jedinki i mutacije
- IV. Odrediti način računanja faktora dobrote (implementirati izračun dobrote i način prelaska u iduću generaciju koristeći funkciju dobrote)
- V. Eksperimentalno utvrditi najpovoljnije parametre algoritma na manjem problemu (veličinu populacije, postotak mutacije i sl.)

Postupak izgradnje genetskog algoritma pokazat ćemo u nastavku na nekoliko jednostavnih primjera.

2. Prvi primjer : generiranje niza znakova

U ovom primjeru kao problem nam se zadaje generirati, pomoću genetskog algoritma, unaprijed poznat niz znakova. Ovo je trivijalan problem i služi samo upoznavanju sa postupkom kreiranja algoritma. Prvi korak je prevesti problem u problem pogodan za optimizaciju. Reći ćemo da je zadatak pronaći niz znakova jednake duljine kao početni s minimalnim brojem pogrešnih znakova na svim mjestima u nizu.

Nakon toga, biramo reprezentaciju DNA lanca i skup gena. Naš DNA lanac prikazat ćemo kao polje znakova jednake duljine kao i početno polje, dok će skup gena koje biramo za svaki DNA lanac biti svi znakovi engleske abecede (velika i mala slova) i razmak. Uočite da smo ovim ograničili generiranje nizova na one nizove koji u sebi imaju upravo samo znakove iz engleske abecede i razmak, ali to nam je dovoljno dobro za ovaj primjer. Ako našem algoritmu predamo niz znakova koji u sebi sadrži interpunkcijske znakove, algoritam neće nikada završiti jer neće uspjeti pronaći takav niz upravo zbog odabranog skupa gena.

Sljedeći korak je izgraditi algoritam za križanje jedinki i mutaciju. Za to možemo napisati sljedeći kod:

```
char[] crossOver(char[] parent1, char[] parent2, double mutationRate) {
    if(parent1.length != parent2.length) throw new DnaNotCompatibleException();
    int length = parent1.length;
    char[] child = new char[length];

    int middle = new Random(System.currentTimeMillis()).nextInt(length);

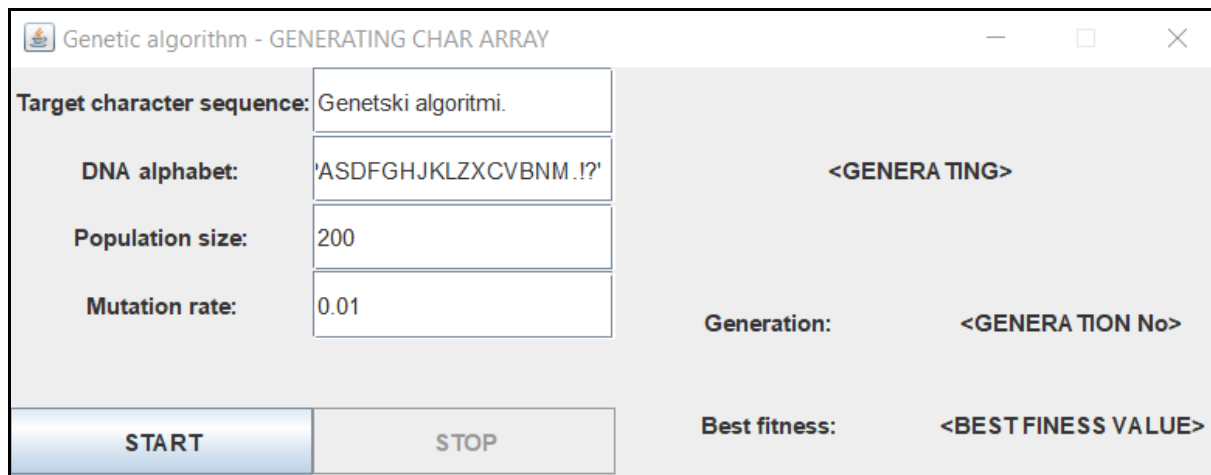
    for(int i=0; i<length; i++) {
        char next;
        if(i<middle) next = parent1[i];
        else next = parent2[i];
        child[i]=next;
    }

    for(int i=0; i<length; i++)
        if(Math.random()<mutationRate)
            child[i] = getRandomFromGeneSet(); //mutation – random gene

    return child;
}
```

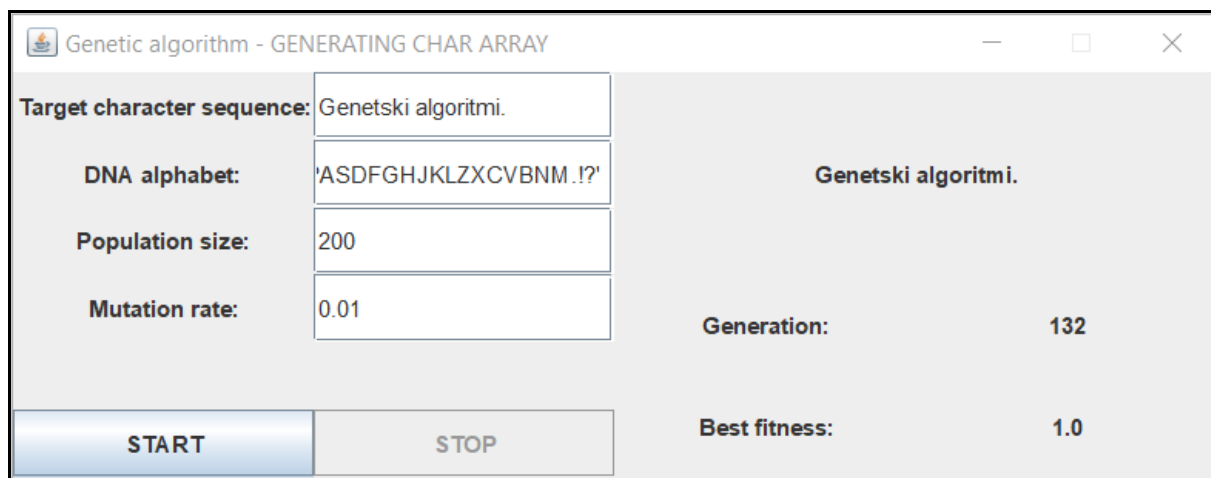
Faktor dobrote možemo računati kao broj pogođenih znakova. No, zbog načina na koji ćemo implementirati prelazak u sljedeću generaciju, računamo ga kao broj pogođenih znakova podijeljeno s brojem ukupnih znakova (odnosno to je postotak pogodaka). Takva implementacija vraća brojeve između 0 i 1 za bilo koju duljinu početnog niza (maksimalna vrijednost ne raste s duljinom niza). To će nam biti bitno kasnije. Također, funkcija dobrote koja vraća broj promašenih znakova kroz ukupan broj znakova ne bi bila dobra implementacija jer bi veću dobrotu imale jedinke koje su dalje od optimalnog rješenja. Takva implementacija kršila bi koncept dobrote.

Ostaje samo izraditi desktop aplikaciju radi lakše vizualizacije našeg algoritma. Ovaj dio prikazan je na slici 2.1.



Slika 2.1.

Aplikacija omogućava unos željenog niza znakova koji naš algoritam mora generirati, abecede gena, veličine populacije te vjerojatnosti mutacije pojedinih gena. Također, omogućuje uvid u trenutni napredak algoritma s desne strane prozora. Pokrenemo li algoritam ovako odabranim postavkama, nakon nekoliko sekundi dobit ćemo rezultat prikazan na slici 2.2.



Slika 2.2.

Primijetite da ćemo svakim izvođenjem algoritma dobiti isti rezultat, ali u različitom broju generacija. To ovisi o tome koliku sreću imamo na početku s generiranjem slučajnih jedinki te o mutaciji svake iduće jedinke.

Kako bismo olakšali izradu ostalih primjera, primijenit ćemo oblikovni obrazac strategija na naš algoritam. Tako ćemo imati kod koji je građen na načelu otvorenosti za dodavanje, zatvorenosti za izmjenu (eng. open-closed principle) te koji je zbog toga lako nadograđivati za ostale primjere bez ikakve promjene postojećeg koda.

3. Drugi primjer : problem trgovačkog putnika

Drugi primjer kojeg ćemo riješiti pomoću genetskih algoritama jest problem trgovačkog putnika, odnosno problem traženja najkraćeg Hamiltonovog ciklusa u grafu. Ovaj je problem jedan od NP-potpunih problema za koji ne postoji poznat algoritam polinomijalne složenosti. Jedini egzaktni algoritam koji ga rješava jest algoritam grube sile čija je složenost faktorijelna te je zbog toga to nepovoljno rješenje. Ovom problemu moramo pristupiti uporabom neke heurističke metode. Odabrat ćemo, naravno, genetske algoritme.

Sada kada imamo temelje algoritma napisane, moramo samo definirati razlike ovog problema u odnosu na prethodni te ih proslijediti našem kodu. Ono što naš algoritam ne zna je križati jedinke, mutirati ih, generirati slučajnu jedinku te izračunati funkciju dobrote za pojedinu jedinku. Nakon što to oblikujemo, izradit ćemo vizualizaciju te usporediti vrijeme izvođenja genetskog algoritma i algoritma grube sile za ovaj problem.

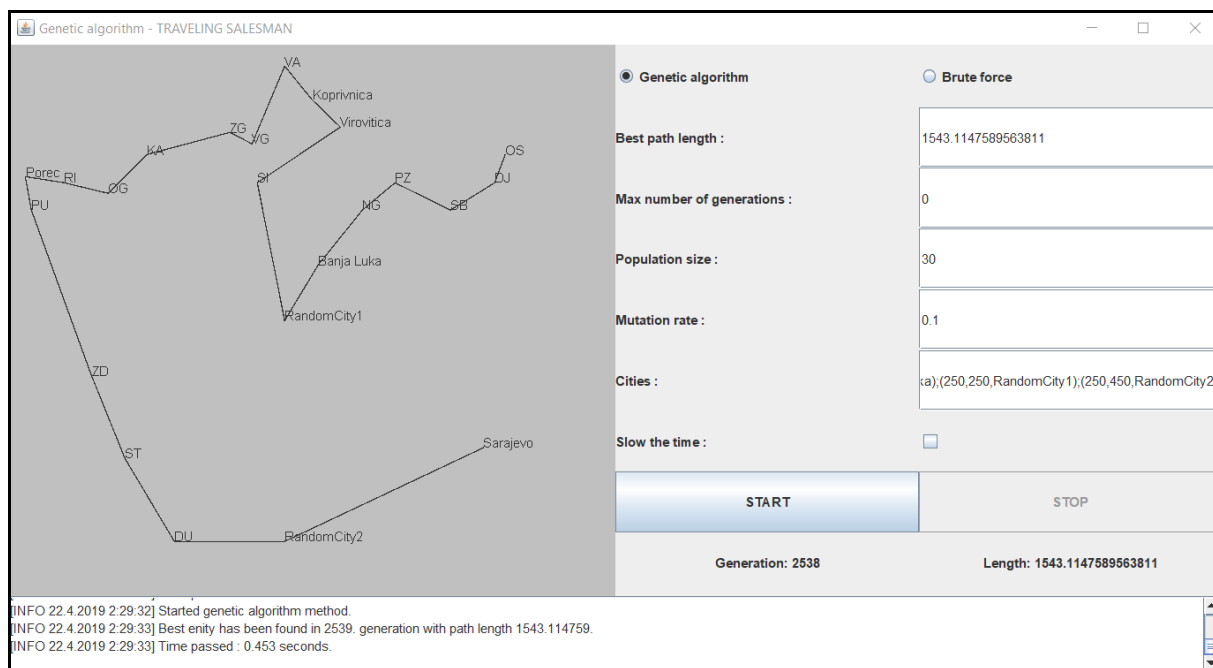
DNA lanac oblikovat ćemo kao niz gradova gdje je prvi u nizu početni grad, a gradove obilazimo redom kojim su navedeni. Lanac ima duljinu broja gradova koje je potrebno obići, a svi gradovi moraju biti zastupljeni točno jednom.

Kod oblikovanja potrebnih operacija moramo paziti na nekoliko stvari. Moramo paziti da ne generiramo niz gradova koji ima više istih gradova, odnosno da su svi gradovi zastupljeni u nizu. Također, kao rezultat operacije križanja ne smijemo dobiti jedinku koja krši navedene zahtjeve DNA lanca niti smijemo mutirati jedinku tako da neki grad u nizu zamijenimo nekim drugim iz početnog skupa gradova. Tako možemo dobiti gradove koje se ponavljaju više puta u lancu, a to ne želimo.

Ove probleme rješavamo korištenjem operacije križanja u poretku (eng. ordered crossover) i mutacije zamjenom (eng. swap mutation) umjesto operacija koje smo koristili u prvom primjeru. Operacija poredanog križanja realizira se tako da se određeni postotak gene uzima od prvog roditelja, a ostatak potrebnih gena od drugog roditelja, ali samo one gene koje već nismo dobili od prvog roditelja.

Algoritam mutacije zamjenom provodi mutaciju tako da umjesto zamjene jednog gena nekim drugim genom iz abecede, provodi zamjenu mjesta dvaju gena u lancu. Ovim smo osigurali da nećemo dobiti multipliciranje jednog gena u DNA lancu.

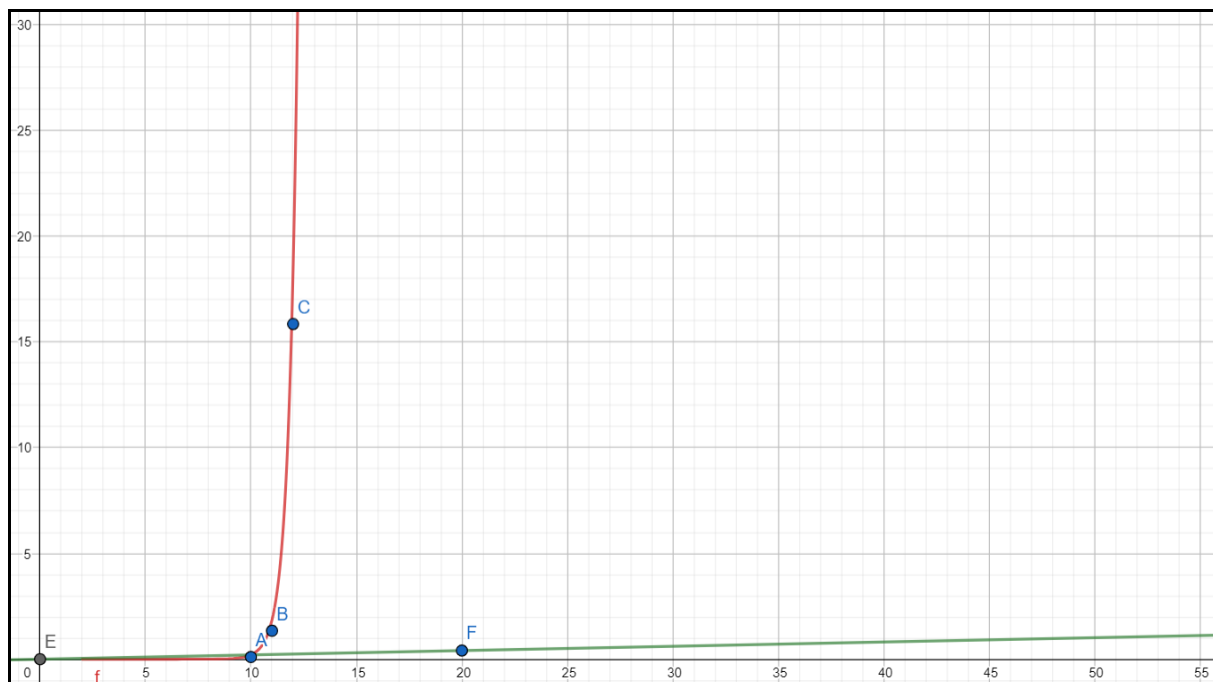
Slika 3.1. prikazuje izgled aplikacije za izvođenje ovog algoritma. Također, aplikacija nam nudi usporedbu genetskog algoritma i algoritma grube sile.



Slika 3.1.

Zbog lakšeg i točnijeg određivanja vremena izvođenja algoritma, program putem korisničkog sučelja nudi unos duljine najkraćeg puta ukoliko nam je ona poznata.

Usporedba vremena izvođenja ovih algoritama prikazana je na slici 3.2. gdje crvena krivulja opisuje vrijeme algoritma grube sile, a zelena vrijeme genetskog algoritma. Os apscisa predstavlja broj točaka u grafu, dok os ordinata bilježi prosječno vrijeme izvođenja u sekundama. Za izračun prosjeka, algoritam se izvodio 10 puta do pronalaska optimalnog rješenja.



Slika 3.1.

4. Zaključak

Kao što smo vidjeli na navedenim primjerima, genetski nam algoritam omogućuje izračun rješenja u realnom vremenu za probleme koji nemaju egzaktni algoritam ili algoritam za njih uopće ne postoji. Međutim, genetski algoritam jest heuristička metoda te se zbog toga ne bi trebao koristiti za probleme gdje postoje egzaktni algoritmi čije je vrijeme izvođenja relativno male složenosti.

Genetski je algoritam definiran na vrlo apstraktnoj razini. On je samo ideja rješenja nekog optimizacijskog problema, a na korisniku algoritma leži odgovornost implementacije pojedinih operacija na kojima se algoritam bazira.

Pokazali smo neke od mogućih implementacija operacija križanja i mutacije, no one neće odgovarati za svaki problem. Na programeru je da odabere neku od postojećih ideja ovih operacija ili razvije novu realizaciju.

Bitno je uvidjeti princip rada genetskog algoritma. U suštini, genetski algoritam nastoji što je brže moguće suziti mogući prostor rješenja da ne bismo morali pretražiti sva moguća rješenja. Iz toga se vidi da cijela umjetnost ovog algoritma leži u dobro oblikovanim operacijama križanja te dobro odabranim parametrima za izvođenje (veličini populacije, postotku mutacije). Dobro odabrani parametri mogu davati fantastične rezultate u relativno kratkom vremenu, no loše odabrani parametri mogu uvelike oštetiti izvođenje algoritma tako da algoritam „zapne“ u lokalnom optimumu jer postotak mutacije nije dovoljno velik da ga „izvuče“ iz optimuma.

5. Literatura

- [1] Marko Čupić. "Prirodom inspirirani optimizacijski algoritmi. Metaheuristike.", <http://java.zemris.fer.hr/nastava/pioa/knjiga-0.1.2013-12-30.pdf>, 2013
- [2] Marko Čupić, Bojana Dalbelo Bašić, Marin Golub. "Neizrazito, evolucijsko i neuroračunarstvo.", <http://java.zemris.fer.hr/nastava/nenr/knjiga-0.1.2013-08-12.pdf>, 2013
- [3] Mitchell Melanie. "An Introduction to Genetic Algorithms", The MIT Press, 1998
- [4] <https://towardsdatascience.com/evolution-of-a-salesman-a-complete-genetic-algorithm-tutorial-for-python-6fe5d2b3ca35>