

**Vizuelizacija ostrvskog genetskog  
algoritma**  
**Seminarski rad u okviru kursa**  
**Računarska inteligencija**  
**Matematički fakultet**

Sandra Radojević, Dara Milojković  
mi16030@alas.matf.bg.ac.rs, mi16100@alas.matf.bg.ac.rs

## **Sažetak**

Premisa iza ovog rada je vizuelni prikaz genetskog algoritma. Ideju možete da vidite ako odete na sajt – ime onog sa youtube-a -. Glavni zadatak ovog rada je da se genetski algoritam predstavi na interesantniji, grafički način. Time želimo da poboljšamo zainteresovanost naučne zajednice i mlađim generacijama da se bave genetskim algoritmima i ostalim granama veštačke inteligencije.

# 1 Uvod

Među prvim algoritmima koji simuliraju genetski sistem živih bića je genetski algoritam (GA). GA modeluje genetsku evoluciju, gde je karakteristika jedinki njen genetski zapis. Kao i u prirodi, pokretačka snaga GA je selekcija, odnosno preživljavanje najspasobnijih. Postoji i paralelni GA koji koristi ideje koje je Darwin zapisao u Teoriji evolucije.

Proučavajući raznovrsne životinjske vrste na Galapagosu, pogotovo ptice, ovaj naučnik je shvatio da na mnogim mestima, malim i izolovanim ostrvima obitavaju populacije jedinstvenih stvorenja. Uspeo je da dokaže da se ova stvorenja razlikuju od svojih rođaka na drugim mestima i otkrio je da su se zebe na Galapagosu prilagodile sredini tako što su iskoristile činjenicu da tamo živi veoma mali broj drugih vrsta ptica – zebe su evoluirale u 80 različitih tipova, od kojih se svaki adaptirao na određeni način života. Neke su čak postale i grabljičice. Konkurentno i distribuirano programiranje je omogućilo da se ova pojava u prirodi prenese i na rešavanje optimizacionih zadataka u računarstvu. Životinja koja je ovde korišćena kao primer genetskog algoritma je miš. Te miševe razlikujemo po bojama, svako ostrvo ima različitu boju i svaka jedinka ima ili tammiju ili svetliju boju u zavisnosti od svog fitnesa.

## 1.1 Paralelni genetski algoritmi

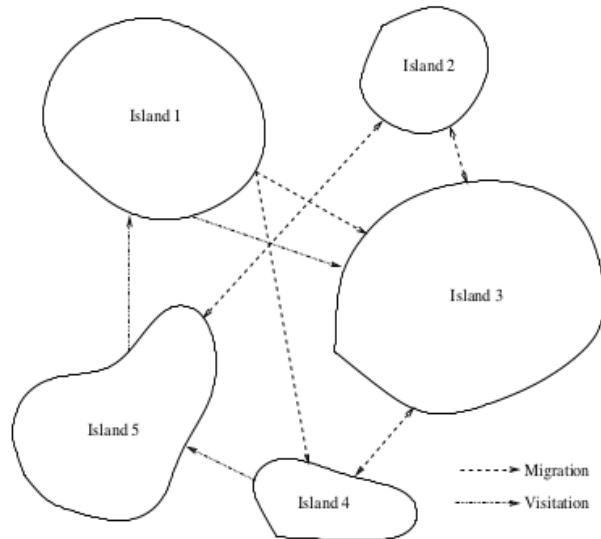
Postoje 3 glavne kategorije paralelnih GA

- Gospodar-sluga (*eng. Master-slave GAs*): karakterističan je po tome što se primenjuje nad celom populacijom, ali je računanje funkcije fitnes distribuirano, tj. nekoliko procesora računaju fitnes za jedinke.
- Fino gradirani GA (*eng. fine-grained GAs*): primenjuje se nad celom populacijom i svaka jedinka se dodeljuje nekom procesoru. Tako da na svakom procesoru postoji neki komšiluk. Operatori selekcije i reprodukcije su striktno ograniceni u tom komšiluku.
- ostrvski (*eng. island GAs*): za razliku od prethodne dve kategorije, ostrvski koristi više populacija u nadi da će raznolikost ostrva pomoći u nalaženju najoptimalnijeg rešenja. Još jedna karakteristična razlika između ostrvskog i prethodnih je da ostrvski algoritam može da se implementira i na jednoprocesorskom sistemu, ali da ima podršku za niti.

## 1.2 Ostrvski algoritam

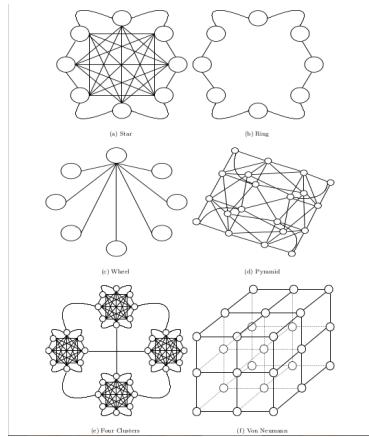
Ostrvski GA podrazumeva da na svakom ostrvu, koje se uključuje u algoritam, postoji populacija nad kojom se primenjuje GA. Samim tim, operatori selekcije i operatori reprodukcije se primenjuju na svakom ostrvu. Međutim, novina koja se uvela u ovoj vrsti algoritma je migracija. Migracija podrazumeva da će jedna (ili više) jedinki otići na drugo ostrvo i postati njegovi članovi.

Migraciju karakterišu:



Slika 1: Prikaz ostrvskog algoritma

- Topologija zajednice- Određuje putanju migracije između ostrva. Na primer topologija prstena, gde je svako ostrvo povezano s svojim najbližim susedom. U ovoj topologiji, optimalno rešenje se sporije širi i ostrva su raštrkana. Samim tim, topologija je veoma bitna za brzinu konvergencije. Međutim ako je neka gusta raspodela ostrva može doći do prevremene konvergencije.



Slika 2: Topologije zajednice

Mnoge topologije su nalik topologijama zasnovanih na optimizaciji rojevima. Najčešće se koriste:

**Zvezda** Svako ostrvo je povezano sa svim ostrvima i migracija je moguća po svim putanjama.

**Prsten** Svako ostrvo je povezano sa svojim susedom i migracija je moguća samo do njih.

**Točak** Jedno ostrvo je sa svima povezano i podseća na koloniju.

- Verovatnoća migracije- Pokreće pitanje kad treba izvršiti migraciju. Ako se pokrene prerano, onda ostrvo može da izgubi šansu da dobro konvergira.
- Mehanizam selekcije jedinke koja migrira
- Mehanizam zamene koja će jedinka biti zamenjena migrantom

Ako se desi migracija, destinacija se bira probabilistički. Omogućeno je korišćenje turnirske selekcije zasnovane na prosečnom fitnesu populacije. Pored ove selekcije, destinacija će razmatrati da li treba prihvati ovu jedinku ili ne. Ovo se takođe vrši probabilistički, npr. da li je njena vrednost fitnes funkcije bolja od prosečne vrednosti fitnes funkcije. Inicializacija populacije se vrši nasumično i to na svakom ostrvu. Nasumičnim biranjem pokrivamo veću oblast u prostoru pretrage. Operatori selekcije su isti kao i kod običnog genetskog algoritma i u zavisnosti od problema mogu proizvesti različite rezultate. Kriterijum zaustavljanja se ne razlikuje od GA:

- Dostignut određen broj generacija
- Globalni fitnes između ostrva se ne menja
- Fitnes generacije je u nekoliko iteracija bio isti

## 2 Pravljenje problema

Korišćen alat za pravljenje je Qt. Qt je višeplatformski aplikacioni interfejs koji se koristi za razvoj grafičkih korisničkih orijentisanih programa, međutim, ne mora da se koristi samo za grafičke interfejse. Koriste se standardi jezika C++, ali on podržava i više različitih jezika kao što su Python i QML. Da bismo iskoristili mogućnosti koje pruža Qt, moramo naći odgovarajući problem iz genetskog algoritma koji će biti dovoljno jednostavan za grafički prikaz i prikaz mogućnosti koje ostrvski genetski algoritam pruža. Kao problem je izabran traženje maksimuma funkcije sa 3 argumenta.

$$F(x, y, z) = |x^2 - x * y + z| \quad (1)$$

Da bismo postavile osnove grafickog dela, potrebna su nam ostrva. Izabrale smo da broj ostrva bude 3 mada moze i drugi broj. Međutim, nije preporučljivo pravljenje mnogo takvih ostrva jer je projekat grafički zahtevan. Ostrva su scene. Mi na svaku scenu namestamo poseban pogled i odgovarajuću pozadinu da bismo ih razlikovali. Klasa Ostrvo nasleduje QGraphicsScene koje je omogućio sam Qt, na toj sceni postavljamo aktere koji predstavljaju naše jedinke koje se kreću slobodno. Možemo primetiti da se scena stalno menja. To omogućuje klasa QTimer koja se aktivira posle nekog zadatog vremena. QTimer-om kontrolišemo kretanje, kao i pokretanje GA na svakom ostrvu. Možemo ovu aktivaciju da smatramo kao promena sezona u životu miševa koji se nalaze na sceni. Kada vreme dođe da se pokrene GA za miševe dolazi vreme sazrevanja i sezona parenja. Sa tehničke strane, aktiviramo nit koja vrši operacije selekcije i reprodukcije. Kada završi sa obradom nit vraća novu populaciju koju mi moramo vremenom uvodimo u staru. Time simuliramo rađanje novih i starenje starih jedinki.

Bilo je polemike da li migracija treba da se vrši u nitima ili u klasi Ostrvo. Presudila je tehnička mogućnost Qt-a. Qt daje mogućnost rada sa signalima. Signal je neko obaveštenje da se u sistemu nešto dogodilo, u konkretnom slučaju napravila se nova generacija. Na nekoliko načina smo pokušale da migranta pošaljemo drugom ostrvu ali to nije bilo moguće jer nit odgovara samo objektu koji ju je stvorio. Migracija se samim tim vrši u klasi Ostrvo.

Migracija je glavni deo ostrvskog algoritma, bilo je veoma teško izabrati dobar uslov koji će povremeno slati jedinku na drugo ostrvo. Rešenje se uzima iz prirode. Ako imamo neku koloniju jedinki u prirodi i nađe nova jedinka koja iz nekog razloga želi da im se priključi, mora da zadobije njihovo poverenje. Ako je ne prihvate ne može da uđe, biva ili ubijena ili će se vratiti svojoj prvobitnoj koloniji. Mi smo ovu odluku simulirale tako što smo našle prosečan fitnes svih na tom ostrvu i uporedile sa migrantovim. Ako je fitnes migranta veći od proseka rado će biti prihvaćen.

Međutim, imamo i sukobljavanje ostrva. Jedno ostrvo ne želi prepusti svoju jedinku nekom drugom ostrvu tako da smo uveli i probabilističku meru (ako je neki pseudo-slučajan broj veći od 0.05). Ako se oba uslova ispune, postavlja se pitanje koliko njih sme da napusti ostrvo. Mnogi algoritmi dopuštaju više prelaza, u početku smo i mi stavile i svaka generacija je imala neku migraciju. Ali smo mi stavile da samo 1 jedinka sme da napusti

ostrvo po generaciji, jer zavisi od funkcije kojoj određujemo maksimum. Što je jednostavnija funkcija to je migracija slabija.

Još jedno glavno svojstvo migracije je topologija migracije. Topologija može biti prstenasta, u obliku zvezde, u obliku točka ili neki hibrid. Topologija zavisi od broja ostrva i problema. Za kompleksne probleme potrebna je bolja povezanost ostrva dok za neke lakše kao ovaj nije toliko izražena. Topologija koja se koristi je usmeren prsten što znači da jedinka sme da ode do susednog ostrva ali ne može da se vrati ukoliko ne obide ceo krug. Potrebno je bilo odrediti koliko jedinki će imati svako ostrvo. Za ovakav lakši zadatak i grafički prikaz nije potrebna velika populacija. Između 10 i 50 jedinki. U svakoj iteraciji algoritma stara populacija se zamenjuje novom. Jedinke biramo nasumično iz nekog intervala koji se nalazi u *input.txt*. Za svaku računamo fitnes funkciju i napravljenu populaciju sortiramo. Međutim, maksimum neke funkcije najbolje računamo kada upotrebimo kao fitnes funkciju:

$$G(x, y, z) = \frac{1}{(1 + F(x, y, z))} \quad (2)$$

Jezik C++ dozvoljava sortiranje po nekoj funkciji tako da su jedinke sortirane po 2. Od broja jedinki zavisi i vreme koje zadamo QTimer-u. To se podešava ručno. Ako imamo N jedinki u populaciji i vreme t koje je potrebno da se jedan par roditelja zameni svojom decom to znaci da globalno vreme T treba da bude:  $T >= \frac{N}{2} * ts$

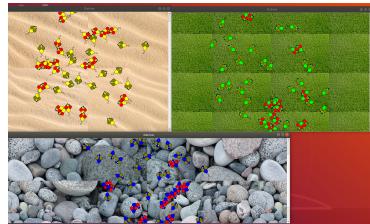
Jedinka je predstavljena u klasi Miš. Ta klasa nasleduje QGraphicsItem koja zna kako da se ponaša u odnosu na scenu. Metod koji joj omogućava haotično kretanje je advance(). Što se tiče GA strane klase Miš, ona poseduje posebnu strukturu Genotype. koja u sebi sadrži funkciju fitnesa 1 kao i:

$$[x, y, z] \quad gde \quad x \in [a_1, b_1], y \in [a_2, b_2], z \in [a_3, b_3] \quad (3)$$

Odnosno, svaka jedinka je predstavljena vektorom realnih vrednosti koji predstavlja neku tačku iz našeg prostora potencijalnih rešenja. U ovom slučaju, taj prostor je trodimenzionalan. U zavisnosti od funkcije koju maximizujemo možemo napraviti i drugačije nizove i funkcije.

Svaka jedinka živi na nekom ostrvu, i svakom ostrvu je pridružena neka boja. Naši miševi će zavisno od ostrva na kom se nalaze neke odredjene boje. Intenzitet te boje će prikazati funkcija fitnesa jedinke dok se kreće po ostrvu. Tako možemo da vizualizujemo napredovanje naše populacije kroz generacije. Što je jedinka tamnija, to ona ima bolji fitnes.

Ovakvom konfiguracijom dobijamo sledeće rezultate.



Slika 3: 3 ostrva sa svojim populacijama bez fitnesa

Na svakom ostrvu se zasebno izvršava genetski algoritam nad njegovom populacijom. Klasa koja to radi zove se Nit. Kao što i ime kaže, ovako će se postići konkurentnost izvršavanja. Nit pokreće QTimer koji je će imitirati petlju kod običnog genetskog algoritma. Samim tim, jedno ostrvo ne mora da bude u procesu GA dok neko drugo možda jeste.  
Funkcije koje su nam na raspolaganju:

- Mutacija

Cilj mutacije je da se očuva bogatstvo genetskog materijala tako što će se proizvesti blaga promena jedinke u nekom smislu. Ovako sprečavamo preterano favorizovanje najboljeg rešenja, obezbeđujemo određeni nivo slučajnosti i manja je verovatnoća da ćemo zaglaviti u nekom lokalnom optimumu. Ova funkcija, shodno tome, menja proizvoljni gen nekom nasumičnom vrednošću koja zadovoljava ograničenja. Tome odgovara zamena vrednosti jedne promenljive. Postavljeni stepen, tj. verovatnoća da će se to dogoditi je 0,05.

- Selekcija

Ovom funkcijom ćemo birati jedinke iz populacije i vršiti njihovu reprodukciju. Kada se dve jedinke izaberu, vrši se njihovo ukrštanje, a zatim i mutacija. Novodobijene jedinke se čuvaju u listi koja će zameniti trenutnu. Za izbor jedinki korišćena je turnirska selekcija. Ona bira 6 nasumičnih jedinki iz populacije i od njih vraća onu koja ima najbolju vrednost fitness funkcije. Pokušano je i ukrštanje susednih jedinki u populaciji, ali su turnirskom selekcijom dobijeni bolji rezultati.

- Ukrštanje

Ova funkcija kombinuje gene dva roditelja da bi formirala decu. Kod nas je korišćeno jednopoziciono ukrštanje, što znači da smo izabrali neki indeks nasumično, i do njega prekopirali gene jednog roditelja, a od te pozicije pa do kraja, od drugog roditelja. Jednim načinom na koji to možemo uraditi je formirano jedno dete, a drugim drugo.

- Elitizam

Imamo mali broj promenljivih, samo tri, pa će promene zasebnih promenljivih dosta uticati na fitness. Zato je bitno da zaštitimo najbolje jedinke kroz generacije. Mi štitimo 6 najboljih jedinki u svakoj iteraciji, tj obezbedjujemo da su one sigurno članovi nove generacije ovom metodom.

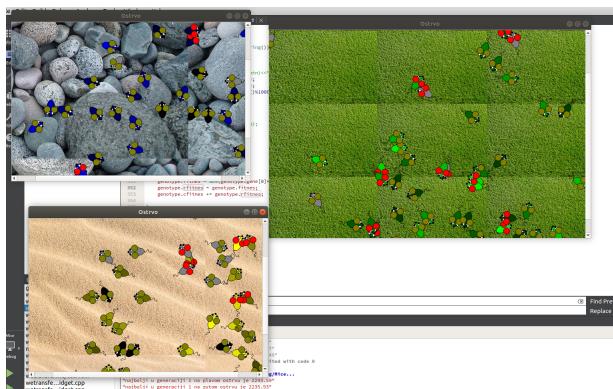
- druge pomocne funkcije

### 3 Smena generacija

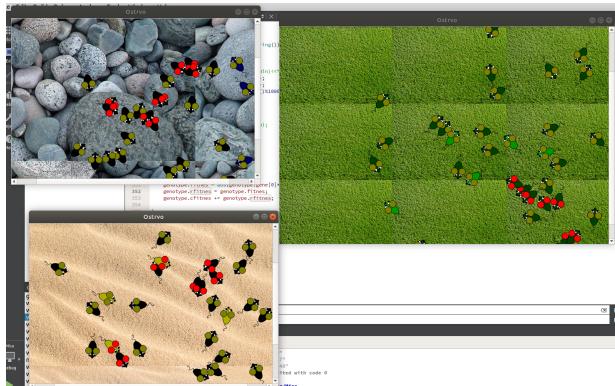
Na kraju GA imamo smenu generacija. Grafički je bilo zahtevnije ovo izvesti jer se sve dešava u sekundama (milisekundama) da neke promene ne mogu biti odmah zabeležene ljudskim okom. Zato je potreban još jedan QTimer koji će imitirati starenje i rađanje. Starenje miševa koji su ušli u proces selekcije i reprodukcije smo obojili sivom bojom i ti miševi će hteti da pobegnu od svoje dece pa se pomeraju ka gornjem levom uglu scene. To je lakše reći nego uraditi jer imamo detekciju kolizija kod miševa i svaki put kad se dva miša sudare njihove usi pocrvene i pokušavaju da se pomere jedan od drugog. Bilo da dodu do ivice ili ostanu na sceni da ih mi vidimo scena će ih obrisati jer smo uspeli sve stare jedinke da zamenimo novim. Oni koji su izabrani elitizmom će vratiti svoju staru boju.



Slika 4: 3 ostrva sa svojim populacijama pre GA



Slika 5: 3 ostrva sa svojim populacijama posle GA



Slika 6: 3 ostrva sa svojim populacijama nakon smene generacija

Kada se jedna iteracija završi globalni QTimer započinje novo pokretanje niti, tj. genetskog algoritma i tako dok ne dostignemo određen broj generacija. Kriterijum zaustavljanja je dostizanje 1000-te iteracije. Algoritam nije brz jer koristimo sinhronizaciju časovnika. Ako imamo 1000 iteracija za T vremena izraženog u milisekundama dobijamo  $1000*T$  milisekundi čekanja.

## 4 Izlaz i zaključak

GA testiramo za parametre:

$$[x, y, z] \quad \text{gde } x \in [20, 100], y \in [100, 200], z \in [10, 50] \quad (4)$$

Nakon testiranja ostrvskog GA, dobili smo sledeće rezultate za najbolje jedinke:

Iteracija	Najbolji na zelenom	Najbolji na plavom	Najbolji na zutom
1	7321.4	7188.65	8921.65
500	7321.5	7188.66	8921.66
999	7321.4	7188.67	8921.7

Nakon testiranja GA, dobili smo sledeće rezultate za najbolje jedinke:

Iteracija	Najbolji
1	7965.97
500	8859.54
999	8280.12

Nakon pokretanja brute-force algoritma za ovu funkciju, kao najbolje rešenje dobijamo:

Iteracija	Najbolji
1000	9970
1000000	9900

Dobijeni rezultati su jako slični, kod ostrvskog možemo da vidimo da su se 2 ostrva ostala ispod 8000 čak i posle 1000 iteracija, dok se Žuto ostrvo održalo na svom maksimumu. Kod običnog GA možemo da vidimo da je opala maksimalna funkcija. Za ovako laku funkciju, nije bilo potrebno paralelizovati i rezultati koji se dobijaju ovim testiranjem, manje-više se podudaraju sa običnim GA algoritmom. Grafika može da pomogne u praćenju podataka kako se koja jedinka ponaša u svakoj iteraciji. Korisno je kada želimo da vidimo kako se parametri problema menjaju. Ljudsko oko lakše i brže detektuje promenu boje nego što čovek uspe da pročita neku liniju teksta. Međutim, grafički je zahtevno ako se radi o nekim kompleksnijim algoritmima, kao što je problem trgovackog putnika za veliki broj gradova. Isto važi i za paralelizaciju, ovako mali program ne može da izvede puni potencijal mogućnosti konkurentnog programiranja. Za velike probleme je poželjno koristiti paralelizaciju, jer se smanjuje vreme čekanja na rezultate. Dodatno, ako je u pitanju ostrvski algoritam, koji omogućuje i migracije, imamo bolju komunikaciju sa rezultatima i bolju pretragu prostora.

Same migracije su ovde vršene probabilistički, i u svakoj generaciji. Međutim, moguće je postaviti intervale koji određuju koliko često će se migracije vršiti, na koliko generacija, što takođe može uticati ne samo na performanse algoritma i krajnji rezultat, već i na vreme izvršavanja.

## 4.1 Dodatak

### Literatura

- [1] Korišćena literatura sa sajta,  
<http://poincare.matf.bg.ac.rs/stefan/ri/index.html>
- [2] Korišćena literatura iz knjige Computational Intelligence, Andries P. Engelbrecht,  
<https://papers.harvie.cz/unsorted/computational-intelligence-an-introduction.pdf>
- [3] Korišćena literatura sa sajta,  
<https://github.com/matfvi/vi/tree/master/2018.2019>
- [4] Korišćena literatura sa sajta,  
<https://github.com/search?q=coarse+grained+genetic+algorithm>