Optimiranje evolucijskim algoritmima

Kuzmić, Jurica

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Jurica Kuzmić

Zagreb, 2017. godina.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentori:

Prof. dr. sc. Dragutin Lisjak, dipl. ing.

Student:

Jurica Kuzmić

Zagreb, 2017. godina.

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem se svojem mentoru na ukazanom povjerenju, savjetima i uputama.

Također se zahvaljujem svojoj obitelji na podršci tijekom dosadašnjeg akademskog obrazovanja i pisanja ovog rada.

Jurica Kuzmić

SVEUČILIŠTE U ZAGREBU

FAKULTET STROJARSTVA I BRODOGRADNJE

Središnje povjerenstvo za završne i diplomske ispite

Povjerenstvo za završne ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa:	
Ur.broj:	

ZAVRŠNI ZADATAK

Student: **JURICA KUZMIĆ** Mat. br.: 0035191236

Naslov rada na hrvatskom jeziku: **OPTIMIRANJE EVOLUCIJSKIM ALGORITMIMA**

Naslov rada na engleskom jeziku: **OPTIMIZATION WITH EVOLUTIONARY ALGORITHMS**

Opis zadatka:

Evolucijski algoritmi predstavljaju stohastički način pronalaženja optimalnog rješenja nekog problema čiji se rad temelji na prirodnoj evoluciji odnosno selekciji opisanoj u djelu "Podrijetlo vrsta" od Charles Darwin-a.

1. Opisati grupe evolucijskih algoritama.
2. Detaljno opisati način rada GA i GP algoritama.
3. Na konkretnom primjeru prikazati optimiranje GP algoritmom.
4. Zaključak.

Zadatak zadan:

30. studenog 2016.

Rok predaje rada:
1. rok: 24. veljače 2017.
2. rok (izvanredni): 28. lipnja 2017.
3. rok: 22. rujna 2017.

Predviđeni datumi obrane:
1. rok: 27.2. - 03.03. 2017.
2. rok (izvanredni): 30. 06. 2017.
3. rok: 25.9. - 29. 09. 2017.

Zadatak zadao:

Izv. prof.dr.sc. Dragutin Lisjak

v.d. predsjednika Povjerenstva:

Izv. prof. dr. sc. Branko Bauer

SADRŽAJ

POPIS SLIKA

POPIS OZNAKA

Oznaka	Jedinica	Opis
$\mathcal{G}$	-	Genotipski prostor
$\mathcal{F}$	-	Fenotipski prostor
n	-	Duljina kromosoma (potencijalnog rješenja)
p_i	-	Relativna pogodnost i-te jedinke
p_m	-	Vjerojatnost mutacije
cb	-	Cijeli broj
D	-	Ukupna dobrota populacije
$\overline{D}$	-	Prosječna dobrota populacije
p_k	-	Vjerojatnost selekcije za eliminaciju
f_p	-	Pogodnost programske strukture / kvadrirana greška

SAŽETAK

Ovaj rad bavi se pružanjem teorijskog uvoda o radu evolucijskih algoritama (EA) kao algoritama koji se mogu upotrijebiti za optimizaciju brojnih problema iz inženjerske struke. Područje istraživanja EA-ma nastaje unutar područja strojnog učenja kojem je cilj osposobljavanje računala za samostalno programiranje. Inspiraciju u svojem radu, EA-mi posuđuju od biološke evolucije prema teoriji Charlesa Darwina. Osnovnim skupinama EA-ma smatraju se: evolucijske strategije (ES), evolucijsko programiranje (EP), genetski algoritmi (GA) te genetsko programiranje (GP). Iznesena teorija o radu algoritama može se upotrijebiti za rješavanje vrlo velikog broja problema kod kojih je moguća primjena nekih od spomenutih algoritama, a upotreba navedene teorije demonstrirana je na primjerima rada GA i GP algoritma koji su dani na završetku ovoga rada.

Ključne riječi: evolucijski algoritmi, evolucijske strategije, evolucijsko programiranje, genetski algoritmi, genetsko programiranje, metode prikaza rješenja, metode selekcije, varijacijski operatori, regresijska analiza genetskim programiranjem (GP).

SUMMARY

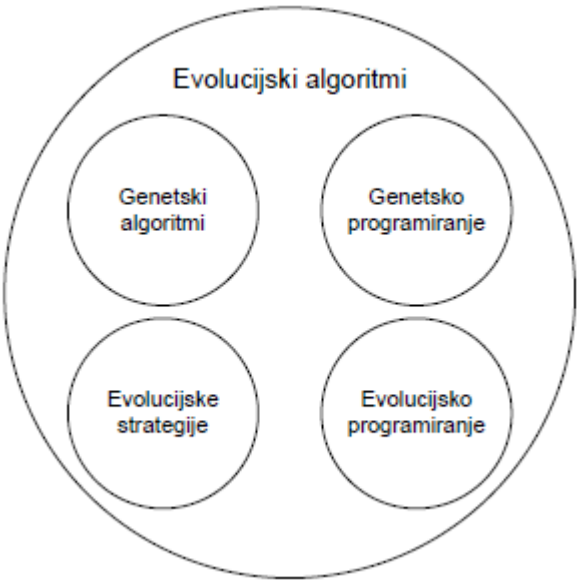
This thesis provides theoretical introduction of the way in which evolutionary algorithms (EAs) work and can be used regarding optimization of wide number of problems that engineers face. EA field of study has formed as part of machine learning, whose main goal is creating computers capable of self-programming. EAs borrow inspiration from biological evolution, which is a theory brought forward by Charles Darwin. Main members of EAs are: evolutionary strategies (ES), evolutionary programming (EP), genetic algorithms (GA) and genetic programming (GP). Theory about the way EAs work that is presented here, can be applied to very large number of problems and is also used on examples of GA and GP runs, given at the end of this thesis.

Key words: evolutionary algorithms (EAs), evolutionary strategies (ES), evolutionary programming (EP), genetic algorithms (GA), genetic programming (GP), solution representation, selection methods, variation operator, regression analysis using genetic programming (GP).

1. UVOD

Ideja o stvaranju računalnih programa koji evoluiraju postoji od vremena samog začetka računalne tehnologije. Još davne 1950-te godine, brojni računalni znanstvenici nastojali su računalima dati sposobnost samostalnog učenja. U tom je pogledu predvodnik bio Arthur Lee Samuel, koji se smatra prvom osobom koja je izvela operaciju u kojoj je računalo samostalno učilo. Pojam kojim je Samuel definirao čitavo spomenuto područje glasi machine learning ili na hrvatskom, strojno učenje. Inicijalni cilj tog područja bio je osposobljavanje računala za samostalno programiranje. To se pokazalo kao previše ambiciozna i u to doba nedostižna težnja, stoga su se istraživači s vremenom okrenuli skromnijim ciljevima. Kao posljedica tog zaokreta nastala je nova definicija strojnog učenja kao područja koje podrazumijeva istraživanje i razvoj računalnih algoritama koji se automatski unaprjeđuju temeljem stečenog iskustva [1]. Kroz ljudsku povijest, ljudi su oduvijek nastojali iskustva stečena promatranjem prirode oko sebe primijeniti u njihovim praktičnim ili istraživačkim radovima. Tako smo oponašanjem ptica i riba dobili avione i podmornice, proučavanjem načina komunikacije šišmiša i dupina osmišljeni su radari i sonari, itd.. Imajući na umu navedena ali i brojna druga uspješna rješenja nastala po principu bionike, tj. principu primjene bioloških ili prirodnih načela u istraživanju i konstruiranju ljudskih sustava, računalni znanstvenici inspiraciju za prilagodljive algoritme također su tražili upravo u prirodi [2]. Tako su proučavajući model procesa biološke evolucije, uočili mogućnost primjene pojednostavljenog Darwinovog evolucijskog modela kako bi stvorili adaptivne algoritame kojima je cilj rješavanje raznih optimizacijskih problema. Upravo takvi algoritmi koji su nastali u sklopu istraživanja unutar područja strojnog učenja, danas su poznati pod nazivom evolucijski algoritmi [1, 3]. Biološka evolucija sastoji se od nekoliko jednostavnih mehanizama: razvoj vrsta odvija se pomoću križanja i rekombinacije, nakon kojih slijedi selekcija koja omogućuje preživljavanje i reprodukciju onim jedinkama koje su bolje prilagođene okolini u kojoj se nalaze. Gledano iz inženjerske perspektive, evolucija ustvari predstavlja metodu pretraživanja vrlo velikog prostora mogućih rješenja. Sva ta moguća rješenja predstavljaju jedinke (organizme) koje se ocjenjuju prema nekim promjenjivim kriterijima uspješnosti nakon čega u konačnici preživljavaju samo one najpogodnije. Kod evolucijskih algoritama koriste se upravo ti spomenuti osnovni mehanizmi pri generiranju novih mogućih rješenja, no oni su naravno nešto pojednostavljeni. Pod klasične evolucijske algoritme, danas ubrajamo: genetske algoritme (GA), evolucijske strategije (ES), evolucijsko programiranje (EP) te genetsko programiranje (GP). Sa stajališta primjene u tehničkoj struci,

ovi algoritmi pokazali su se vrlo uspješnim u rješavanju raznih teških optimizacijskih problema kao i ostvarenju strojnog učenja, zbog čega su posljednjih godina postali vrlo zanimljiv predmet istraživanja a time i predmet ovoga rada [3]. U prvom djelu rada biti će dane teoretske osnovne zajedničke svim evolucijskim algoritmima. U drugom dijelu pokušat ću obraditi svaki od algoritama zasebno, s naglaskom na genetske algoritme (GA) i genetsko programiranje (GP).



Slika 1. Osnovne skupine evolucijskih algoritama

2. EVOLUCIJSKI ALGORITMI

2.1. Paralela između teorije evolucije i evolucijskih algoritama

Osobom koja je formulirala osnove moderne teorije evolucije smatra se engleski prirodoslovac Charles Darwin. On se tijekom svojih putovanja bavio proučavanjem raznih životinjskih vrsta iz svih krajeva svijeta te je svoje spoznaje 1859. godine iznio u knjizi *O podrijetlu vrsta*. U toj knjizi iznosi ideju o nastanku i evoluciji vrsta uz pomoć prirodne selekcije. Darwin je do takve ideje došao primijetivši da za sve populacije živih bića vrijede dva fenomena koja ih povezuju . Logički bi se moglo zaključiti da bi broj jedinki neke populacije trebao eksponencijalno rasti zbog jednostavne činjenice da živa bića obično stvaraju više potomka, međutim podaci prikupljeni sa terena tijekom istraživanja pokazali su da to nije tako. Praćenjem veličine populacija raznih životinjskih vrsta, Darwin je uočio prvi od spomenutih fenomena, a to je da ta brojka u stvari teži nekom konstantnom iznosu. Osim toga, druga zajednička pojava koju primjećuje, a koja nastaje iz generacije u generaciju, jest pojavljivanje različitosti svojstava (fenotipa) među jedinkama unutar istih vrsta. Upravo te različitosti upućuju ga na zaključak da živa bića razvijaju neka nova svojstva te da ona s dobrim osobinama kao što su otpornost na neke bolesti, imaju veću vjerojatnost da opstanu u borbi za ograničene resurse i prenesu svoja pozitivna obilježja na svoje potomke, za razliku od onih koji takve osobine ne razviju. U konačnici zaključuje da je posrijedi regulacija populacija živih bića putem prirodne selekcije, te da samo oni koji se uspiju najbolje prilagoditi svojoj također promjenjivoj okolini, preživljavaju. Iz ovakve formulacije teorije o prirodnoj evoluciji, mogu se primijetiti četiri osnovna preduvjeta potrebna da bi do evolucije prirodnim odabirom uopće došlo [1]:

1. Mora postojati mogućnost reprodukcije jedinki u populaciji
2. Mora doći do varijacije koja utječe na vjerojatnost preživljavanja jedinki
3. Mora postojati mogućnost nasljeđivanja
4. Na raspolaganju moraju biti ograničeni resursi, što će izazvati natjecanje za opstanak

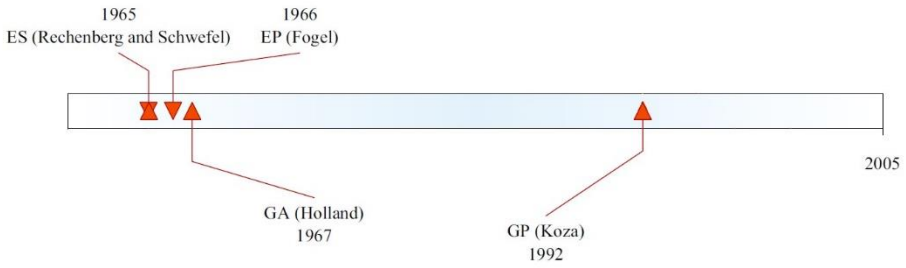
Navedena četiri faktora rezultiraju prirodnom selekcijom koja mijenja (evoluira) karakteristike početne populacije u nekom vremenskom periodu. Upravo na tim istim principima počiva ideja o evolucijskim algoritmima zbog čega su oni i dobili svoje ime. Evolucijski algoritmi funkcioniraju na način da definiraju cilj u obliku nekog kvalitativnog kriterija koji se zatim koristi pri evaluaciji potencijalnih rješenja kroz više koraka njihove rafinacije. Ukoliko je kvalitativni kriterij ispravno definiran, te se upotrijebi dovoljan broj koraka rafinacije

(iteracija), ti algoritmi izbacit će optimalno ili približno optimalno rješenje postavljenog problema. Postupak izbora najboljih pojedinaca za razmnožavanje (potencijalnih rješenja) kod ovih algoritama zove se selekcija, dok se kvalitativan kriterij temeljem kojeg se ona vrši definira funkcijom pogodnosti. Pored toga, kako bi se osiguralo da nova potencijalna rješenja u nekom koraku rafinacije, ne postanu identične kopije svojih roditelja, mora postojati i neka tehnika reprodukcije sa mehanizmom varijacije. Ukoliko tog mehanizma ne bi bilo, rad ovog algoritma ne bi imao smisla baš kao što bi to i u prirodi značilo da je evolucija živih bića nemoguća. Varijacija se stoga, kao i u prirodi, ovdje postiže tzv. varijacijskim operatorima, a to su: mutacija, koja se u kontekstu evolucijskih algoritama zove inovacijski operator te izmjenom genetičkog materijala među jedinkama (križanje i rekombinacija) koja se naziva konzervacijski operator. Različite tehnike unutar područja evolucijskih algoritama koriste različite varijacijske operatore. Tako postoje tehnike koje većinom rade s inovacijskim operatorima, dok neke većinom rade s konzervacijskim [1]. Detaljnije o nastanku evolucijskih algoritama, kao i načinu njihova rada biti će rečeno u narednim poglavljima.

2.2. Razvoj evolucijskih algoritama

Začetnikom evolucijskih algoritama smatra se Richard Friedberg koji je 1958. godine izveo eksperiment čiji je cilj bio izrada algoritma koji bi stvarao niz programa u asemblerskom jeziku za rad na računalu sposobnom prepoznati tek jedan bit informacija. Asemblerski jezik predstavlja najniži oblik programskog jezika koji sadrži upute potrebne za osnovno funkcioniranje nekog programibilnog uređaja te je svojstven samo uređaju za koji se piše. Strukture programa koje su nastajale, bile su sposobne obavljati operacije zbrajanja dva bita. Iako je bio ograničen skromnom računalnom moći svoga doba, njegov algoritam funkcionirao je na sličan način kao što evolucijski algoritmi funkcioniraju danas. Započeo je nasumičnim stvaranjem jedne ili više različitih programskih struktura. Nakon toga, upotrijebljen je varijacijski operator, u ovom slučaju bila je to mutacija, tj. nasumična promjena u jednom bitu programske strukture. Za evaluaciju dobivenih programskih struktura, odnosno provođenje selekcije, koristio je binarnu pogodnost, prema kojoj je dobiveni program ili pogodan ili nije. Takav pristup predstavljao je ozbiljan nedostatak zbog toga što takva povratna informacija nije mogla biti upotrijebljena kao vodilja za idući postupak traženja pogodne programske strukture. Iako je za današnje poimanje rezultat njegova rada skroman, u to doba takav način razmišljanja bio je vrlo napredan i zbog toga se smatra pretečom modernih evolucijskih algoritama čime Friedberg postaje začetnikom te grane unutar područja strojnog učenja [1]. U narednim

godinama uslijedio je razvoj danas najpoznatijih evolucijskih algoritama. Najprije su nastale evolucijske strategije koje je 1965. godine prvi predložio Schwefel u svojem diplomskom radu, da bi ih kasnije Rechenberg u sklopu svoje doktorske disertacije razradio. Zatim je uslijedio razvoj evolucijskog programiranja koje razvija Fogel 1966., nakon čega slijede genetski algoritmi 1967. godine i njihov začetnih Holland te na kraju genetsko programiranje 1992. godine za čiju pojavu je najzaslužniji Koza [2]. Vremenska crta koja prati razvoj spomenutih algoritama dana je na slici.



Slika 2. Vremenska crta nastanka osnovnih pripadnika EA-ma [2]

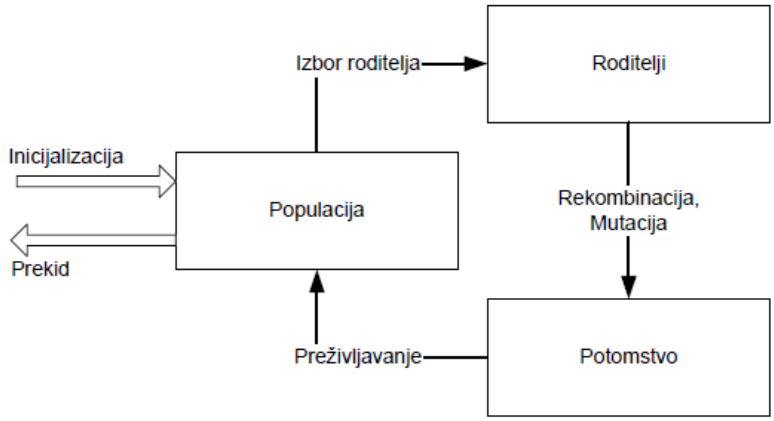
2.3. Princip rada jednostavnog evolucijskog algoritma

Kao što je već rečeno, evolucijski algoritmi inspirirani su biološkom evolucijom te služe kako bi se izvršili neki određeni zadaci učenja ili optimiziranja. Različite vrste ovih algoritama razvijene kroz povijest imaju nekoliko osnovnih poveznica, a to su [1]:

- populacija rješenja
- selekcija
- inovacijski operatori
- konzervacijski operatori
- kvalitativne razlike

Kod jednostavnog evolucijskog algoritma, u prvom koraku polazi se od generiranja početne populacije rješenja. U sljedećem koraku, ocjenjuje se pogodnost svakog člana generirane populacije. Ta pogodnost f_i pojedinca x_i ocjenjuje se funkcijom pogodnosti Φ: $f_i = \Phi(x_i)$. Pojedinci koji rezultiraju većim iznosom funkcije pogodnosti smatraju se boljim od onih manjeg iznosa. Nekih općenitih smjernica za oblikovanje funkcije pogodnosti nema, već se ona prilagođava problemu koji se nastoji riješiti pomoću tzv. metode pokušaja i pogreške. Ovdje leži temeljna razlika između prirodne evolucije i evolucije računalnim algoritmom, jer prirodna evolucija uvijek ima jasno određenu funkciju pogodnosti pomoću koje one manje prilagođene

jedinke eliminira, za razliku od umjetne kod koje je vrlo važno tu funkciju definirati ispravno zbog čega je ovdje posrijedi spomenuta metoda pokušaja i pogreške. Upravo je to jedan od glavnih razloga zbog kojih se kaže da evolucijski algoritmi rade prema pojednostavljenom principu prirodne evolucije. Nakon što je procjena pogodnosti provedena, stvara se nova generacija populacije rješenja koristeći mehanizam selekcije uz dodatak varijacijskih operatora. Selekcijski mehanizam pomoću funkcije pogodnosti osigurava da pojedinci koji imaju veću vrijednost pogodnosti dobiju veću šansu za preživljavanje i prijenos svojeg genetskog materijala sljedećoj generaciji putem mutacija, križanja i rekombinacije od ostalih [3]. Takav postupak ponavlja se sve dok se ne postigne neki uvjet prekida. Na slici je prikazan dijagram toka jednostavnog evolucijskog algoritma, a ispod se nalazi pseudokod koji zajedno ilustriraju upravo opisani proces rada.



Slika 3. Dijagram toka jednostavnog EA-a [3]

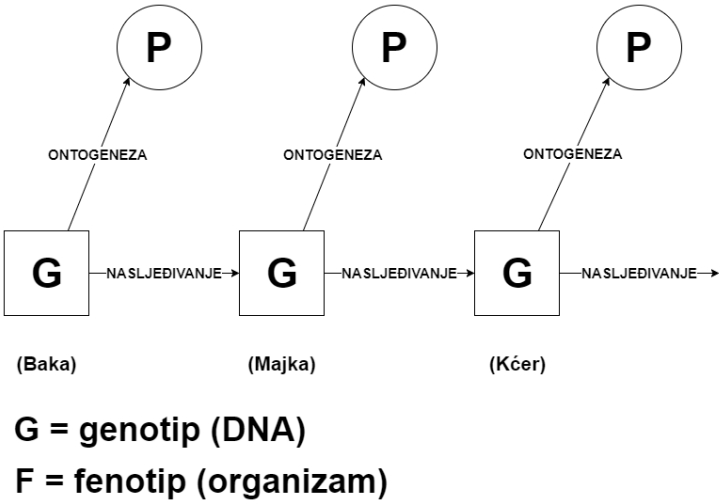
```
POČETAK
        INICIJALIZIRAJ populaciju slučajnih kandidata;
        OCIJENI svakog kandidata
        PONAVLJAJ DO (UVJET PREKIDA ZADOVOLJEN)
                1 IZABERI roditelje;
                2 REKOMBINIRAJ parove roditelja;
                3 MUTIRAJ rezultirajuće potomke;
                4 OCIJENI nove kandidate;
                5 IZABERI jedinke za novu generaciju;
        prekid
KRAJ
```

Slika 4. Pseudokod jednostavnog EA-a [3]

2.4. Prikaz populacije rješenja

Kod evolucijskih algoritama rješenja su prikazana u obliku koji je u prirodi ekvivalentan genotipu tj. skupu kromosoma neke populacije. Kromosomi u biologiji predstavljaju lančaste tvorevine koje se nalaze u jezgri svake stanice biljnog ili životinjskog podrijetla te posjeduju

sve informacije o fenotipu promatrane jedinke. Skup informacija koji opisuje jedno svojstvo zapisan je u dio kromosoma koji se zove gen. Kromosomi dolaze u parovima, jedan kromosom dolazi od oca, dok drugi dolazi od majke. Iz toga se može zaključiti da za svako svojstvo koje neka jedinka posjeduje postoje dva gena tj. dvije informacije. Naziv za gen u takvom paru gena gdje jedan i drugi nose informaciju za jedno svojstvo naziva se alel. U genetskom paru odnos gena može biti ravnopravan ili neravnopravan što određuje koje će svojstvo kod djece prevladati tj. u slučaju ravnopravnosti kojom će konačnom kombinacijom svojstava rezultirati. Sva svojstva neke jedinke nisu zapisana u samo jednom paru kromosoma već njih može biti više, pa tako npr. kod čovjeka postoje 23 para kromosoma. Na molekularnoj razini, gen predstavlja slijed određenog broja nukleotidnih parova uzduž molekule deoksiribonukleinske kiseline (DNK). Molekula DNK ima oblik dviju spirala koje su izgrađene od fosforne kiseline i šećera, dok su mostovi koji povezuju te spirale građeni od dušičnih baza. Dvije povezane dušične baze čine bazni, tj. nukleotidni par. Upravo te dušične baze predstavljaju najmanju jedinicu informacije u živom organizmu, a najbliži ekvivalent tome kod računala predstavlja jedan bit (0 ili 1). Zbog toga je jedan on načina prikaza jedinki kod evolucijskih algoritama upravo pomoću binarnih vrijednosti. Sveukupnost gena nekog organizma koji se prenosi s roditelja (pola s očeve strane, pola s majčine) na djecu zove se genom ili genotip neke jedinke [4]. Genotip predstavlja osnovni mehanizam za ostvarivanje različitosti unutar populacije zbog toga što se genetske promjene uzrokovane mutacijom i rekombinacijom prenose upravo preko genoma, dok fenotip za razliku od njega predstavlja sveukupnost vidljivih svojstava organizma koja će odrediti njegovu sposobnost da preživi dovoljno dugo kako bi se mogao razmnožavati. Osim navedenih pojmova, korisno bi bilo uvesti još jedan pojam, a to je ontogenza koja definira odnos između fenotipa i genotipa. Ona kaže da promjene u molekuli DNK nekog organizma mogu promijeniti taj organizam međutim, obratno u pravilu ne vrijedi. Slikoviti primjer toga u prirodi je otac koji tijekom svojeg života stekne veliku mišićnu masu radeći teške fizičke poslove i njegov sin koji to svojstvo naravno ne nasljeđuje već ga i on mora steći. Stoga se može zaključiti da se svi biološki mehanizmi nasljeđivanja u prirodi poput: kopiranja roditeljske DNK i prosljeđivanja djeci, mutacije te rekombinacije roditeljske DNK, odvijaju na razini genotipa ($\boldsymbol{\mathcal{G}}$), dok prirodna selekcija djeluje samo na razini fenotipa ($\boldsymbol{\mathcal{F}}$) zbog toga što je ontogeneza u pravilu jednosmjerna ulica [1]. Takav model nasljeđivanja prikazan je na slici.



Slika 5. Princip ontogeneze [1]

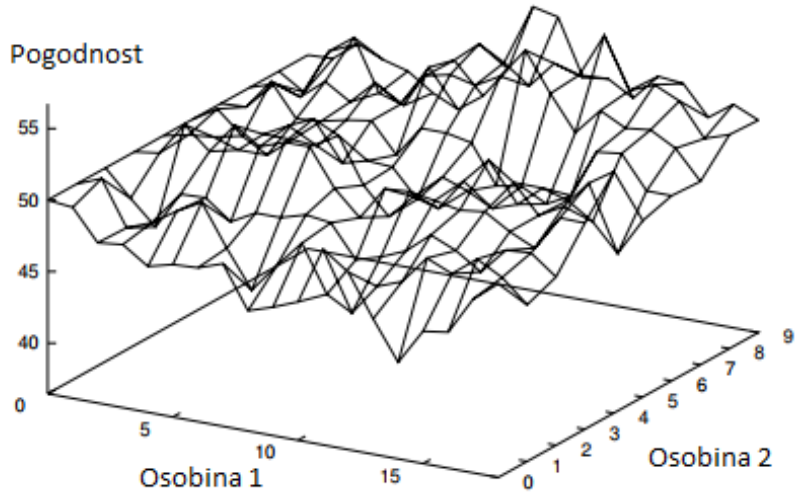
Očigledno je da su ova dva koncepta važna u prirodnoj evoluciji jednako kao i u umjetnoj kojom se bave evolucijski algoritmi. Kod algoritama fenotip i genotip predstavljaju dva zasebna prostora s podacima za koje je potrebno odrediti metodu preslikavanja tih podataka: $\mathcal{F} \rightarrow \mathcal{G}$ i $\mathcal{G} \rightarrow \mathcal{F}$. Preslikavanje podataka ustvari predstavlja kodiranje u slučaju $\mathcal{F} \rightarrow \mathcal{G}$ i dekodiranje u slučaju $\mathcal{G} \rightarrow \mathcal{F}$. Vrednovanje pojedinca odvija se u prostoru $\mathcal{F}$, a evolucijski operatori izvode se u $\mathcal{G}$ prostoru [3]. Traži se dakle način na koji će članovi populacije i njihov fenotip, na čijoj se razini odvija selekcija, biti prikazani u algoritmu čiji varijacijski operatori rade s genotipom. Jedan član populacije u algoritmu najčešće je prikazan kao niz znakova, polja ili stabla te se kao i u prirodi naziva kromosom ili ponekad gen. Najčešći oblici u kojima se podaci o populaciji rješenja tj. genotipu jedinki prikazuju za svaku skupinu evolucijskih algoritama su sljedeći [5]:

- Genetski algoritmi: binarni nizovi
- Evolucijske strategije: vektori s realnim vrijednostima
- Evolucijsko programiranje: stroj s konačnim brojem stanja
- Genetsko programiranje: LISP stabla

Osim već spomenutog binarnog načina prikazivanja, postoji još mnogo drugih. Neki osnovni načini prikaza podataka prema kojima se oni kodiraju su sljedeći [5]:

- binarni način prikaza
- cjelobrojni prikaz
- prikaz pomoću realnih brojeva (prikaz pomoću plutajuće točke)
- permutacijski prikaz
- prikaz u obliku stabla

Kod izbora načina prikazivanja, potrebno je odabrati onaj koji je najprikladniji problemu koji se rješava. Ovdje je važno naglasiti da nabrojeni oblici i načini prikaza nisu jedini, već samo najpoznatiji jer oblika prikaza kao i različitih tipova problema postoji vrlo mnogo. Svaki prikazani kromosom koji se dekodira tj. rješenje, predstavlja jednu točku u hiperprostoru ($n+1$ dimenzionalnom prostoru) fenotipa, omeđenom sa n parametara tj. osobina [3]. Najbolju ilustraciju navedenog daje iduća slika na kojoj je prikazana adaptivna površina u trodimenzionalnom Kartezijevom koordinatnom sustavu koja povezuje sve točke rješenja nekog problema sa dva parametra. Na osima se nalaze spomenuta dva parametra koji obilježavaju neka vidljiva svojstva, a visina površine u svakoj točki govori o pogodnosti odgovarajuće kombinacije svojstava tj. rješenja. Najoptimalnija rješenja nalaze se na najvišim brjegovima [6].

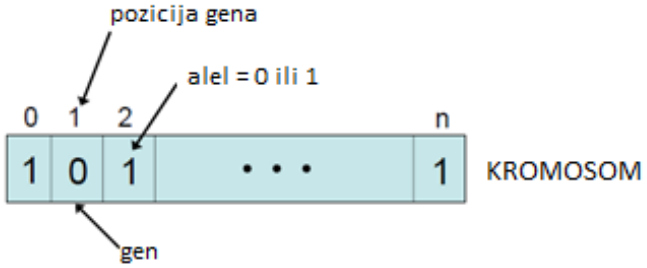


Slika 6. Ilustracija adaptivne površine [6]

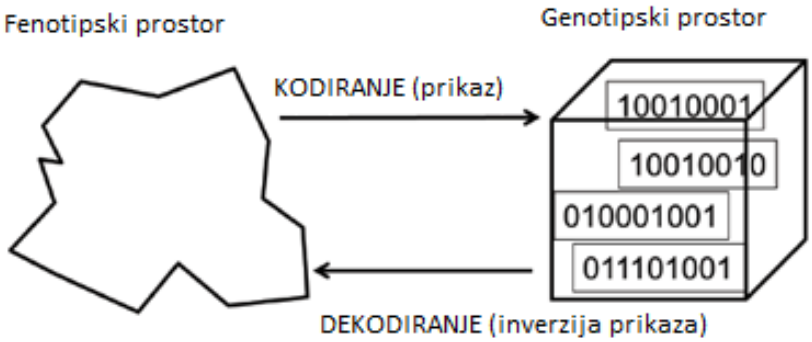
Ideja iza metode kojom će se provesti postupak samog kodiranja podataka koji se prikazuju usporedno definira i metodu dekodiranja te način prikaza rješenja. Pronalazak shema za kodiranje koje dobro reflektiraju promatrani problem predstavlja izazov o kojem ovisi ispravnost i efikasnost funkcioniranja evolucijskih algoritama [1]. U nastavku će kod svake metode prikaza biti dan i jedna primjer kodiranja genotipskog prostora. Važno je naglasiti da genotipski i fenotipski prostor ne moraju kod nekih problema imati jednaki prikaz, već se u tim slučajevima vrši dekodiranje kod kojeg se npr. binarni prikaz dekodira u prikaz s realnim ili cjelobrojnim vrijednostima.

2.4.1. Binarni način prikaza

Binarni način prikazivanja predstavlja jednu od najranijih metoda korištenih u tu svrhu. Obično se upotrebljava u prikazu rješenja kao binarnog niza brojeva kojeg nazivamo kromosom. Na idućoj slici prikazan je primjer izgleda takvog jednog kromosoma. Svako polje u nizu predstavlja jedan gen. Gledano s lijeva na desno, redni broj nekog polja predstavlja poziciju promatranog gena dok svaka binarna vrijednost (0 ili 1) unutar tih polja predstavlja jedan alel [5]. Prema tome, kromosom sadrži niz gena, a populacija tih kromosoma čini genotip odnosno prostor $\boldsymbol{\mathcal{G}}$. Na temelju takvog genotipa moguće je izvršiti dekodiranje kako bi se dobio fenotip koji predstavlja neka dobivena vidljiva svojstva tj. prostor $\boldsymbol{\mathcal{F}}$. Duljina n kromosoma ovisi o broju parametara potrebnih za rješavanje postavljenog problema [3].



Slika 7. Izgled kromosoma [5]



Slika 8. Prikaz preslikavanja F → G i G→F [5]

Ovakav prikaz koristi se za probleme čiji je fenotipski prostor sastavljen od tzv. Boolean varijabla odlučivanja (da ili ne) zbog toga što ih jedinice i nule u genotipskom prostoru mogu jednostavno reprezentirati. Ilustrativni primjer kodiranja genotipskog prostora daje tzv. problem naprtnjače. Zamislimo neku praznu naprtnjaču koja se želi napuniti, a za koju nam je poznata vrijednost njezina kapaciteta koji reprezentira njezinu veličinu i nosivosti. Pritom

nam je za svaki od n predmeta također poznata vrijednost kojom on utječe na kapacitet naprtnjače [7]. Recimo da se ovdje radi o $n = 8$ sljedećih predmeta: kapi, rukavicama, užadi, plameniku, boci vode, kutiji s hranom, željeznoj šipci te koloturi za užad . Potrebno je maksimizirati broj stvari u toj naprtnjači a da se vrijednost njezina kapaciteta ne prekorači. Ukoliko se koristi binarno kodiranje, činjenicu o tome koja od stvari se nalazi u naprtnjači kod optimalnog rješenja reprezentiraju nule i jedinice kao na slici.

Kapa	1
Rukavice	1
Uže	1
Plamenik	0
Boca vode	1
Kutija s hranom	1
Željezna šipka	0
Kolotura za užad	0

Slika 9. Kodiranje sadržaja naprtnjače binarnim prikazom

Kromosom sa slike predstavlja rješenje u kojem je točno prikazano koje predmete je potrebno staviti u naprtnjaču da bi ona bila optimalno napunjena, međutim nije nam poznato koju količinu pojedinih predmeta smijemo staviti.

2.4.2. Cjelobrojni prikaz

Prikaz pomoću cijelih brojeva smatra se boljom metodom od binarnog prikaza jer omogućuje da se kod problema s isključivo cjelobrojnim parametrima oni izravno kodiraju [5]. To znači da vrijednosti koje se dobiju u jednom kromosomu tj. rješenju u prostoru genotipa, predstavljaju izravnu reprezentaciju odgovarajućeg svojstva u prostoru fenotipa. Na primjeru problema naprtnjače iz binarnog prikaza moguće je objasniti i ovaj prikaz i kodiranje. Svaki predmet koji nam je na raspolaganju za stavljanje u naprtnjaču mogli bismo kodirati brojevima od 0 do 7 kao na slici.

Kapa	0
Rukavice	1
Uže	2
Plamenik	3
Boca vode	4
Kutija s hranom	5
Željezna šipka	6
Kolotura za užad	7

Slika 10. Kodiranje sadržaja naprtnjače cjelobrojnim prikazom

Nakon provedbe nekog od pogodnih evolucijskih algoritama koji će u obzir uzeti kapacitet naprtnjače i težinu utjecaja kojeg svaki od tih predmeta na taj kapacitet imaju, može se dobiti kromosom koji predstavlja rješenje kao na sljedećoj slici.

1	2	1	3	3	5	6	8

Slika 11. Moguće rješenje problema naprtnjače iskazano cjelobrojnim vrijednostima

2.4.3. Prikaz pomoću realnih brojeva

Kod ovog prikaza numeričke vrijednosti parametra kodiraju se kao realni brojevi a kromosomi tj. rješenja predstavljena su vektorom $x = \langle x_1, \dots, x_n \rangle$, gdje n predstavlja duljinu kromosoma odnosno broj parametra svojstvenih problemu koji se rješava [5]. Izgled kromosoma s realnim brojevima dan je na sljedećoj slici. Ovdje je također moguće primijeniti izravno kodiranje gdje su točke iz genotipskog prostora izravno povezane s odgovarajućim točkama fenotipskog prostora ovisno o problemu.

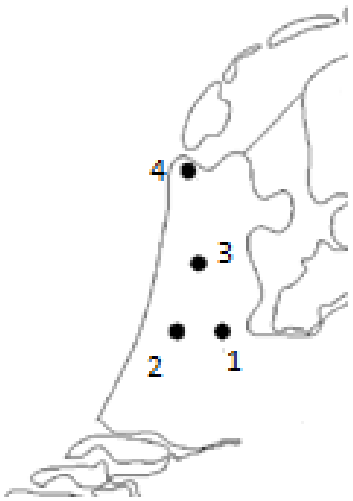
0,234	0,176	0,222	0,454

Slika 12. Kromosom s realnim brojevima

$$x = \langle 0.234, 0.176, 0.222, 0.454 \rangle$$

2.4.4. Permutacijski prikaz

Ova metoda upotrebljava se za prikaz rješenja problema čije numeričke vrijednosti parametra označavaju objekt raspoređen unutar nekog skupa različitih objekata. Problemi kod kojih se koristi podrazumijevaju pronalazak optimalnog niza tih objekata prilikom čega se vrijednost koja označava svaki pojedini objekt smije upotrijebiti samo jednom. Jedan kromosom u ovom slučaju predstavlja niz brojeva poredanih u nekom poretku koji reprezentira jedno rješenje. Primjer takvih slučajeva su: problem planiranja proizvodnje (redoslijed obrade na alatnim strojevima) te problem putujućeg trgovca (problem pronalaska najkraće rute koja obuhvaća sve lokacije uz vraćanje u polazišnu točku) koji je ilustriran na slici. Gradovi su označeni nekim brojevima od 1 do n a jedna moguća ruta koja obuhvaća sve gradove i vraća se na početak predstavlja jednu permutaciju. Ukoliko bi se u danom primjeru problem ograničio samo na prva 4 grada, vektori $[1,2,3,4]$ i $[3,4,2,1]$ bile bi valjane dvije permutacije, a mogući broj permutacija bio bi jednak $n!$ odnosno u ovom slučaju $4! = 24$.



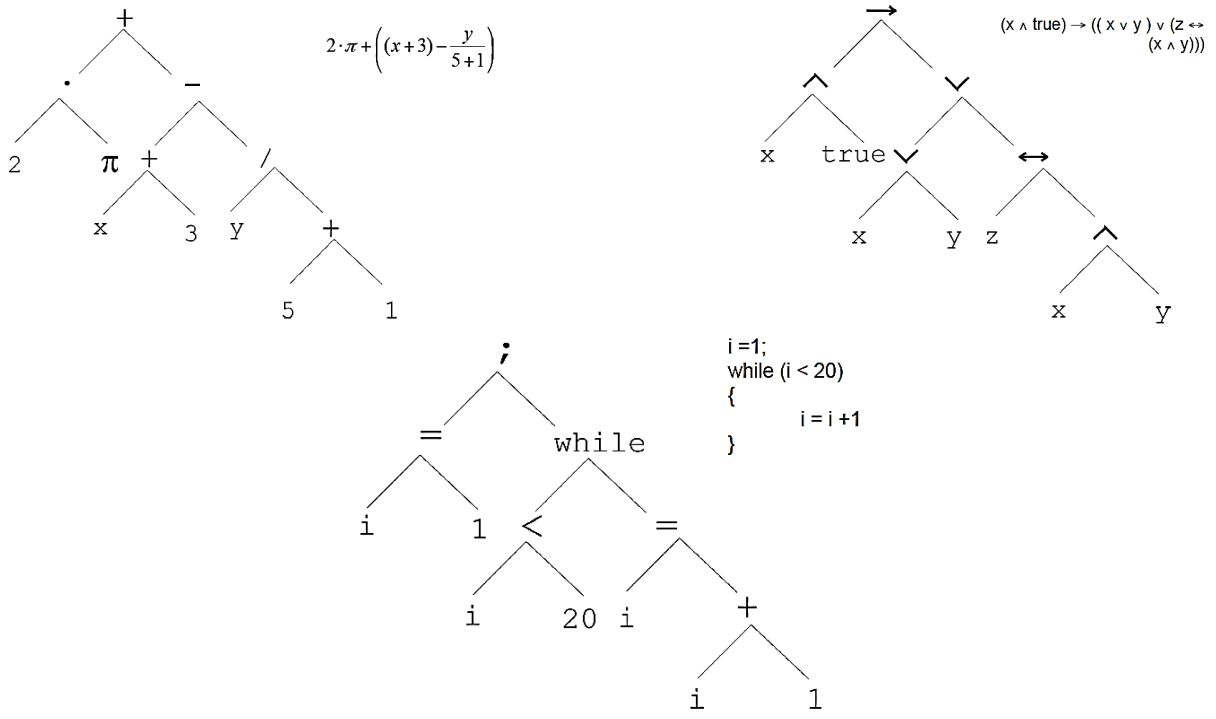
Slika 13. Raspored gradova za koje se traži redoslijed putovanja

Kromosom A	1	2	3	4
Kromosom B	3	4	2	1

Slika 14. Dva kromosoma koja predstavljaju dvije valjane permutacije

2.4.5. Prikaz u obliku stabla

Ovaj način prikaza koristi stabla kako bi prikazao rješenja tj. kromosome koji za razliku od ostalih metoda prikaza nisu linearne strukture fiksne veličine, već nelinearne strukture koje variraju dubinom i širinom. Primjeri prikaza aritmetičke i logičke formule kao i jednostavnog računalnog programa ilustriran je na slici.



Slika 15. Prikaz aritmetičke (lijevo), logičke (desno) formule te jednostavnog računalnog programa (dolje) u prikazu stablom [5]

2.5. Inicijalizacija algoritma

Inicijalizacija je ime nadjenuto postupku odabira početne populacije rješenja. Ona podrazumijeva odabir pojedinca te populacije kao i definiranje njezine veličine. Taj postupak obično se izvodi stohastički, međutim, ukoliko je unaprijed poznato neko ekspertno znanje o domeni koja se pretražuje, onda se to znanje može upotrijebiti. Kod većine postupaka inicijalizacije, koriste se četiri osnovne strategije [8]:

1) Nasumično generiranje
    a. pseudo-nasumično – generiranje nezavisnih vrijednosti koje će tvoriti ravnomjerno (uniformno) raspoređenu inicijalnu populaciju

b. kvazi-nasumično – generiranje nezavisnih ali namjerno raspršenih vrijednosti inicijalne populacije

2) Sekvencijalna diverzifikacija

- uniformno uzorkovanje prostora odlučivanja (prostora rješenja)
- rješenja se generiraju u sekvencama radi optimiziranja diverzificiranosti

3) Paralelna diverzifikacija

- rješenja se generiraju na paralelan nezavisan način

4) Heurističko generiranje

- rješenja se generiraju pomoću heuristike, primjer toga je korištenje nasumičnog pohlepnog algoritma

Razlog zbog kojeg se pretežito koriste ovakve stohastičke strategije kod evolucijskih algoritama leži u tome što se ti algoritmi kod teorijskih razmatranja testiraju kako bi se njihova efikasnost i točnost rada mogla međusobno uspoređivati pa bi uvođenje iskustvenog znanja u bilo kojem dijelu pretrage prostora rješenja zapravo bilo varanje [3]. Uz takvu tvrdnju, važno je imati na umu da ukoliko se evolucijski algoritmi koriste u realnim situacijama tj. problemima, onda se upotreba bilo kakvog ekspertnog znanja o tom problemu preporuča, međutim, s obzirom da ugrađivanje tog znanja zahtijeva ulaganje određenog vremena, kod vrlo složenih problema ponekad ta primjena nema smisla.

2.6. Postupak selekcije roditelja

Nakon što je temeljem neke od strategija inicijalizacije stvorena početna populacija, potrebno je u prvom kao i u svakom idućem koraku iz trenutne populacije odabrati jedinke koje će biti roditelji u procesu reprodukcije tj. izvršiti njihovu selekciju. Važno je naglasiti da je selekcija kod evolucijskih algoritama, za razliku od inicijalizacije najčešće deterministička metoda bazirana na vjerojatnosti. Vjerojatnost odabira pojedine jedinke ovisi o njezinoj pogodnosti koja se ocjenjuje pomoću funkcije pogodnost. Za rješenja koja rezultiraju većim iznosom vjerojatnije je da će postati roditelji, međutim čak i oni najgori pojedinci imaju određenu vjerojatnost za to. Uobičajene strategije selekcije su [8]:

- Kotač za rulet
- Stohastičko univerzalno uzorkovanje
- Turnirska selekcija

Spomenute strategije koriste se za selekciju roditelja međutim, također ih se može upotrijebiti i kod selekcije za preživljavanje [5]. U ovom radu naknadno će biti obrađene (poglavlje 2.8) strategije selekcije koje se najčešće koriste kod preživljavanja a koje su kao takve navedene u literaturi te u ovom trenutnom poglavlju nisu spomenute. Ovisno o potrebi, i jedne i druge metode mogu se upotrijebiti i za selekciju roditelja i za selekciju kod preživljavanja.

2.6.1. Selekcija metodom kotača za rulet

Ovo je najčešće korištena strategija koja se zasniva na izračunu relativne pogodnosti p_i za svaku od jedinki u promatranoj populaciji prema sljedećoj formuli:

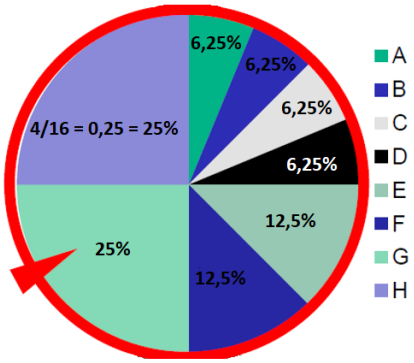
$$p_i = \frac{f_i}{\sum_{i=1}^{N} f_i}, \; f_i \geq 0 \qquad \textbf{(1)}$$

f_i – vrijednost pogodnosti i-te jedinke dobivena iz funkcije pogodnosti

N – veličina promatrane populacije

Za svaku jedinku i njezinu pripadajuću relativnu vjerojatnost p_i formira se jedan sektor unutar kružnice sa pripadajućim središnjim kutom koji iznosi $2\pi p_i$. Kada se kružnica na taj način popuni, dobije se nekoliko sektora koji reprezentiraju sve jedinke promatrane populacije. Selekcija se vrši na način da se oko te kružnice zarotira neka zamišljena točka (ili strelica) koja se nasumično zaustavlja na nekom sektoru čime tog pojedinca bira za roditelja, odnosno vrši selekciju. Upravo je zbog takvog postupka izbora roditelja koji podsjeća na vrtnju kotača za rulet ova metoda i dobila svoj naziv. Iz navedenog je jasno da oni pojedinci koji imaju veću relativnu pogodnost, ujedno imaju i veću vjerojatnost da budu odabrani. Ovaj postupak izbora ponavlja se sve dok se ne dobije željena veličina populacije roditelja. Prilikom toga neke jedinke biti će možda odabrane i po nekoliko puta, dok neke možda uopće neće biti uzete u obzir [2]. Primjer vršenja selekcije roditelja metodom kotača za rulet dan je u nastavku.

jednika	prikladnost
A	1
B	1
C	1
D	1
E	2
F	2
G	4
H	4



Slika 16. Primjer selekcije metodom kotača za rulet

Nedostatak ove metode leži u tome da neka rješenja koja u ranoj fazi evolucijskog postupka postižu visoke vrijednosti dobrote prevladaju, odnosno budu ponavljano izabrana u sljedeće populacije čime se smanjuje opseg pretraživanja a time algoritam ograničava na podprostor koji ne mora biti blizu globalno optimalnog rješenja [3].

2.6.2. Selekcija stohastičko univerzalnim uzorkovanjem

Kod ove metode također se koristi prikaz sektorima unutar kružnice temeljem izračuna relativne pogodnosti p_i, no za razliku od metode kotača za rulet, ovdje se u jednom okretaju odjednom bira više roditelja. Ukoliko nije definirano drugačije, odjednom se bira najviše onoliko roditelja koliko je promatrana populacija velika. Kutovi između svih zamišljenih točaka (ili strelica) koje se rotiraju oko kružnice jednaki su i iznose: $\frac{2\pi}{m}$, gdje m predstavlja broj jedinki koje se odjednom biraju. Za svaku se jedinku kod ove metode izbora, očekuje da bude odabrana n puta što je definirano izrazom [2]:

$$n_i = p_i \cdot m = p_i = \frac{f_i}{\sum_{i=1}^{N} f_i} \cdot m \qquad \textbf{(2)}$$

p_i – relativna pogodnost i-te jedinke

m – količina jedinki koje se biraju odjednom

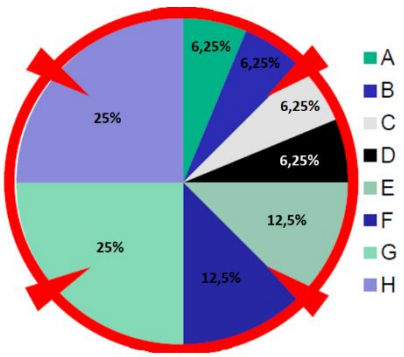
f_i – vrijednost pogodnosti i-te jedinke dobivena iz funkcije pogodnosti

N – veličina promatrane populacije

Za razliku od metode kotača za rulet, ovdje se istovremenim izborom više jedinki izbjegava pristranost zbog čega je ovo generalno gledano bolja metoda.

Primjer vršenja selekcije roditelja metodom stohastičko univerzalnog uzorkovanja dan je u nastavku. U danom primjeru odjednom se bira $m = 4$ roditelja iz populacije od 8 jedinki.

jedinka	prikladnost
A	1
B	1
C	1
D	1
E	2
F	2
G	4
H	4



Slika 17. Primjer selekcije stohastičko univerzalnim uzorkovanjem [8]

2.6.3. Turnirska selekcija

Za provođenje turnirske selekcije, potrebno je nasumično odabrati k jedinki iz promatrane populacije te usporediti vrijednosti njihove pogodnosti. Odabrane jedinke se na taj način nadmeću kao u turniru, od kud dolazi naziv ove metode. Najbolja jedinka s najvećim iznosom pogodnosti pobjeđuje i bira se za roditelja. Taj se postupak ponavlja sve dok se ne postigne željeni broj roditelja. Broj jedinki k definira veličinu turnira te predstavlja visinu selekcijskog pritiska na odabrane jedinke. Najčešće upotrebljavana turnirska selekcija je tzv. binarna kod koje je veličina $k = 2$ [2].

Primjer turnirske selekcije kod koje se biraju $k = 3$ jedinke, dan je u nastavku.

Slika 18. Primjer turnirske selekcije [8]

2.7. Reprodukcija jedinki upotrebom varijacijskih operatora – dobivanje potomka

Za definiranu populaciju roditelja sada je potrebno odrediti operatore varijacije koji će osigurati stvaranje novih varijanti rješenja odnosno djece u svakoj iteraciji. Prethodno su bile spomenute dvije vrste ovakvih operatora, a to su: inovacijski i konzervacijski operatori.

2.7.1. Inovacijski operator - mutacija

Inovacijski operatori osiguravaju sagledavanje novih aspekata problema na način da isprobavaju nove vrijednosti parametara. Kod evolucijskih algoritama, inovacijskim operatorom smatra se mutacija koja je definirana pomoću tri pripadajuća parametra:

- jačinom djelovanja
- širinom kojom istovremeno djeluje na različite komponente unutar jednog rješenja
- frekvencijom djelovanja unutar cjelokupnog algoritma

Mutacija obično podrazumijeva male promjene u odabranim jedinkama koje u genotip uvode nova svojstva koja nisu postojala kod roditelja. Snažan operator mutacije stvorio bi jedan novi

nasumičan parametar na danoj postojećoj lokaciju unutar rješenja. Ukoliko bi se primijenio na sve lokacije unutar rješenja, nastalo bi rješenje koje je u potpunosti različito od svojeg prethodnika, a kada bi takvom operatoru dodijelili maksimalnu frekvenciju, došlo bi do brisanja svih generiranih informacija o populacijama u dotadašnjem procesu rada evolucijskog algoritma [1]. Vjerojatnost mutacije svakog pojedinog gena unutar pojedinog rješenja označava se s p_m te je uobičajeno da je ona toliko mala da u prosjeku dozvoljava tek mutaciju jednog gena po kromosomu. Ovaj operator najčešće se koristi kod genetičkih algoritama s binarnim prikazom, a stvari koje treba imati na umu ukoliko se odlučuje o njegovoj upotrebi kod nekog problema su sljedeće [8]:

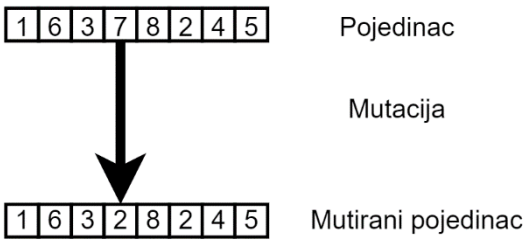
- treba osigurati da je svako rješenje u prostoru pretraživanja moguće dosegnuti
- da su rješenja koja će biti generirana valjana (ponekad nije moguće dobiti valjanja rješenja kod vrlo ograničenih problema)
- promjene koje radi trebale bi biti male uz mogućnost njihove kontrole – mala promjena u genotipu trebala bi povlačiti malenu promjenu u fenotipu (jaka lokalnost)

Izgled djelovanja operatora mutacije kod različitih prikaza dan je u nastavku:

a) Za binarni prikaz

Slika 19. Mutacija kod binarnog prikaza [3]

b) Za cjelobrojni prikaz



Slika 20. Mutacija kod cjelobrojnog prikaza [3]

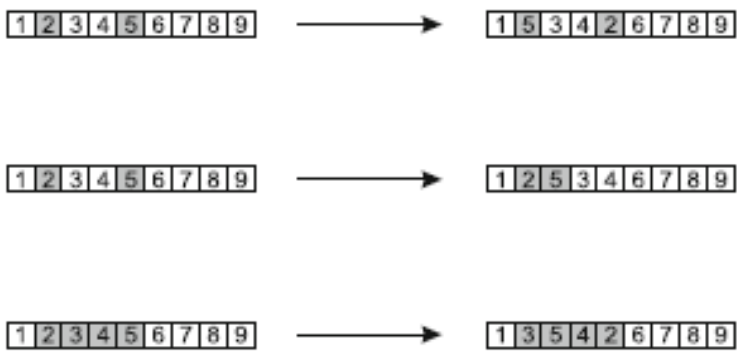
c) Za prikaz s realnim vrijednostima

Nasumično se mijenja vrijednost (alel) gena u kromosomu unutar njegove domene koja je ograničena s donjom L_i i gornjom U_i granicom [3*].

$$\langle x_1, \dots, x_n \rangle \rightarrow \langle x_1', \dots, x_2' \rangle \ \ gdje\ je\ \ x_i, x_1' \in [L_i, U_i]$$

d) Za prikaz s permutacijama

Vrši se zamjena neka dva parametara u nizu, izvlačenje nekog parametra iz niza i ubacivanje na neko drugo mjesto uz pomak ili nasumično miješanje nekog skupa parametara u nizu [3, 8].



Slika 21. Mutacija kod prikaza s permutacijama. Zamjena neka dva parametra (gore), Izvlačenje parametara (sredina) te nasumično miješanje skupa parametara (dolje) [3]

2.7.2. Konzervacijski operator – rekombinacija (križanje)

Ovaj operator upotrebljava se kako bi se utvrdilo i sačuvalo dosad naučeno kroz postupak rada evolucijskog algoritma. Glavni alat kojim se to postiže je rekombinacija dvaju ili više rješenja (kromosoma). Kombiniranjem korisnih dijelova informacija različitih jedinki u pravilu rezultira boljim ukupnim rješenjem, stoga se ta metoda koristi kako bi se ubrzala pretraga za globalno optimalnim rješenjem [1]. Poput mutacije, ovdje se radi o stohastičkoj metodi kod koje se izbor dijelova informacije koji će se miješati biraju nasumično. Primjena operatora rekombinacije (konzervacijskog operatora) varira od algoritma do algoritma. Kod genetskog programiranja (GP), najčešće se koristi ovaj operator ali u nekim slučajevima moguće je koristiti i mutaciju. Što se tiče genetskih algoritama (GA), kod njih se kao varijacijski operatori isključivo koriste upravo konzervacijski, dok se kod evolucijskog programiranja (EP) nikada ne koriste. Rekombinacijski operatori koji koriste više od dva roditelja matematički su mogući za implementaciju u radu evolucijskog algoritma, međutim u prirodi takvo što nije moguće, pa unatoč dokazanim određenim prednostima korištenja takve multi-rekombinacije, nisu često

upotrebljavani [3]. Zbog toga ću se u ovom radu koncentrirati isključivo na rekombinaciju između dva roditelja. Osnovni operatori rekombinacije (križanja) te njihovo djelovanje kod različitih vrsta prikaza rješenja dano je u nastavku [8]:

a) Za binarni i cjelobrojni prikaz

➔ križanje u jednoj točki

- lokacija križanja između dva gena u kromosomu bira se nasumično dijeljenjem svakog roditelja u dva segmenta međusobno jednake duljine, nakon čega slijedi unakrsna izmjena tih segmenata
- u ovom slučaju oba roditelja imaju jednak doprinos za oba potomka

Slika 22. Primjer križanja u jednoj točki za binarni prikaz [8]

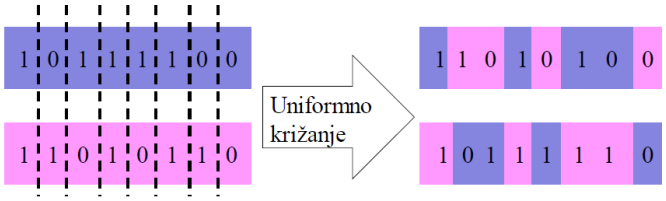
➔ križanje u N točaka

- ukoliko se križanje vrši u $N = 2$ točaka, vrši se nasumično dijeljenje dva kromosoma roditelja između dva gena tako da se formira $N + 1$ segment čije su duljine kod oba roditelja jednake
- nakon podjele vrši se unakrsna razmjena segmenata roditelja temeljem slučajnog odabira segmenta koji ulazi u križanje ili temeljem neke druge unaprijed određene metode odabira

Slika 23. Primjer križanja u dvije točke za binarni prikaz [8]

➔ uniformno križanje

- točke križanja se nalaze između svaka dva gena u kromosomu oba roditelja te se križanje vrši nasumičnim odabirom nekog gena iz kromosoma nekog od roditelja koji se mijenjaju



Slika 24. Primjer uniformnog križanja za binarni prikaz[8]

b) Za prikaze s realnim vrijednostima

➔ srednje-centrično

- križanje kod kojega su potomci blizu središnje vrijednosti roditelja
- npr. aritmetičko križanje – geni potomka se nalaze u blizini prosječne vrijednosti gena oba roditelja

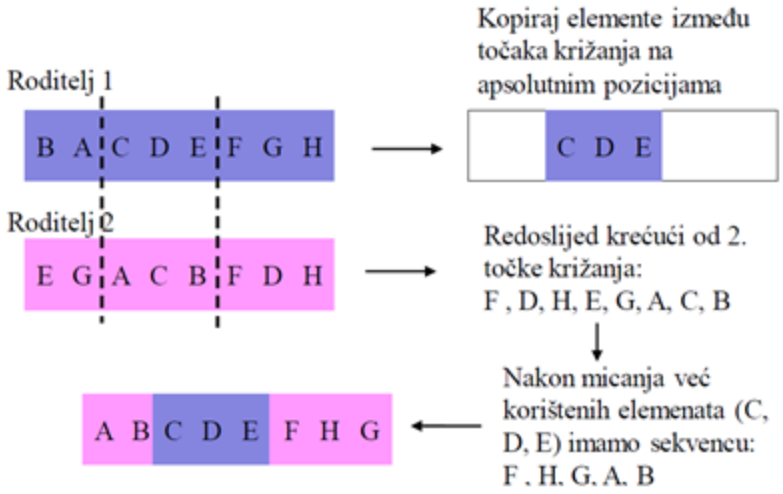
➔ roditeljski-centrično

- potomci su vrijednostima gena blizu jednom od roditelja, prilikom čega svaki od roditelja ima jednaku vjerojatnost da potomak bude u susjedstvu njegovih vrijednosti

c) Za prikaz s permutacijama

➔ slijedno križanje

- postupak križanja je sljedeći:
  1. nasumično odabrati dvije točke križanja
  2. kopirati gene između odabrane dvije točke križanja jednog od roditelja u kromosom potomka na iste apsolutne pozicije
  3. počevši od druge točke križanja na potomku i drugom roditelju, neupotrijebljenom u koraku 2, kopirati elemente tog drugog roditelja

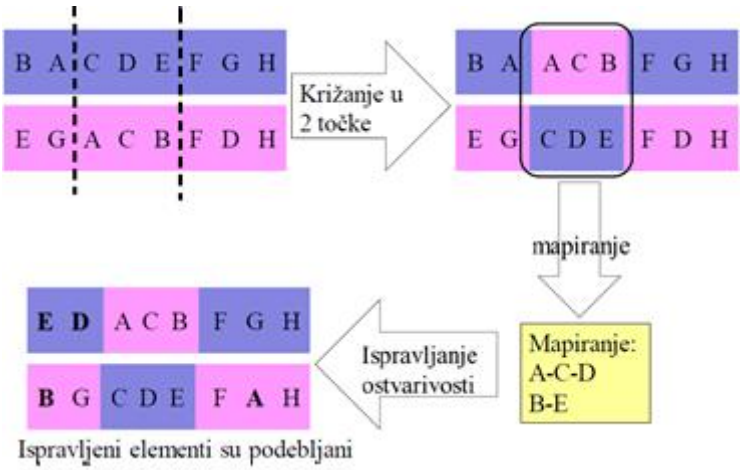


Slika 25. Primjer slijednog križanja za prikaz permutacijama [8]

➔ parcijalno mapirano križanje

- postupak križanja je sljedeći:

1. nasumično odabrati dvije točke križanja tako da se stvore tri segmenta s dva podniza jednake duljine u svakom roditelju (podnizovi se nalaze desno od prvog segmenta)
2. vrši se zamjena segmenata između roditelja nasumičnim odabirom jednog od dva podniza za zamjenu čime nastaju dva potomka koja najvjerojatnije nisu ostvariva
3. utvrđuje se relacija preslikavanja gena na osnovi zamijenjenih podnizova
4. temeljem utvrđene relacije iz prethodnog koraka, vrši se ispravak neostvarivosti potomaka na genima neizmijenjenih segmenata

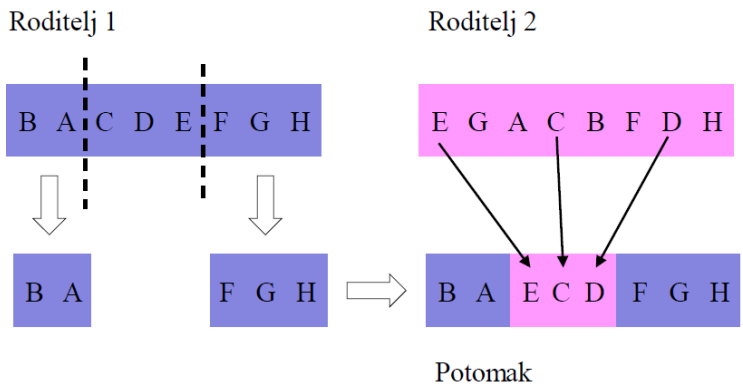


Slika 26. Primjer parcijalno mapiranog križanja za prikaz s permutacijama [8]

➔ križanje u dvije točke

- postupak križanja je sljedeći:

1. nasumično odabrati dvije točke križanja
2. elementi u vanjskim segmentima (oko centralnog) kopiraju se od jednog roditelja te se na tim apsolutnim pozicijama upisuju u kromosom potomka
3. ostale nepopunjene pozicije potomka popunjavaju se elementima drugog neupotrijebljenog roditelja iz koraka 2 koji su iznosom jednaki elementima iz neupotrijebljenog centralnog segmenta prvog roditelja tako da se prepisuju u redoslijedu u kojem se pojavljuju kod drugog roditelja



Slika 27. Primjer križanja u dvije točke kod prikaza s permutacijama [8]

2.8. Selekcija kod preživljavanja

Kod ove selekcije odlučuje se koje će od jedinki roditelja i njihovih potomaka preživjeti i prijeći u sljedeću generaciju. Glavni tipovi selekcije kod preživljavanja su [8]:

- zamjena temeljena na prikladnosti

a) roditelji i potomci rangiraju se temeljem njihova iznosa prikladnosti te se uzima određeni broj μ najboljih

b) stohastički se odabere određeni broj μ jedinki prema njihovoj relativnoj pogodnosti (metoda kotača za rulet, stohastičko univerzalno uzorkovanje)

c) eliminacijska selekcija: u svakoj iteraciji, vrlo mali broj, najčešće jedan, potomak zamjenjuje slaba rješenja u trenutnoj populaciji deterministički ili stohastički

- generacijska zamjena
  - potomci potpuno zamjenjuju populaciju roditelja u svakoj generaciji tako da se stvori onoliko potomka koliko ima roditelja te se svi roditelji maknu

Moguće je uočiti da ovi opći tipovi selekcije kod preživljavanja obuhvaćaju i tipove navedene u prethodnom poglavlju (2.6) o selekciji roditelja što ide u prilog tamo iznesenoj tvrdnji da se one metode mogu koristiti za obje selekcije. Ove dvije selekcije namjerno su odvojene kako bi se naglasilo da za svaku postoje različite metode te kako nije isključivo nužna primjena istih metoda i za jednu i za drugu, kao i da s obzirom na broj metoda koje postoje, nije moguće sve nabrojati.

2.9. Uvjet zaustavljanja

Evolucijski algoritmi ponavljaju proces rafinacije rješenja opisan u dosadašnjim poglavljima sve dok se ne postignu neki uvjeti za zaustavljanje kada dobiveno rješenje postaje ujedno i konačno rješenje postavljenog problema. Definiranjem tražene vrijednosti pogodnosti rješenja određuje se i točka zaustavljanja međutim, s obzirom da su evolucijski algoritmi stohastički, ta razina pogodnosti ne mora uvijek biti dostižna, stoga je potrebno definirati neke dodatne uvjete koji će rezultirati zaustavljanjem. Prema tome, točka zaustavljanja postiže se ukoliko je prekoračena tražena vrijednost pogodnosti ili ukoliko se ispune neki od sljedećih uvjeta [3]:

- prekoračenje maksimalno dopuštenog vremena korištenja procesora na računalu
- ukupan broj izračuna pogodnosti dosegao je postavljeni limit
- poboljšanje vrijednosti pogodnosti nalazilo se ispod definirane granične vrijednosti u nekom određenom vremenskom periodu (vremenski period može biti neki broj generacija ili broj ispitivanja pogodnosti)
- raznolikost generiranih populacija pada ispod neke definirane granične vrijednosti

3. EVOLUCIJSKE STRATEGIJE (ES)

Evolucijske strategije (ES) osmišljene su 1960. godine od strane Rechenberga i Schwefela koji su pokušavali osmisliti sheme za razvoj optimalnih oblika tijela s minimalnim trenjem u aerodinamici. Kod ES-a, potencijalna rješenja prikazuju se pomoću vektora realnih brojeva. Broj na određenom mjestu unutar vektora opisuje neku karakteristiku samog rješenja. Osnovno svojstvo EA-a je samoprilagodba parametara, što znači da se neki parametri u toku rada mijenjaju na određeni način tj. koevoluiraju zajedno s potencijalnim rješenjima te su zajedno s njima sastavni dio svakog rješenja. Najranije evolucijske strategije sastojale su se od algoritama s dva člana koji su radili u vektorskom prostoru, a označavali su se kao (1+1) evolucijske strategije. Kod njih se, potomci generiraju dodatkom nasumičnog broja (mutacija) svakom od elemenata roditeljskog vektora te procjenom je li došlo do poboljšanja pogodnosti. Ukoliko je došlo do poboljšanja, novonastali vektor se usvaja kao potomak, u suprotnom ne. Osim toga, također je nastao još jedan princip ES-a nazvan (1,1) evolucijske strategije kod kojih se čitavi roditeljski vektor uvijek u potpunosti mijenja. Nasumični brojevi koji se ubacuju ili kojima se mijenja čitavi vektor, vade se iz normalne (Gaussove) razdiobe sa srednjim vrijednosti jednakom nula te standardnom devijacijom σ, koja ovdje označava veličinu koraka mutacije.. Što se tiče križanja, osnovni princip kod ES-a uključuje dva roditelja koji stvaraju jedan potomak. Kako bi se došlo do λ potomaka, križanje se provodi λ puta. Postoje dva načina na koja se križanje može provesti: diskretno križanje ili srednje-križanje. Kod diskretnog križanja, jedan od elemenata (alela) svakog roditelja koji su svi s jednakom vjerojatnošću odabira, nasumično se odaberu te izmjene te tako nastane potomak. Kod srednjeg križanja, vrijednosti oba roditelja se uzmu te se odredi njihova srednja vrijednost za svaki pripadajući par elemenata (alela) čime se dobije novi potomak. Prilikom selekcije, kod ES-a ona se najčešće provodi metodom (μ, λ), po principu odabira λ najboljih pojedinaca iz populacije μ roditelja kojima se u potpunosti mijenja roditeljska populacija. Također postoji $(\mu + \lambda)$ metoda kod koje se stohastičkim odabirom iz populacije roditelja i potomaka $(\mu + \lambda)$ nova generacija bira deterministički na način da se zadrži top μ roditelja po rangu pogodnosti te oni tada tvore novu generaciju [3].

Prikaz	Vektori s realnim brojevima
Rekombinacija	Diskretna ili posredna
Mutacija	Perturbacijom pomoću normalne (Gaussove) raspodjele
Selekcija roditelja	Jednoliko nasumična
Selekcija kod preživljavanja	Deterministička elitistička zamjena po principu (μ, λ) ili $(\mu + \lambda)$
Posebnost	Samoprilagodljivost utjecaja mutacije

Slika 28. Posebnosti ES-a [3]

4. EVOLUCIJSKO PROGRAMIRANJE (EP)

Evolucijsko programiranje (EP) razvijeno je od strane Fogela, 1960., u sklopu njegova nastojanja da simulira evoluciju kao proces učenja s ciljem stvaranja umjetne inteligencije. Inteligenciju kojoj je težio, definirao je kao sposobnost sustava da prilagodi svoje ponašanje u svrhu postizanja postavljenog cilja u promjenjivim uvjetima okoline. Klasični sustavi EP-a, koriste prikaz pomoću tzv. strojeva s konačnim brojem stanja tj. konačnih automata. Konačni automat predstavlja matematički model koji se sastoji od konačnog broja mogućih stanja, definiranih prijelaza između tih stanja kao i definiranih akcija koje obavlja. U novije se vrijeme međutim, koriste prikazi pomoću realnih brojeva te se stoga EP-e gotovo stopilo s ES-a. Osnovna razlika između ova dva evolucijska algoritma je u tome da kod EP-a nema križanja pojedinaca tijekom evolucijskog rada. Nadalje, postoje razlike i u selekciji, jer kod EP-a svaki roditelj stvara točno jednog potomka ($\mu = \lambda$), ali se ti roditelji zajedno spajaju u zajedničku roditeljsku populaciju ($\mu + \mu$) iz koje se nova generacija stvara uz pomoć vjerojatnosti primjenom neke od metoda poput selekcije kotačem za rulet. Sustavi EP-a, dakle kao varijacijski operator koriste isključivo mutaciju koja radi po principu nasumičnog odabira pomoću normalne (Gaussove) vjerojatnosti kako bi stvorili nove potomke poboljšanog iznosa pogodnosti [3].

Prikaz	Vektori s realnim brojevima
Rekombinacija	Nema
Mutacija	Perturbacija pomoću normalne (Gaussove) raspodjele
Selekcija roditelja	Deterministička (svaki roditelj stvara jednog potomka pomoću mutacije)
Selekcija kod preživljavanja	Probabilistička po principu ($\mu + \mu$)
Posebnost	Samoprilagodljivost utjecaja mutacije

Slika 29. Posebnosti EP-a [3]

5. GENETSKI ALGORITAM (GA)

5.1. Općenito o GA-ima

Genetski algoritmi danas su najčešće upotrebljavani i najpoznatiji pripadnici evolucijskih algoritama. Prvi put spominju se 1967. godine od strane Johna H. Hollanda koji se bavio istraživanjima na području kompleksnih prilagodljivih sustava u sklopu strojnog učenja. Iako su prvotno zamišljeni kao alat pri istraživanju adaptivnog ponašanja, pokazali su se kao dobra metoda optimizacije matematičkih funkcija [3]. U slučaju primjene kod optimizacije, njihova je temeljna namjena pretraga prostora rješenja s ciljem pronalaženja globalnog optimuma. Spomenuta pretraga provodi se pomoću kombinacije stohastičkih i heurističkih metoda, kojima je cilj osigurati da algoritam tijekom tog postupka ne ostane zarobljen u nekom od lokalnih optimuma. Kaže se da je posrijedi primjena kombinacije stohastičkih i heurističkih metoda iz razloga što se radi o nasumičnom odabiru prilikom selekcije i provođenja reprodukcije, potpomognutom primjenom teorije vjerojatnosti i statističke fizike koji usmjeravaju postupak ka traženom rješenju po principu „pokušaja i pogrešaka“ [4, 9, 10]. Sve metode evolucije spomenute u prethodnim poglavljima primjenjuju se i pri radu genetskih algoritama. Princip njihova rada temelji se na održavanju populacije jedinki koju se može označiti sa $P(t)$, gdje t označava pripadajuću generaciju. Svaka jedinka predstavlja potencijalno rješenje postavljenog problema te se ocjenjuje u pogledu neke mjere pogodnosti. Neke od tih jedinki zatim podliježu stohastičkim transformacijama pomoću varijacijskih operatora koji stvaraju novu generaciju potomaka oznake $C(t)$ koji se procjenjuju s aspekta pogodnosti. Nova generacija $P(t+1)$ nastaje selekcijom najpogodnijih jedinki iz $C(t)$ i $P(t)$ populacije. Taj postupak ponavlja se nekoliko generacija dok se ne postigne neki od uvjeta prekida [2, 11]. Pseudo kod rada genetskog algoritam dan je slikom.

Procedura: Genetski algoritmi
Kreni
    $t \leftarrow 0$;
    inicijaliziraj $P(t)$;
    ocjeni $P(t)$;
    while (nije zadovoljen uvjet prekida) **do**
    begin
        rekombiniraj $P(t)$ da se dobije $C(t)$;
        ocjeni $C(t)$;
        odaberi $P(t+1)$ iz $P(t)$ i $C(t)$;
        $t \leftarrow t+1$;
    end

Slika 30. Pseudo kod rada genetskog algoritma [2, 11]

Genetski algoritmi posebno su zanimljivi kod rješavanja optimizacijskih problema u grani Industrijskog inženjerstva, gdje pronalaze brojne primjene kao što su: dizajniranje optimalnog sustava automatizacije proizvodnje, optimiranje redoslijeda obrade dijelova u proizvodnom pogonu, raspoređivanje aktivnosti kod rada s više projekata, optimizacija samog procesa obrade dijelova kao i još mnogo drugih zanimljivih primjena [11]. Zbog mogućnosti rješavanja brojnih problema u praksi, ovi algoritmi postali su područje velikog interesa za sve inženjere uključene u proizvodnju u svim granama industrije. U vrijeme začetka tzv. jednostavnog (kanonskog) genetskog algoritma, Holland je koristio binarno kodiranje kako bi prikazivao rješenja. Uz binarni prikaz, ti algoritmi koristili su selekciju temeljenu na iznosu pogodnosti, imali su nisku vjerojatnost mutacije p_m te naglasak stavljen na rekombinaciju (križanje) kao temeljnu metodu stvaranja novih potencijalnih rješenja [3]. U sljedećoj tablici prikazani su osnovni elementi kojima se opisuje rad jednostavnog genetskog algoritma.

Prikaz	Niz bitova (binarni)
Rekombinacija	Križanje u jednoj točki
Mutacija	Zamjena bita
Selekcija roditelja	Ovisna o iznosu pogodnosti – pomoću metode kotača za rulet
Selekcija kod preživljavanja	Generacijska zamjena

Slika 31. Posebnosti GA [3]

U nastavku će biti rečeno nešto više o specifičnostima ovog algoritma koje nisu spomenute u prethodnim poglavljima, a bitne su za razumijevanje ove tematike te će uz to biti dani konkretni primjeri njegova rada.

5.2. Prikaz i dekodiranje rješenja

Rješenja se kod genetskih algoritama mogu prikazati pomoću binarnih, cjelobrojnih ili realnih brojeva, a koji će se prikaz koristiti ovisi o postavljenom problemu za rješavanje. Izbor načina prikaza izravno utječe na učinkovitost rada algoritma, stoga se taj korak smatra vrlo važnim. U industrijskom inženjerstvu, rješavanje određenih problema upotrebom binarnog prikaza i kodiranja često je nemoguće. Problemi s kojima se inženjeri u proizvodnji najčešće susreću podrazumijevaju optimizaciju neke funkcije s ograničenjima ili određivanje optimalnog redoslijeda za što se koriste kodiranja realnim i cjelobrojnim vrijednostima [11]. Kod praktične primjene genetskih algoritama međutim, pokazalo se da binarni prikaz daje najbolje rezultate u pogledu učinkovitosti kod većine slučajeva gdje ga je moguće primijeniti, što je zajedno s činjenicom da model jednostavnog GA-a koristi binarni prikaz razlog zbog kojega se većina

teorije iznesene u literaturi bavi upravo njime. Kod takvog prikaza, kromosom (rješenje) je u obliku binarnog vektora koji predstavlja kodiranu vrijednost $x \epsilon [x_{dg}, x_{gg}]$. Dužina n tog kromosoma utječe na preciznost preslikavanja iz genotipskog u fenotipski prostor i obrnuto. Broj različitih kombinacija nula i jedinica koje je moguće zapisati u neki binarni vektor duljine n dan je izrazom 2^n. Temeljem toga, binarnim zapisom moguće je zapisati bilo koji broj koji se nalazi u intervalu $[0, 2^n - 1]$ [4]. Važno je spomenuti da se prilikom odabira odgovarajućeg prikaza genotipskog prostora, s kojim će algoritam baratati, taj prikaz ne mora slagati s prikazom fenotipskog prostora kojeg inicijalno zapisujemo i u kojeg će konačno rješenje biti potrebno preslikati dekodiranjem. Upravo navedeno je glavni razlog zbog kojega je potrebno uvesti neke postupke kodiranja i dekodiranja u nastavku:

a) želi li se neki binarni broj pretvoriti u odgovarajući cjelobrojni, koristi se izraz [4]:

$$cb = \sum_{i=0}^{n-1} B_i 2^i \quad \textbf{(3)}$$

b) ukoliko se binarni broj pretvara u neki realni broj u rasponu $[x_{dg}, x_{gg}]$, onda se to radi pomoću izraza [4]:

$$x = x_{dg} + \frac{cb}{2^n - 1}(x_{gg} - x_{dg}) \quad \textbf{(4)}$$

c) iz prethodnog izraza moguće je prikazati kako bi se vršila pretvorba realnog broja natrag u cjelobrojni:

$$cb = \frac{x - x_{dg}}{x_{gg} - x_{dg}} \cdot (2^n - 1) \quad \textbf{(5)}$$

d) Naposljetku ostaje pretvaranje cijelih brojeva u binarni zapis, što se radi na način da se taj broj dijeli sa 2 i zapiše cjelobrojni rezultat te se ostatak tog dijeljenja (koji je ili 1 ili 0) piše sa strane. Dijeljenje se provodi sve dok rezultat nije jedinica, nakon čega dobiveni ostaci predstavljaju binarni zapis broja.

Kada se u genotipskom prostoru koristi binarni zapis nekog realnog broja iz fenotipskog prostora, kodiranje i dekodiranje nije moguće provesti uz potpunu točnost, već se javlja greška. Da bi rješenje x imalo neku preciznost od p decimala, mora biti zadovoljena nejednakost [4]:

$$(x_{gg} - x_{dg}) \cdot 10^p < 2^n - 1 \quad \textbf{(6)}$$

Iz te nejednakost proizlazi da ukoliko se želi da neko rješenje x ima preciznost od p decimala, kromosom koji ga reprezentira mora biti duljine [4]:

$$n \geq \frac{log\left[\left(x_{gg} - x_{dg}\right) \cdot 10^{p} + 1\right]}{log2} \quad \textbf{(7)}$$

5.2.1. Prikaz Grayevim kodom

Binarno kodiranje ima i određenih nedostataka koji se tiču tzv. Hammingove udaljenosti, odnosno broja bitova u kojima se razlikuju neka dva susjedna broja koja su njima reprezentirana. Prikaz Grayevim kodom koristi se kako bi se Hammingova udaljenost između dva susjedna broja prikazana u binarnom zapisu smanjila na jedan, odnosno da bi se oni razlikovali samo u jednom bitu. Na primjer, ukoliko se promatraju dva susjedna broja u fenotipskom prostoru, 1023_{10} i 1024_{10} , njihovi binarni ekvivalenti genotipskog prostora su $(01111111111)_2$ i $(10000000000)_2$. Moguće je uočiti da je, iako se radi o susjednim brojevima, njihova Hammingova udaljenost maksimalna iz razloga što se binarni zapisi razlikuju u svim bitovima. Ukoliko se dogodi da je pronađeno rješenje u jednom koraku rada algoritma $(01111111111)_2$, a traženo optimalno rješenje kojem se ustvari teži $(10000000000)_2$, bilo bi potrebno promijeniti sve bitove [11]. Kako bi se tom problemu doskočilo primjenjuje se algoritam transformacije binarnog broja $b = b_1 b_2 \dots b_{m-1} b_m$ u Grayev kod $g = g_1 g_2 \cdots g_m g_{m-1}$.

TRANSFORMACIJA BINARNOG ZAPISA U GRAY KOD

Kreće se s desne strane zapisa od bita b_m te se promatra njegov prethodnik b_{m-1}. Ukoliko je $b_{m-1} = 1$ onda na lokaciji bita b_m mora biti zapisan kod $g_m = 1 - b_m$, međutim ako je $b_{m-1} = 0$, bit na lokaciji b_m ostaje nepromijenjen ($g_m = b_m$). U sljedećem koraku promatra se bit b_{m-1} i njegov prethodnik b_{m-2} itd. sve dok se ne dođe do bita b_1 i ne dobije kod g_1 na toj lokaciji. Konačno dobiveni kod je tzv. binarno reflektirani Gray kod [12].

Kod se naziv reflektiran zato što ga je moguće generirati na sljedeći način. Uzme li se Gray kod 0 i 1 te se on raspiše u oba smjera: 0, 1 i 1, 0, uz dodavanje nula prvoj polovici i jedinica drugoj polovici dobiva se 2-bitni Grayev kod: 00, 01, 11, 10. Nastavi li se, zapis prethodnog 2-bitnog koda u oba smjera izgleda 00, 01, 11, 10 i 10, 11, 01, 00. Ponovno uz dodavanje nula prvoj polovici i jedinica drugoj dobiva se 3-bitni Grayev kod: 000, 001, 011, 010, 110, 111, 101, 100 [12] [4].

5.3. Inicijalizacija i uvjet završetka

Populaciju potencijalnih rješenja predstavlja $VEL\_POP(t)$ binarnih vektora, gdje je t vrijeme ili redni broj generacije. Obično se uzima da je veličina te populacije konstantna tijekom rada algoritma (evolucije) te ona predstavlja jedan od osnovnih parametara algoritma. Proces inicijalizacije provodi se jednostavno generiranjem VEL_POP slučajnih brojeva u intervalu $[0, 2^n - 1]$ tj. binarnih vektora dužine n bitova. Evolucijski proces se ponavlja sve dok se ne zadovolji jedan od uvjeta zaustavljanja koji mogu biti: unaprijed zadani broj iteracija ili vrijeme trajanja optimiranja.

5.4. Funkcija dobrote

Funkcija dobrote ili funkcija ocjene kvalitete jedinke se u literaturi još naziva fitness funkcija, funkcija sposobnosti, funkcija cilja ili eval funkcija i u najjednostavnijoj interpretaciji ekvivalent je funkciji f koju treba optimizirati [4]:

$$dobrota\ (v) = f(x) \tag{8}$$

U prethodnom izraz binarni vektor v predstavlja realan broj $x \in [x_{dg}, x_{gg}]$. Što je dobrota jedinke veća, jedinka ima veću vjerojatnost preživljavanja i križanja. Funkcija dobrote je ključ za proces selekcije. Ta funkcija mora vjerno odražavati problem koji se rješava pa njezino definiranje predstavlja najveću poteškoću kod optimizacije nekog zadanog problema. Tijekom procesa evolucije, genetski algoritam generira, iz generacije u generaciju, populacije čije su ukupne i prosječne dobrote sve bolje i bolje. Ukupna dobrota populacije D i prosječna dobrota populacije $\overline{D}$ su definirane formulama [4]:

$$D = \sum_{i=1}^{VEL\_POP} dobrota\ (v_i) \tag{9}$$

$$\overline{D} = \frac{D}{VEL\_POP} \tag{10}$$

5.5. Postupak selekcije

Genetske algoritme, s obzirom na vrstu selekcije može se podijeliti na [4]:

- **Generacijske** – u jednoj iteraciji raspolažu s dvije populacije (što je ujedno i nedostatak GA) jer se odabiru jedinke iz stare populacije koje tvore novu te nakon selekcije sudjeluju u procesu reprodukcije
    - Karakteristične vrste selekcija koje pripadaju ovoj skupini su: jednostavna selekcija (selekcija metodom kotača za rulet, univerzalno stohastična metoda) i turnirska selekcija
- **Eliminacijske** – radi po principu traženja loših rješenja (kromosoma) u populaciji koje eliminira te reprodukcijom generira i na njihovo mjesto stavlja bolja rješenja (ovdje spada eliminacijska turnirska selekcija)
    - kod metoda eliminacijske selekcije, umjesto da je vjerojatnost odabira ili selekcije proporcionalna funkciji dobrote, ona je to veća što je dobrota manja, zbog čega je umjesto funkcije dobrote potrebno definirati funkciju kazne kako slijedi [4]:

$$p_k \approx kazna\ (v_k), k = 1,2, \dots, VEL\_POP \quad \textbf{(11)}$$

$$kazna\ (v_k) = \max_i \{dobrota(v_i)\} - dobrota\ (v_k) \quad \textbf{(12)}$$

p_k – vjerojatnost selekcije za eliminaciju

- time je riješen problem očuvanja kromosoma s najvećom dobrotom jer je njegova kazna jednaka nuli stoga i vjerojatnost eliminacije

Pojedine metode koje pripadaju ovim skupinama već su opisane u prethodnim poglavljima zbog čega nema potrebe za daljnjim proširivanjem ove teme.

5.6. Translacija ili pomak

Kod metoda jednostavne selekcije, jedan kromosom može se pojaviti više puta u sljedećoj populaciji. Što je jedinka bolja, to je veća vjerojatnost da bude odabrana u svakom od VEL_POP slučajnih izvlačenja. Ako je $dobrota\ (v_i) = 2 \cdot \overline{D}$ tada će se v_i najvjerojatnije pojaviti dva puta u slijedećoj generaciji. Takvim postupkom se "teoretski" može dogoditi da u slijedećoj populaciji, prije križanja i mutiranja, bude samo jedna jedinka u VEL_POP primjeraka. Pored tog nedostatka važno je spomenuti i to da funkcija $f(x)$ ne smije poprimiti

negativne vrijednosti ni za koji x_i jer bi tada $dobrota\ (v_i)$ bila negativna što nema smisla. Na prvi pogled, problem je riješen dodavanjem nekog velikog broja M funkciji $f(x)$ međutim, to bi učinilo funkciju dobrote približno jednakom za sve kromosome, odnosno učinilo bi ih sve podjednako dobrima i lošima kod selekcije, što ponovno nije dobro. Konačno rješenje pronađeno je u translaciji odnosno pomaku funkcije cilja. Funkciji cilja $f(x)$ dodaje se konstanta koja je jednaka minimumu zadane funkcije u zadanom intervalu:

$$dobrota\left(v_i(x)\right) = f(x) - \min\{f(x)\}, \qquad za\ i = 1,2,\dots,VEL\_POP \qquad \textbf{(13)}$$

Ta vrijednost minimuma nije ista za svaku generaciju, zbog čega ju je potrebno u svakom koraku ponovno računati. Ovim postupkom osigurano je da dobrota ne može poprimiti negativne vrijednosti te da su one približno jednake.

5.7. Parametri algoritma

Genetski algoritmi imaju sljedeće parametre: veličina populacije, broj generacija ili iteracija i vjerojatnost mutacije. Za generacijski genetski algoritam potrebno je još navesti i vjerojatnost križanja. Broj jedinki za eliminaciju zadaje se umjesto vjerojatnosti križanja kod eliminacijskog

genetskog algoritma. Vjerojatnost mutacije ovisi o broju jedinki za eliminaciju, jer upravo je toliko jedinki potrebno generirati križanjem. Uobičajene vrijednosti navedenih parametara koje se koriste kod „malih“ i „velikih“ populacija dane su sljedećoj slici.

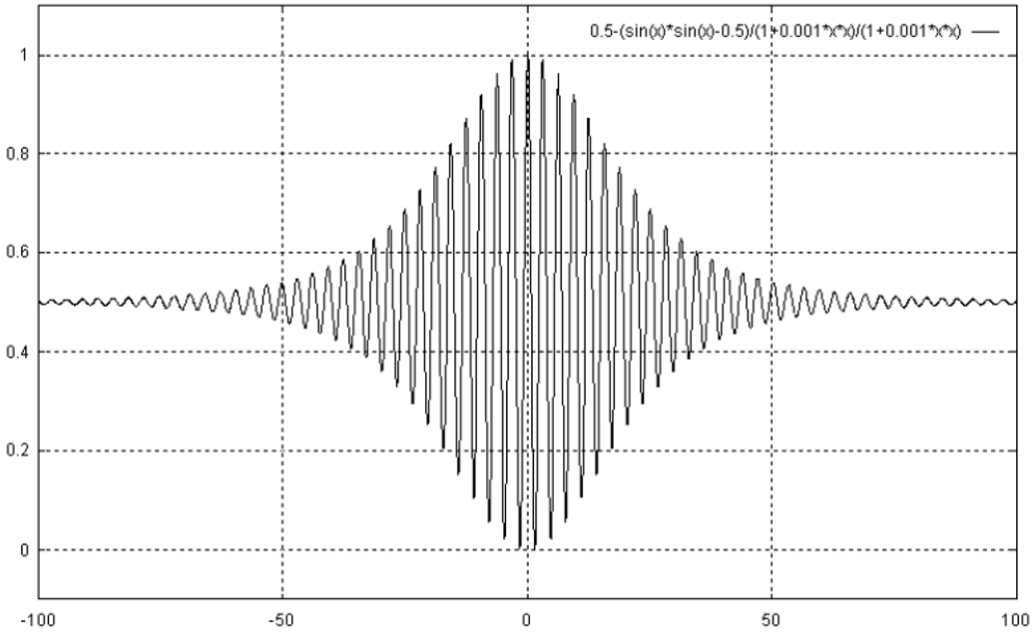
Parametri	oznaka	*"mala"* populacija	*"velika"* populacija
Veličina populacije	VEL_POP	30	100
Vjerojatnost mutacije	p_m	0.01	0.001
Vjerojatnost križanja	p_c	0.9	0.6
Broj jedinki za eliminaciju	M	VEL_POP/2	VEL_POP/4

Slika 32. Uobičajene vrijednosti parametara kod GA [4]

Za različite vrijednosti parametara algoritma, algoritam daje različite rezultate: brže ili sporije dolazi do boljeg ili lošijeg rješenja. Parametri genetskog algoritma ne moraju biti konstantni za vrijeme evolucije. Dinamički parametri mogu biti funkcija vremena ili broja iteracije (što je najčešći slučaj) ili mogu biti funkcija veličine raspršenosti rješenja. Ako su rješenja raspršena, valja povećati učestalost križanja, a smanjiti mutaciju. S druge strane, ako su rješenja uniformna potrebno je učiniti suprotno: povećati učestalost mutacije, a smanjiti učestalost križanja, jer križanjem dvaju istih rješenja neće se dobiti ništa novo, već samo treći, isti, kromosom.

5.8. Primjer rada jednostavnog eliminacijskog GA

Rad genetskog algoritma prikazan je korak po korak na jednostavnom primjeru optimiranja funkcije cilja jedne realne varijable prema [4]. Zadatak je pronaći globalni optimum funkcije prikazane na slici.



Ispitna funkcija $f(x) = 0.5 - \frac{\sin^2(x) - 0.5}{\left(1 + 0.001 \cdot x^2\right)^2}$

Slika 33. Zadana funkcija cilja za potragu optimuma pomoću GA [4]

Na prvi pogled, sa slike je vidljivo da se globalni optimum nalazi u točci (0, 1). Odabran je binarni prikaz duljine kromosoma od $n = 32$ bita čime je postignuta preciznost na $p = 7$ decimala. Broj jedinki u populaciji je 16, broj jedinki za eliminaciju 8 a broj selektiranih kromosoma za mutaciju, 5. Iako je odabrana veličina populacije suviše mala, ona je dovoljno dobra za demonstraciju principa rada GA. Kod odabranog kromosoma za mutaciju, mutira jedan do tri gena. Inicijalizacijom se generira početna populacija pomoću generatora slučajnih brojeva što se vidi na slici.

rbr	kromosom	x	f(x)
0	10001010011100000111111010011100	8.1558	0.1381
1	11000001001000011101000011011110	50.8844	0.5127
2	01110010101111011111001011111000	-10.3578	0.3813
3	10011110101110000110010000001010	24.0002	0.3712
4	10101100101110100000010010110100	34.9427	0.5727
5	10001111110110110011100100010110	12.3878	0.8521
6	11001000110000000110001011010000	56.8371	0.5234
7	01010100111001011101001100000010	-33.6736	0.4785
8	11100001101100011101010001001100	76.3239	0.4970
9	11000110001011101110110011001110	54.8307	0.4702
10	00100011011101011111100000101000	-72.2962	0.5129
11	00100001110001110011100101111010	-73.6108	0.4890
12	10000011101100110000010101100100	2.8901	0.9308
13	00101110001101100010010000000110	-63.8973	0.4897
14	10001110101010111000101100000000	11.4610	0.2667
15	10000101011001001000111101110010	4.2131	0.2386

Slika 34. Slučajno generirana populacija od 16 jedinki s evaluacijskim vrijednostima [4]

Prva iteracija.

1. korak: **evaluacija**. Izračunavaju se vrijednosti kromosoma x pretvaranjem binarnog zapisa u realni broj. Dobivena vrijednost uvrštava se u funkciju cilja i dobiva se f(x).

2. korak: **procjena**. Identificira se najbolja i najlošija jedinka i izvrši se translacija. U ovom slučaju najbolja jedinka je kromosom s rednim brojem 12, a najlošija je kromosom s rednim brojem 0. Izračunava se dobrota po formuli za translaciju te se, s obzirom da se radi o eliminacijskog selekciji, vrši izračun kazne.

3. korak: **selekcija**. Generira se M slučajnih brojeva u intervalu [0,max{kp}], gdje je max{kp} najveća vrijednost kumulativne dobrote. U prvom koraku, prije prve eliminacije, najveća vrijednost kumulativne dobrote iznosi 7.16. Ako je slučajno generirani broj r u intervalu [kpi-1,kpi] tada se i-ti kromosom briše. S pomoću generatora slučajnih brojeva generiran je broj 2.343. Za broj r=2.343 briše se jedinka s rednim brojem 4. Nakon svake eliminacije potrebno je ponovno izračunati kumulativnu dobrotu za sve jedinke koje slijede. U ovom slučaju eliminirana je jedinka s rednim brojem 4 i treba izračunati kumulativne dobrote jedinkama koje slijede: s rednim brojevima 5 do 15. Tek kada se izračuna novi (manji) max{kp}, generira se novi slučajni broj u novom intervalu. Vrijednost max{kp} je manja od one u prethodnom koraku za iznos kazne eliminirane jedinke. Generirani su još slijedeći slučajni brojevi: 0.17, 3.059, 4.261, 0.178, 3.002, 1.692 i 1.462. Postupak se ponavlja za svaki r. Na taj način u navedenom primjeru eliminirane su slijedeće jedinke 4, 0, 9, 14, 1, 11, 7 i 6.

Rb	d	kazna	kp	kromosom	x	f(x)	križanje
0	0.00	0.79	0.79	10001010011100000111111010011100	8.1558	0.1381	10+2
1	0.37	0.42	1.21	11000001001000011101000011011110	50.8844	0.5127	8+2
2	0.24	0.55	1.76	01110010101111101111100101111000	-10.3578	0.3813	
3	0.23	0.56	2.32	10011110101110000110010000001010	24.0002	0.3712	
4	0.43	0.36	2.67	10101100101110100000010010110100	34.9427	0.5727	13+12
5	0.71	0.08	2.75	10001111110110110011100100010110	12.3878	0.8521	
6	0.39	0.41	3.16	11001000110000000110001011010000	56.8371	0.5234	12+2
7	0.34	0.45	3.61	01010100111001011101001100000010	-33.6736	0.4785	3+8
8	0.36	0.43	4.05	11100001101100011101010001001100	76.3239	0.4970	
9	0.33	0.46	4.51	11000110001011101110110011001110	54.8307	0.4702	13+10
10	0.37	0.42	4.92	00100011011101011111100000101000	-72.2962	0.5129	
11	0.35	0.44	5.37	00100001110001110011100101111010	-73.6108	0.4890	3+12
12	0.79	0.00	5.37	10000011101100110000010101100100	2.8901	0.9308	
13	0.35	0.44	5.81	00101110001101100010010000000110	-63.8973	0.4897	
14	0.13	0.66	6.47	10001110101010111000101100000000	11.4610	0.2667	10+13
15	0.10	0.69	7.16	10000101011001001000111101110010	4.2131	0.2386	

Slika 35. Stanje populacije nakon selekcije, ali prije križanja (crno su označene jedinke za eliminaciju) [4]

4. korak: **križanje**. Križanjem preostalih jedinki popunjavaju se eliminirane jedinke. Veličina populacije je tijekom cijelog procesa optimiranja jednaka. Jedinke koje su preostale imaju međusobno jednaku vjerojatnost križanja. U prvom od M koraka se nadomješta jedinka s rednim brojem 4 i to križanjem slučajno odabranih jedinki 13 i 12. Jedinka pod rednim brojem 0 dobiva se križanjem slučajno odabranih jedinki 10 i 2, itd.

5. korak: **mutacija**. Selektira se BR_M=5 puta jedna jedinka za mutaciju. Jedna jedinka može više puta biti odabrana za mutaciju. Sve jedinke imaju istu vjerojatnost mutacije osim najbolje čija je vjerojatnost mutacije 0 (elitizam). Slučajnim odabirom selektirane su slijedeće jedinke za mutaciju: 2, 11, 2, 5 i 7 (jedinka s rednim brojem 2 je dva puta selektirana). Mutirani bitovi su u tablicama napisani masno.

rbr	kromosom	križanje
0	01100010011111011111100010101000	10+2
1	01100011101110011101011011011000	8+2
2	01110**0**10101111011111001010**1**11**1**00**0**	
3	10011110101110000110010000001010	
4	10101111001100100010010100100100	13+12
5	10001111110110**1**10011100100010110	
6	10100011101111110010010101110000	12+2
7	11010100101110010111010**1**00001110	3+8
8	11100001101100011101010001001100	
9	00101111001101000011110000101010	13+10
10	00100011011101011111100000101000	
11	100**1**10101011001000000101010010**1**0	3+12
12	10000011101100110000010101100100	
13	00101110001101100010010000000110	
14	00100110001101100011010000100100	10+13
15	10000101011001001000111101110010	

Slika 36. Populacija nakon prve iteracije [4]

Druga iteracija.

Nakon prve iteracije nije postignut bolji rezultat. Najbolja jedinka je ostala ista, ali je prosječna vrijednost funkcije cilja populacije porasla s 0.4828 na 0.5212.

rbr	d	kazna	kromosom	x	f(x)	križanje
0	0.15	0.54	01100010011111101111100010101000	-23.0531	0.3880	4+8
1	0.48	0.21	01100011101110011101011011011000	-22.0891	0.7215	12+5
2	0.12	0.58	01110110101111101111100100110101 0	-7.2328	0.3543	7+11
3	0.13	0.56	10011110101110000110010000001010	24.0002	0.3712	
4	0.25	0.44	10101111001100100010010100100100	36.8718	0.4925	
5	0.61	0.08	10001111111011010001110010010110	12.3847	0.8513	
6	0.38	0.31	10100011101111110010010101110000	27.9271	0.6213	4+8
7	0.28	0.42	11010100101110010111010100001110	66.1910	0.5157	
8	0.26	0.43	11100001101100011101010001001100	76.3239	0.4970	
9	0.28	0.41	00101111001101000011110000101010	-63.1218	0.5168	11+15
10	0.27	0.42	00100011011101011111100000101000	-72.2962	0.5129	5+3
11	0.10	0.59	10011010101100100000010101001010	20.8558	0.3436	
12	0.69	0.00	10000011101100110000010101100100	2.8901	0.9308	
13	0.25	0.44	00101110001101100010010000000110	-63.8973	0.4897	11+3
14	0.25	0.44	00100110001101100011010000100100	-70.1471	0.4932	3+12
15	0.00	0.69	10000101011001001000111101110010	4.2131	0.2386	

Slika 37. Populacija u drugoj iteraciji prije križanja [4]

Već u drugoj iteraciji postiže se poboljšanje. Jedinka s rednim brojem 1 dobivena križanjem jedinki 12 i 5 postaje najbolja. Neke jedinke (2,6,10 i 13) koje su bile pošteđene eliminacije u prvoj iteraciji sada su eliminirane. Do 278. generacije najbolja jedinka populacije predstavljala je prvi lokalni optimum, tj. maksimum: 01111100000000000000000000000000 ($x = -3.12499997744$). U 279-toj generaciji pronađeno je područje globalnog optimuma. Preostali dio procesa do 338. generacije je fino podešavanje rješenja. I konačno u 338. generaciji pronađeno je rješenje (globalni optimum): 10000000000000000000000000000000 ($x \cong 0$).

rbr	d	kazna	kromosom	x	f(x)	križanje
0	0.39	0.45	11100001001100001011010000100100	75.9299	0.5053	
1	0.84	0.00	10000111111100110001000100000110	6.2105	0.9587	
2	0.52	0.32	10011100101110000100010101001110	22.4373	0.6388	3+10
3	0.26	0.59	10011110101110000110010000001010	24.0002	0.3712	
4	0.38	0.47	10101111001100100010010100100100	36.8718	0.4925	
5	0.74	0.11	10001111111011010001110010010110	12.3847	0.8513	
6	0.32	0.52	10101101101100011101010100000100	35.6990	0.4368	10+5
7	0.40	0.44	11010100101110010111010100001110	66.1910	0.5157	3+0
8	0.38	0.46	11100001101100011101010001001100	76.3239	0.4970	15+3
9	0.10	0.74	10000001011001011000011101000010	1.0911	0.2137	12+1
10	0.25	0.60	10001110110110000110111000011110	11.5980	0.3610	
11	0.23	0.62	10011010101100100000010101001010	20.8558	0.3436	0+1
12	0.82	0.03	10000011101100110000010101100100	2.8901	0.9308	
13	0.25	0.60	10011110101100000110010100001010	23.9758	0.3636	10+1
14	0.53	0.31	10011111101110010000010100110010	24.7834	0.6469	15+3
15	0.00	0.84	10001101111001001000111100110010	10.8538	0.1158	

Slika 38. Populacija u trećoj iteraciji prije križanja [4]

iteracija	x	f(x)	kromosom
0	-6.09471749191	0.93219392743	011110000011001011100001000001010
2	-2.99842290184	0.97113253692	011111000010100101111101000001110
13	-3.02607852570	0.97792283178	011111000010000001101010001000010
14	-3.04753526651	0.98218118643	011111000001100101100001000110110
17	-3.12077973576	0.98997638820	011111000000000010110001000000101
23	-3.12135082277	0.98999591769	011111000000000010011001000011101
24	-3.12135110216	0.98999592708	011111000000000010011001000010111
26	-3.12137457149	0.98999671596	011111000000000010011000000011111
27	-3.12154258208	0.99000233173	011111000000000010010001000000111
31	-3.12364015335	0.99006777924	011111000000000000111001000010010
32	-3.12430553677	0.99008673589	011111000000000000011101001000001
33	-3.12440393100	0.99008946536	011111000000000000011001000000000
35	-3.12454623709	0.99009337933	011111000000000000010011000010000
40	-3.12468700649	0.99009721193	011111000000000000011101001000001
42	-3.12468924167	0.99009727247	011111000000000000011101000010001
45	-3.12468998673	0.99009729265	011111000000000000011101000000001
46	-3.12497590276	0.99010495561	011111000000000000000001000000101
49	-3.12497608902	0.99010496055	011111000000000000000001000000001
53	-3.12498782368	0.99010527162	011111000000000000000000100000101
54	-3.12499974461	0.99010558735	011111000000000000000000000000101
55	-3.12499993088	0.99010559229	011111000000000000000000000000001
246	-3.12499997744	0.99010559352	011111000000000000000000000000000
279	0.05464351272	0.99701408914	100000000001000111100111111010100
288	0.05268848037	0.99722373129	100000000001000101000011111010100
293	0.04963653629	0.99753578537	100000000001000001000011111010000
294	0.00081167091	0.99999934053	100000000000000001000100000010110
305	0.00001883600	0.99999999964	100000000000000000000000110010100
308	0.00001296867	0.99999999983	100000000000000000000000100010110
309	0.00001287553	0.99999999983	100000000000000000000000100010100
316	0.00000700820	0.99999999995	100000000000000000000000010010110
318	0.00000104774	1.00000000000	100000000000000000000000000010110
326	0.00000086147	1.00000000000	100000000000000000000000000010010
332	0.00000011642	1.00000000000	100000000000000000000000000000010
338	0.00000002328	1.00000000000	100000000000000000000000000000000

Slika 39. Primjer cijelog evolucijskog procesa [4]

6. GENETSKO PROGRAMIRANJE (GP)

6.1. Općenito o GP-u

Genetsko programiranje relativno je novi pojam unutar područja evolucijskih algoritama. Nastao je najkasnije od svih prethodno spomenutih pripadnika, 1992. godine, kada ga je računalni znanstvenik i sveučilišni profesor John R. Koza prvi puta spomenuto u svojoj knjizi naslova: *Genetsko programiranje: Programiranje računala primjenom prirodne selekcije*. Narednih godina, brojni znanstvenici razradili su niz različitih sustava temeljenih na idejama iznesenim u toj knjizi. Danas je zajednički naziv koji se koristi za sustave razvijene temeljem ideja iz te knjige upravo genetsko programiranje. Dok se prethodno obrađene metode, koje uključuju genetske algoritme, evolucijske strategije i evolucijsko programiranje, pretežito koriste za rješavanje optimizacijskih problema, GP se za razliku od njih, definira kao metoda direktne evolucije računalnih programa ili algoritama čija svrha leži u ostvarivanju učenja. Dakle, GP ne traži neku ulaznu veličinu kojom se postiže optimum neke funkcije, već traži čitavi model (program) koji rezultira najvećim iznosom pogodnosti. Stoga se GP vraća prvotno iznesenom cilju strojnog učenja kojeg je Samuel kao začetnik tog područja zacrtao, a to je učenje računala kako da se sama programiraju. Najveći cilj područja strojnog učenja, te posebice GP-a, je mogućnost da kažemo računalu koji zadatak želimo da ono izvrši te da ga se računalo samo nauči izvršavati. GP bi to ostvario na način da omogući računalu da programira samo sebe ili neka druga računala ovisno o potrebi. Danas GP još uvijek nema takvu razinu mogućnosti međutim, nakon objave Kozine knjige, došlo je do promijene u pogledu širine problema koji se mogu riješiti strojnim učenjem te su GP metode čak nadmašile performanse ostalih sustava strojnog učenja, dobivenih raznim drugim istraživanjima. Osim toga, GP kao metoda danas je u mogućnosti evoluirati programe koji su bolji od najboljih programa pisanih od strane ljudi, a koji služe za rješavanje čitavog niza problema u računalnom kao i industrijskom inženjerstvu. Granice između raznih pripadnika evolucijskih algoritama često su zamršene, pa to isto vrijedi i za granice između EA-a koji su GP i onih koji nisu. Najvažnije osobine koje su zajedničke svim GP sustavima su sljedeće [1]:

- **Stohastičko donošenje odluka**. GP koristi tzv. pseudo-nasumične brojeve (koriste se deterministički algoritmi za stvaranje tih „nasumičnih“ brojeva) kako bi se oponašala nasumičnost prirodne evolucije. U nekoliko stupnjeva razvoja novog programa, GP odluke donosi primjenom statistike i raznih stohastičkih procesa.

- **Programske strukture**. GP nove programske strukture varijabilne duljine stvara pomoću osnovnih jedinica koje se nazivaju operatori (eng. *function set*) i operandi (eng. *terminal set*). Stvaranje programa pomoću operatora i operanda započinje na samom početku rada GP-a, u trenutku stvaranja inicijalne populacije. Za prikaz populacija koriste se strukture u obliku stabla, linearne te graf strukture.
- **Genetski operatori**. GP inicijalne programe u populaciji evoluira pomoću genetskih operatora. Križanje između dva zasebna programa jedan je od osnovnih genetskih operatora u GP-u, a osim njega koriste se mutacija i reprodukcija programa. Križanje i mutacija provode se uz visoku vjerojatnost.
- **Simulacija evolucije populacije temeljem selekcije temeljene na pogodnosti**. GP razvija populaciju programa paralelno. Nit vodilja simulirane evolucije je pritom neki oblik selekcije temeljene na pogodnosti, kojom se biraju programi za daljnja poboljšanja.

U sljedećoj tablici prikazani su osnovni elementi kojima se opisuje rad genetskog programiranja. Više o samim elementima i principu rada biti će rečeno u poglavljima koja slijede.

Prikaz	Strukture u obliku stabla
Rekombinacija	Izmjena među podstablima
Mutacija	Nasumična promjena u stablima
Selekcija roditelja	U ovisnosti o iznosu pogodnosti
Selekcija kod preživljavanja	Generacijska zamjena

Slika 40. Posebnosti kod GP-a [3]

6.2. Operatori i operandi kao osnovne građevne jedinice GP-a

Operatori i operandi imaju različite uloge kod stvaranja strukture programa. Generalno gledano, operandi pružaju vrijednost sustavu, dok operandi procesuiraju vrijednosti dostupne u sklopu tog sustava. Zajedno, oni predstavljaju čvorove unutar prikaza pomoću stabla, a taj pojam koristi se i kod novijih načina prikazivanja kao što su linearne strukture i strukture u obliku grafa. Operandi se sastoje od ulaznih podataka, konstanta ili nultih funkcija [$f(0)$] kojima GP program barata. Engleski naziv za operand je *terminal* koji proizlazi riječi *terminate* što znači završi, jer se nalaze na završetku svake od grana u prikazu pomoću stabla, predstavljajući neke stvarne numeričke vrijednost koje mogu poslužiti kao ulazne informacije kod operatora. Operatori se sastoje od tvrdnji, matematičkih operatora te funkcija, koji su

dostupni promatranom GP programu. Odabir operatora ovisi o domeni problema koji se rješava GP programom te je njihova raznolikost vrlo velika. Neki primjeri operatora su sljedeći [1]:

- **Logički operatori**
  - AND, OR, NOT, XOR
- **Aritmetički operatori**
  - PLUS, MINUS, MULTIPLY, DIVIDE
- **Transcendentni operatori**
  - TRIGONOMETRIC, LOGARITHMIC FUNCTIONS
- **Operatori dodjeljivanja vrijednosti varijabli**
  - ukoliko je a neka varijabla GP sustava, $a \coloneqq 1$ bila bi funkcija koja dodjeljuje varijabli a vrijednost 1, a u stablu reprezentacija te funkcije bilo bi nešto oblika (ASSIGN a 1)
- **Uvjetni operatori**
  - IF, THEN, ELSE, CASE, SWITCH
- **Operatori za upravljanje prebacivanjem**
  - GO TO, CALL, JUMP
- **Operatori petlji**
  - WHILE … DO, REPEAT … UNITL, FOR … DO
- **Sub-rutine**
  - ovdje je list funkcija značajno opširniji nego kod prethodno navedenih jer se može upotrijebiti bilo koja funkcija koje se programer može dosjetiti
  - primjer robotske primjene, moguće je upotrijebiti: READ SENSOR, TURN LEFT, TURN RIGHT, MOVE AHEAD itd..

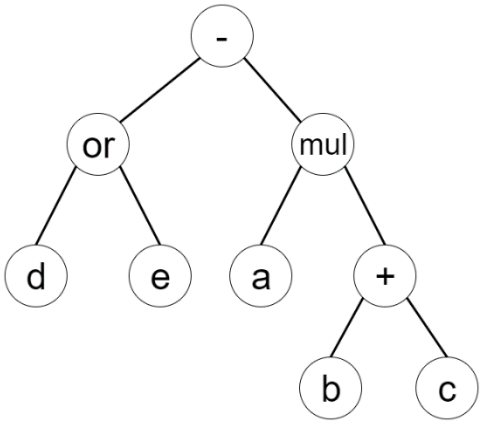
Kod odabira odgovarajućih operatora i operanda valja voditi računa o tome da oni moraju biti dovoljno pogodni kako bi prikazali rješenje problema koji se rješava. Na primjer, ako se set odabranih operatora koji će se koristiti sastoji samo od operatora zbrajanja (PLUS), neće se moći riješiti mnogo zanimljivih problema. S druge strane, odabirom previše različitih operatora dolazi do povećanja prostora pretraživanja što potragu za rješenjem otežava. Preporučeni set kojim se može krenuti u rješavanje problema sastoji se od aritmetičkih i logičkih operatora kako slijedi [1]: PLUS, MINUS, TIMES, DIVIDE, OR, AND, XOR.

Opseg problema koji se njima mogu riješiti iznenađujuće je velik, zbog čega se njihova upotreba i preporuča. Također pri odabiru operatora treba odabrati one koji su pogodni za baratanje očekivanim setom vrijednosti. Primjer pogrešnog odabira u tom pogledu bio bi DIVISION u slučaju da se pojavljuju vrijednosti jednake nuli, pošto dijeljenje s nulom nije moguće [1].

6.3. Prikaz strukture programa/modela

Svaki računali program može se prikazati kao stablo ili šuma stabala, gdje unutarnji čvorovi stabla imaju ulogu operatora, a listovi operanda. Programi su kod GP-a ustvari strukture operatora i operanda uz koje se nalaze pravila o tome kada će se i kako svaki od tih osnovnih građevnih jedinica izvršiti. Tri osnovne programske strukture koje GP koristi su:

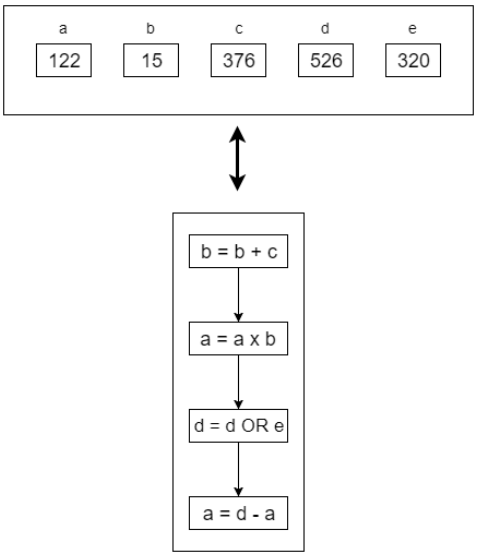
a) prikaz pomoću stabla



Slika 41. Prikaz strukture modela pomoću stabla [1]

Prikazano stablo može se izvršiti u bilo kojem redoslijedu međutim, postoje neke konvencije koje se obično slijede. Jedan od pristupa je da se izvršenje započinje odvijati od čvora koji je najdalje na lijevoj strani. Drugi pristup je kretanje od korijena stabla, dakle najgornjeg čvora prema dolje, kod kojeg se najprije izvršavaju operatori, a onda tek operandi.

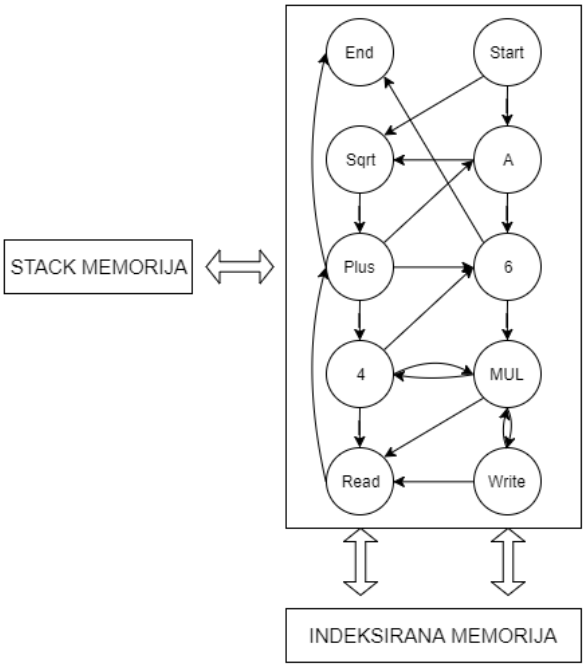
b) linearni prikaz



Slika 42. Linearni prikaz modela sa prostorom za pohranu [1]

Linearni prikaz jednostavno predstavlja niz uputa za izvršavanje koje se provodi odozgora prema dolje ili s lijeva na desno, ovisno o tom kako je struktura nacrtana. Za razliku od prikaza u obliku stabla, linearni prikaz zahtjeva prostor za pohranu ulaznih podataka jer ne sadrži potrebne podatke u samom prikazu.

c) prikaz u obliku grafa



Slika 43. Prikaz u obliku grafa [1]

Od 3 osnovna prikaza programskih struktura, ova je najnovija. Grafovi imaju mogućnost prikazivanja vrlo kompleksnih programskih struktura na kompaktan način. Naziv prikaza sa slike je PADO, a tu se radi o nizu čvorova povezanih strelicama. Ovakav prikaz ne samo da podržava petlje i rekurzije, već ih i potiče. Izvršenje ove strukture kreće od *Start* čvora, a završava kada se dođe do *End*. Poput linearnog prikaza i ovdje je potrebna zasebna memorija za pohranu varijabli.

Bez obzira na spomenuto, GP programske strukture često se nazivaju virtualnim strukturama iz razloga što u računalnoj memoriji izgledaju znatno drugačije, ali se izvršavaju po redoslijedu kao da izgledaju poput jedne od osnovnih struktura.

6.4. Inicijalizacija GP-a – Ramped half-and-half metoda

Inicijalizacija kod GP-a , tj. generiranje početne populacije rješenja kod najčešćeg prikaza u obliku stabla, može se provesti na nekoliko načina. Postupak je nešto drugačiji nego kod ostalih pripadnika evolucijskih algoritama jer je ograničavajući parametar maksimalna veličina programa koja se u prikazu stablom reprezentira maksimalnom dubinom samog stabla. Dubina stabla je minimalni broj čvorova koje je potrebno proći kako bi se od polazišnog čvora došlo do nekog konačnog tj. najudaljenijeg. Najpoznatija metoda inicijalizacije je tzv. *ramped half-and-half* metoda. Kod te metode bira se maksimalna inicijalna dozvoljena dubina D_{max} stabla te se zatim generira svaki član inicijalne populacije pomoću operatora i operanda jednom od sljedećih metoda [1] [3]:

- Full metoda
- Grow metoda

6.4.1. Full metoda

Sadržaj čvora na nekoj dubini d bira se iz seta operatora ako vrijedi $d < D_{max}$ ili iz seta operanda ako je $d = D_{max}$, počevši od korijenskog čvora (najgornji čvor).

6.4.2. Grow metoda

Kod ove metode, grana u stablu mogu imati različite dubine koje iznosom mogu ići do D_{max}. Stablo se generira počevši od korijena (najgornji čvor) izborom sadržaja pripadajućeg čvora na stohastički način iz skupa $Operatori \cup Operandi$ ukoliko je $d < D_{max}$.

6.5. Funkcija pogodnosti

Pogodnost je u slučaju GP-a, mjera koja tijekom simulirane evolucije ukazuje na to koliko je dobro evoluirani program naučio pretvarati dobivene ulazne veličine u tražene izlazne. Cilj uvođenja funkcije koja može dati tu vrijednost pogodnosti isti je kao i kod ostalih EA, a to je pružanje povratne informacije GP-u o tome koji pojedinci (programi) bi trebali imati veću vjerojatnost reprodukcije a koji veću vjerojatnost eliminacije. U primjeni kod GP-a moguće je definirati nekoliko svojstava koje funkcija pogodnosti može imati [1]:

a) **Kontinuirana pogodnost.** Svojstvo kod kojeg mala poboljšanja u tome koliko je dobro program naučio pretvarati ulazni podatak u traženi izlazni uzrokuju mala poboljšanja u iznosu pogodnosti dok veća poboljšanja, usporedno tome, uzrokuju veća poboljšanja iznos pogodnosti.

b) **Standardizirana pogodnost.** Kod funkcije pogodnosti, definira se da 0 označava najpogodnijeg pojedinca.

c) **Normalizirana pogodnost.** Kod funkcije pogodnosti, definira da se pogodnost mora nalaziti između 1 i 0.

6.5.1. Upotreba seta podataka kao izvora učenja GP sustava

Kako bi GP sustav mogao odraditi svoj posao, a to je pronalaženje modela koji na neke ulazne podatke daje ispravne izlazne podatke, potrebno ga je opskrbiti setom podataka koji će poslužiti kao izvor za učenje. Temeljem tih podataka, funkcija pogodnosti može pružiti iznos pogodnosti svake programske strukture koja nastaje radom algoritma. Primjer navedenog bio bi neki set podataka iz tablice, koji bi poslužili za trening sustava.

	Ulaz	Izlaz
Slučaj 1	1	2
Slučaj 2	2	6
Slučaj 3	4	20
Slučaj 4	7	56
Slučaj 5	9	90

Slika 44. Set podataka za trening GP sustava [1]

Recimo da se pomoću GP-a želi razviti model koji će predvidjeti iznos stupca izlaznih vrijednosti iz tablice uz poznavanje ulaznih. Rješenje ovog problema je trivijalno te je traženi

model opisan matematičkom funkcijom $f(x) = x^2 + x$. Jednostavna kontinuirana funkcija pogodnosti koja se može koristiti za ovaj problem uključivala bi zbroj apsolutne vrijednosti razlika između stvarno dobivenog izlaznog podatka nekog programa i traženog izlaznog podataka zadanog trening setom. Ukoliko se sa o_i označi i-ti izlazni podatak iz trening seta, a sa p_i i-ti izlazni podatak dobiven programom koji predstavlja potencijalno rješenje GP-om, pogodnost se može izraziti [1]:

$$f_p = \sum_{i=1}^{n} |p_i - o_i| \tag{14}$$

Ova funkcija pogodnosti je kontinuirana jer kako se p_i sve više približava o_i, to je pogodnost bolja sve dok ona za traženu funkciju ne iznosu nula. Alternativni način formiranja funkcije pogodnosti bio bi pomoću izračuna sume kvadriranih udaljenosti između p_i i o_i, što se naziva kvadrirana greška a izraz je oblika [1]:

$$f_p = \sum_{i=1}^{n} (p_i - o_i)^2 \tag{15}$$

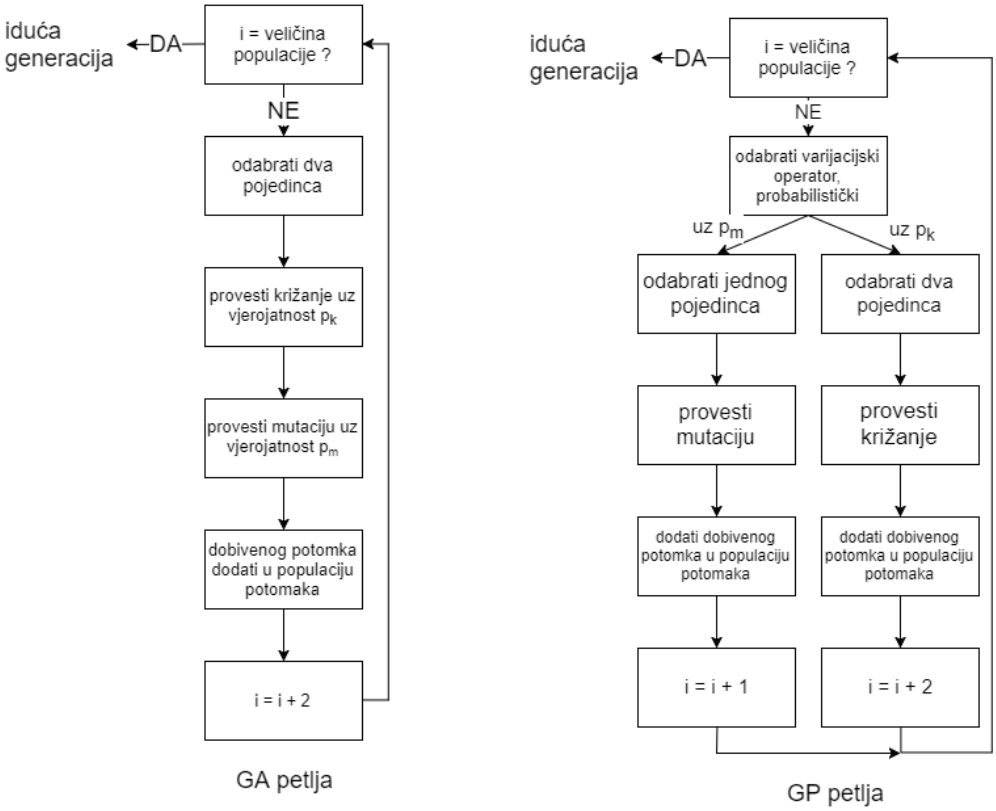
Kada je set podataka koji služi za učenje sastavljen od isključivo numeričkih ulaznih i izlaznih podataka, proces stvaranja programa pomoću GP-a naziva se simbolička regresija. Osim simboličke regresije, postoji i linearna regresija, ali za razliku od nje, simbolička osim traženja parametra modela traži i sam model, dok ga je kod linearne potrebno pretpostaviti.

6.6. Selekcija

Nakon što je kvaliteta pojedinca određena primjenom odgovarajuće funkcije pogodnosti, program GP-a mora odlučiti na koje pojedince uzeti za daljnju obradu varijacijskim operatorima. Metode selekcija jednake su već opisanim metodama u poglavlju evolucijskih algoritama, stoga ih nema potrebe ponavljati.

6.7. Varijacijski operatori kod GP-a

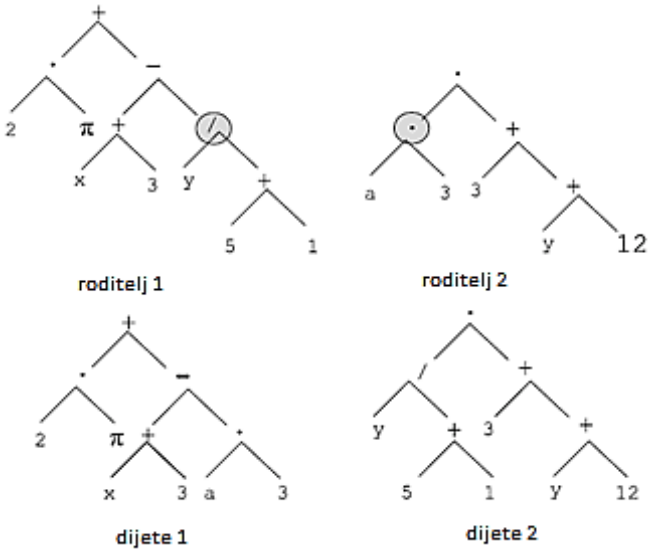
Kao varijacijski operatori koriste se: reprodukcija, križanje i mutacija. Oni imaju istu namjenu kao i kod ostalih EA. Za razliku od GA gdje se potomci stvaraju križanjem nakon kojeg slijedi mutacija, kod GP-a se potomci obično stvaraju ili mutacijom ili križanjem. Razlika je ilustrirana slikom na kojoj je prikazana tipična petlja GA i GP.



Slika 45. Usporedba primjene varijacijski operatora kod GA i GP petlje [3]

6.7.1. Križanje

Kod križanja dolazi do zamjene slučajno odabranih pod-stabala dviju odabranih jedinki, čime se svojstva stabala roditelja prenose na stabla djecu.



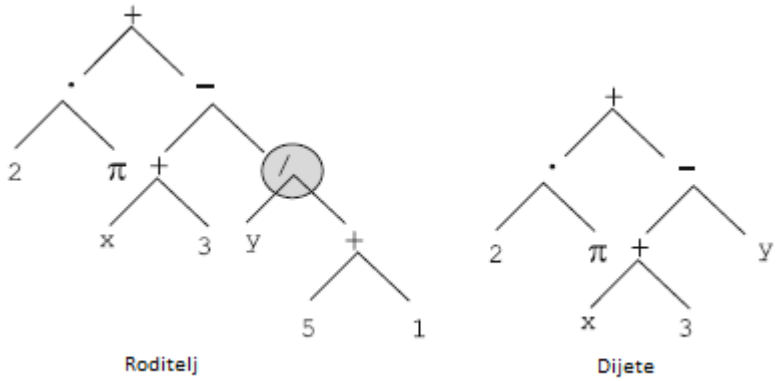
Slika 46. Primjer križanja kod prikaza stablom [3]

Važni parametri koji utječu na križanje su [3]:

- vjerojatnost odabira križanja kao genetskog operatora
- vjerojatnost odabira pojedine jedinke
- vjerojatnost odabira pojedinih čvorova jedinki kao korijena pod-stabala.

6.7.2. Mutacija

Mutacija kod prikaza stablom GP-a radi na principu odabira nasumičnog čvora u kojem se stablo „reže“ te umetanja novog nasumično generiranog pod-stabla na tu lokaciju.



Slika 47. Primjer mutacije kod prikaza stablom [3]

Važni parametri koji utječu na mutaciju su [3]:

- vjerojatnost odabira mutacije kao genetskog operatora
- vjerojatnost odabira pojedinog čvora jedinke kao korijena pod-stabla.

6.7.3. Reprodukcija

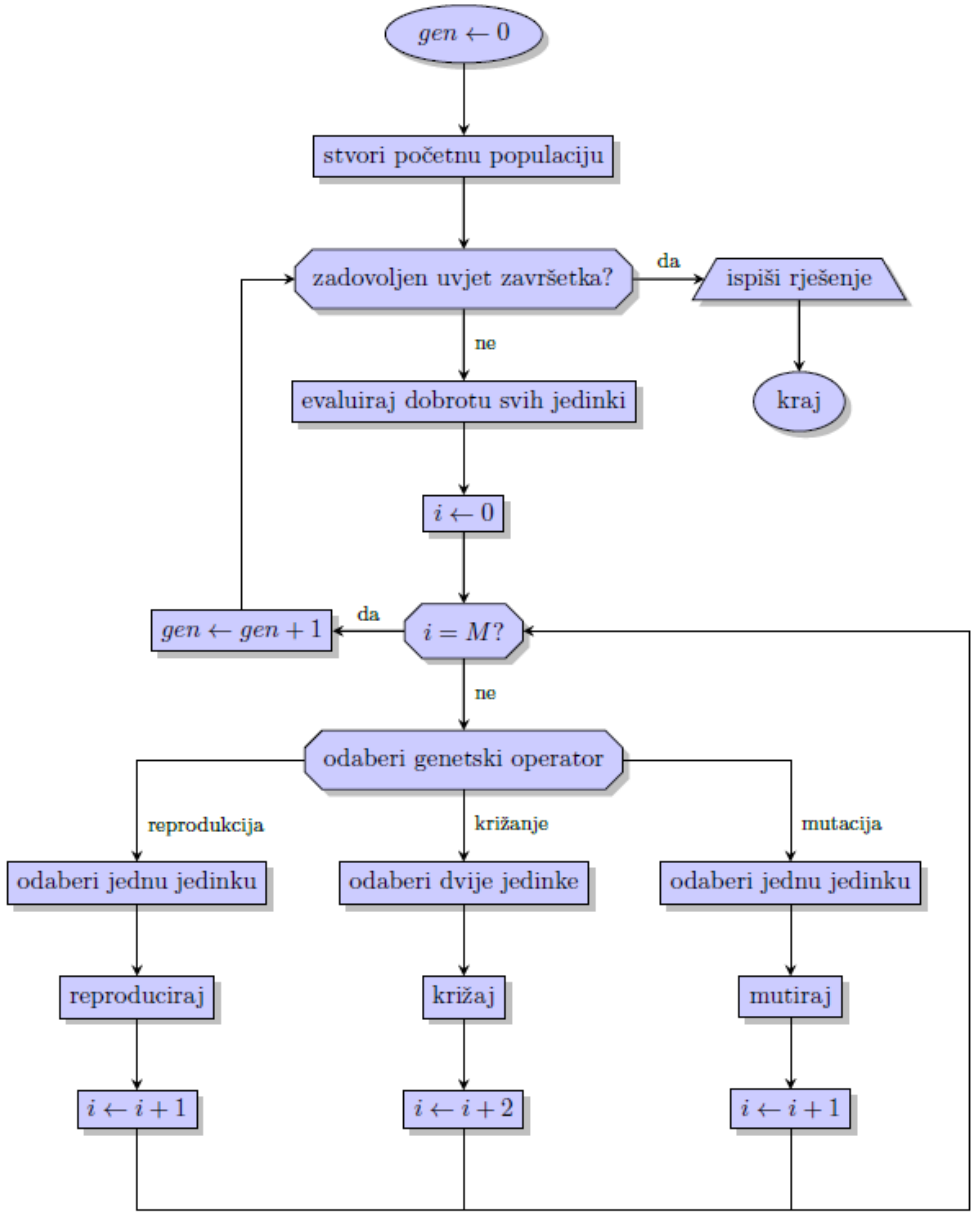
Reprodukcija je jednostavno kopiranje odabrane jedinke i umetanje iste u novu generaciju populacije. Semantika ovog genetskog operatora je preživljavanje odabrane jedinke, a očekivana posljedica je povećanje prosječne dobrote populacije.

Parametri kod reprodukcije su:

- vjerojatnost odabira reprodukcije kao genetskog operatora
- vjerojatnost odabira pojedine jedinke

6.8. Blok dijagram principa rada GP-a

Konačno moguće je, zbog bolje jasnoće, dati pregled principa rada genetskog programiranja prikazanog u blok dijagramu sa slike.



Slika 48. Blok dijagram principa rada GP-a [13]

7. OPTIMIRANJE PRIMJENOM GP-A

7.1. Regresijska analiza jednostavne matematičke funkcije pomoću GP-a

Postavljen je problem ispitivanja ovisnosti jedne zavisne varijable (regresanda) o jednoj nezavisnoj (regresorskoj) varijabli s ciljem utvrđivanja analitičkog izraza (modela) koji opisuje tu povezanost. Funkcija nad kojom se provodi analiza je sljedeća:

$$f(x) = x + x^2 + x^3 + x^4 \quad \textbf{(16)}$$

Navedena funkcija naziva se kvartni polinom te predstavlja funkciju koju je John Koza koristio u svojim eksperimentalnim razmatranjima GP-a. Kao set podataka za učenje kod ovog zadatka definirani su iznosi funkcije $f(x)$ u ovisnosti o varijabli x u tablici na temelju kojih će GP sustav provoditi procjenu pogodnosti modela. Interval iz kojeg su varijable x u trening setu podataka je $[-1,1]$.

x	$f(x)$
-1	0
-0,8947	-0,16963
-0,7895	-0,26978
-0,6842	-0,31722
-0,5789	-0,32547
-0,4737	-0,30525
-0,3684	-0,26426
-0,2632	-0,20736
-0,1579	-0,13628
-0,0526	-0,04997
0,0526	0,05552
0,1579	0,187391
0,2632	0,355506
0,3684	0,572537
0,4737	0,854738
0,5789	1,220338
0,6842	1,691769
0,7895	2,293429
0,8947	3,052166
1	4

Slika 49. Set podataka za trening GP sustava kod regresijske analize zadane funkcije [1]

Koraci GP-a:

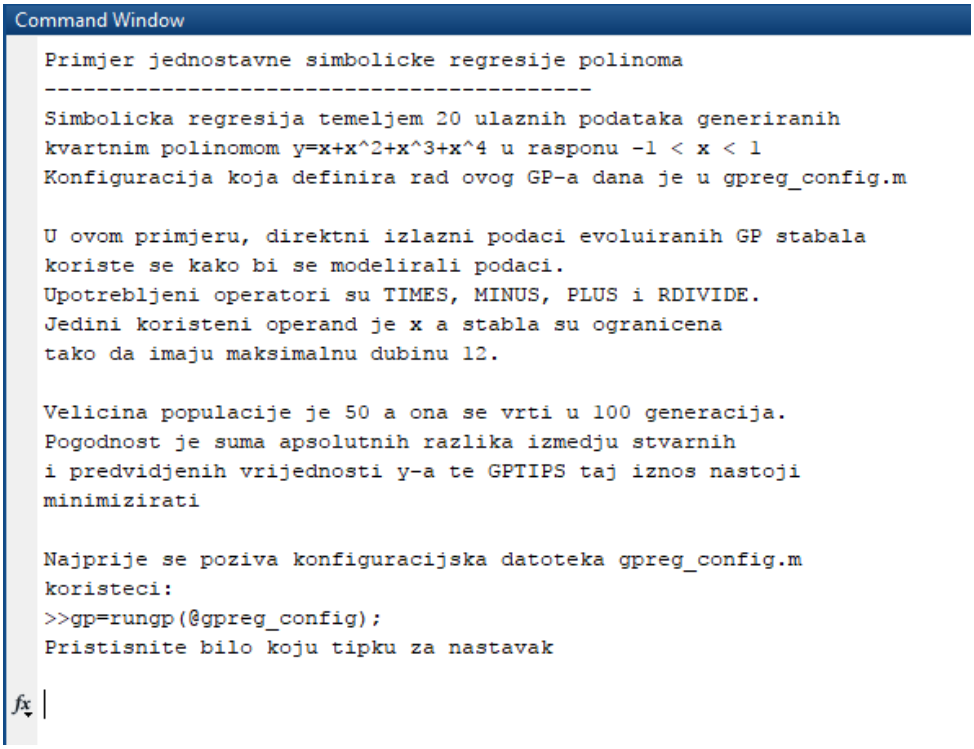
1) **Najprije se definiraju operandi i raspon u kojem smije biti nasumična konstanata**: operand je x, a konstante kod ovog problema nisu potrebne
2) **Zatim slijedi definiranje operatora:** aritmetički operatori +, -, *, /
3) **Formira se funkcija pogodnosti:** funkcija pogodnosti bazirana na srednjoj vrijednosti pogreške
4) **Definiranje ostalih parametara koji uključuju:** veličinu populacije, vrstu selekcije, uvjet prekida, maksimalni broj generacija, maksimalnu dubinu nakon mutacije, metodu inicijalizacije

Odabrane postavke GP sustava mogu se prikazati u jednoj preglednoj tablici.

Parametri	Vrijednosti
cilj:	dobiti funkciju koja će odgovarati trening setu
operandi:	x
operatori:	ADD, SUB, MUL, DIV
veličina populacije:	50
selekcija:	turnirska selekcija, veličine 2
maksimalni broj generacija:	100
maksimalna dubina stabla	12
maksimalna dubina nakon mutacije:	7
metoda inicijalizacije:	grow
uvjet prekida:	postignut minimalni iznos funkcije pogodnosti ili dosegnuto 100 generacija

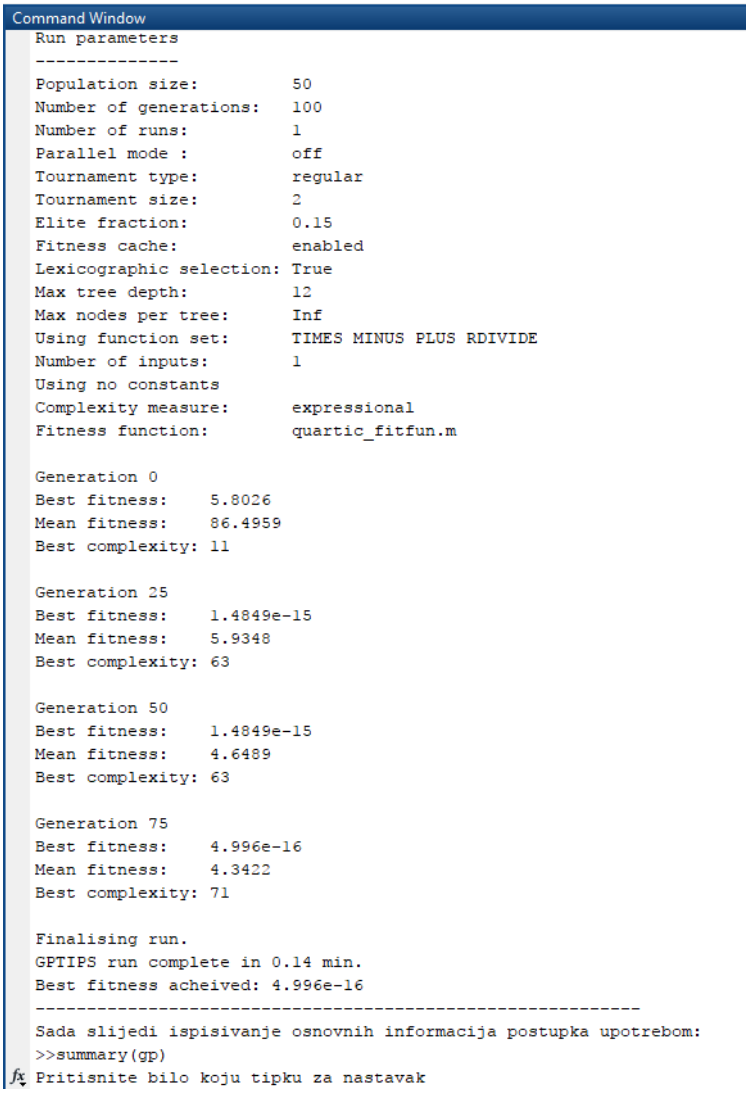
Postavljeni problem riješen je upotrebom MATLAB-a i instaliranog toolboxa naziva Genetic Programming & Symbolic Regression for MATLAB (GPTIPS) [14]. Upotrijebljeni osnovni kod za rad programa koji to izvodi dan je u prilogu ovog rada a kod za ostale funkcije koje se koriste kako bi se problem riješio pomoću GP metodologije, dobiven je zajedno sa navedenim toolbox-om.

Pokretanjem koda unutar datoteke gpreg.m dobiva se prozor sa sljedeće slike.



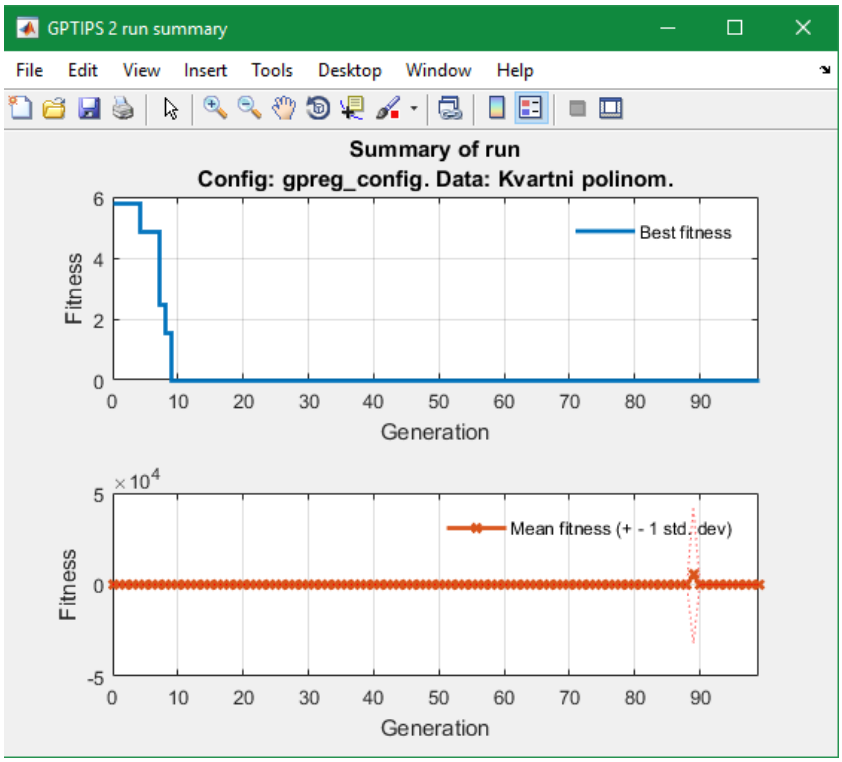
Slika 50. Pokretanje regresijske analize u MATLAB-u pomoću koda unutar gpreg.m datoteke

Nakon pritiska bilo koje tipke dobiva se ispis parametara pomoću kojih će se optimizacija raditi, kao i rezultat optimizacije u svakoj 25. generaciji od 100.



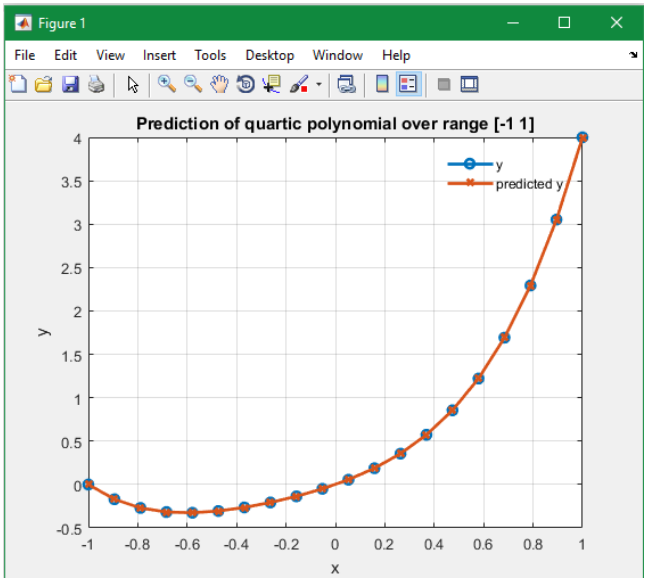
Slika 51. Ispis osnovnih parametara rada GP-a s rezultatima prosječne pogodnosti za svaku 25. generaciju

Nakon toga slijedi prikazivanje kretanja iznosa apsolutne i prosječne pogodnosti po generacijama tijekom izvođenja algoritma, što je prikazano na slici 52.



Slika 52. Iznosi apsolutne i prosječne pogodnosti kroz generacije

Zatim je moguće dobiti vizualni uvid u poklapanje predviđanja dobivenog najboljeg modela (narančasto) s traženim vrijednostima funkcije $f(x)$ (plavo) prikazano slikom 53.



Slika 53. Kretanje iznosa dobivenih najboljim modelom predviđanja (narančasto) i traženih vrijednosti (plavo)

Nadalje, program ispisuje izraz za najbolji dobiveni model, u kojoj je on generaciji pronađen, pri kojoj dubini stabla i pri kojem minimalnom iznosu pogodnosti.

```
Najbolji model postupka pohranjuje se u:
gp.results.best.eval_individual{1} :

plus(plus(plus(x1,times(x1,x1)),times(times(x1,x1),x1)),times(times(times(x1,x1),x1),x1))

Ovaj model ima dubinu stabla: 5
Pronadjen je u generaciji: 54
Ima iznos pogodnosti: 4.996e-16

Koristeci symbolic math toolbox moze se pronaci
pojednostavljena verzija izraza:
npr. pomocu naredbe GPPRETTY izvrsene nad najboljim modelom:
>>gppretty(gp,'best')
Pritisnite bilo koju tipku za nastavak
```

Slika 54. Izraz koji reprezentira najbolji dobiveni model s osnovnim podacima

Poslije toga, najbolji model pokušava pojednostaviti što rezultira izrazom sa slike. Ovdje je važno na umu imati napomenu koja je ispisana, a to je da taj izraz ne mora uvijek najbolje odgovarati traženom.

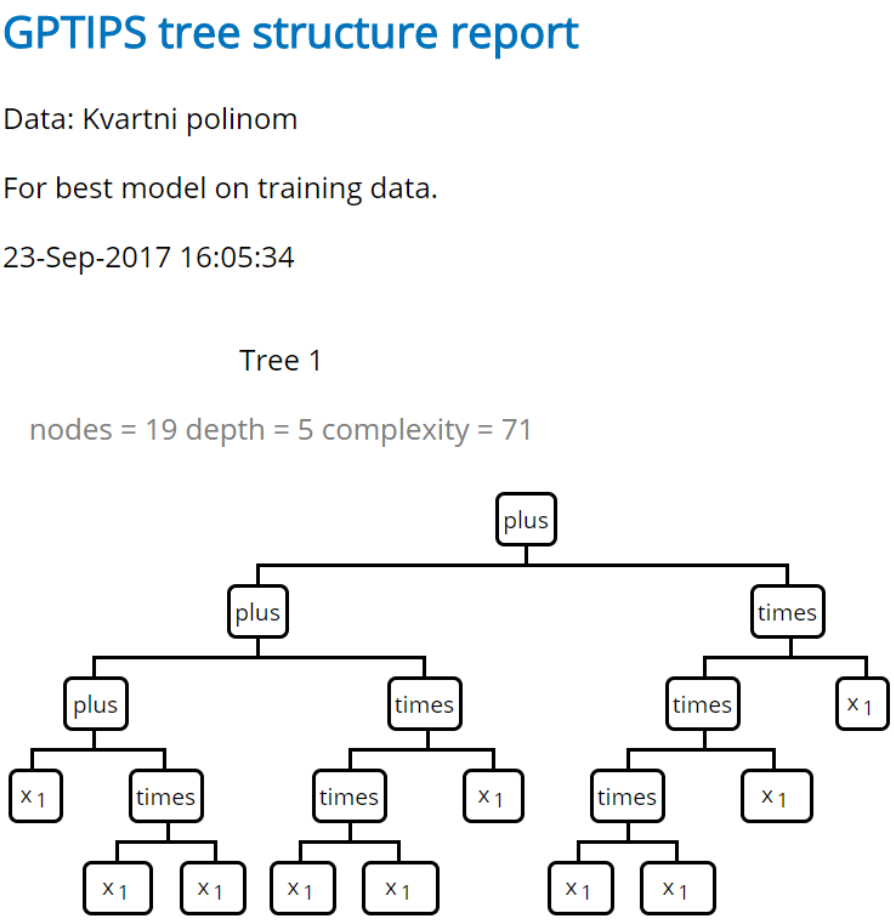
```
Simplified overall GP expression
--------------------------------
                  2
x1 (x1 + 1.0) (x1  + 1.0)


Dobiveni pojednostavljni izraz mogao bi biti trazeni izraz
upotrijebljen za stvaranje trening podataka.
NAPOMENA: Opcenito je neobicno da GP evoluira tocan oblik
funkcije upotrijebljene za generiranje trening podataka.

Sada slijedi vizualizacija najboljeg modela pomocu prikaza stablom:
>>drawtrees(gp,'best');
Pritisnite bilo koju tipku za nastavak
```

Slika 55. Pojednostavljeni izraz najboljeg modela

Na kraju se formira prikaz konačnog modela (rješenja) u obliku stabla.



Slika 56. Izgled konačnog rješenja u obliku stabla

Iz prethodne slike dobiva se izraz koji reprezentira konačno rješenje tj. traženi model kojim bi bilo moguće predvidjeti tražene iznose. Izraz je oblika:

$$f(x) = x_1^2 + x_1 + (x_1 \cdot x_1^2) + (x_1^2 \cdot x_1^2) = x_1^2 + x_1 + x_1^3 + x_1^4 \qquad \textbf{(17)}$$

Što odgovara traženom obliku kvartnog polinoma na temelju kojeg su izrađeni trening podaci.

8. ZAKLJUČAK

Evolucijski algoritmi pripadnici su područja strojnog učenja kao algoritmi koji nastoje oponašanjem biološke evolucije pronaći rješenja za niz različitih optimizacijskih problema. Temeljni mehanizmi biološke evolucije, poput križanja, mutacije, rekombinacije te selekcije upotrebljavaju se kao alati pomoću kojih ti algoritmi vrše pretragu često vrlo velikog prostora rješenja. Prvotni cilj strojnog učenja bio je osposobljavanje računala za samostalno programiranje. Taj cilj se u to doba možda pokazao preambicioznim međutim, danim pregledom algoritama, moguće je uočiti da se oni sve više počinju vračati i približavati tome cilju, posebice njihov najnoviji pripadnik, a to je GP. Nadalje, ovaj rad je kroz obradu načina prikaza rješenja ukazao na to da različiti problemi koje je moguće rješavati pomoću EA-a, zahtijevaju odabir odgovarajućeg načina prikaza jer u protivnom, dobivena rješenja ne bi imala smisla. Kod inicijalizacije EA-a, za rješavanje nekog realnog problema, korisno je posjedovati neko ekspertno znanje o domeni u kojoj se nalazi problem koji se rješava. To znanje može poslužiti kako bi se podesili parametri rada algoritma u svrhu postizanja što je veće moguće točnosti i efikasnosti. Unatoč prednostima koje to znanje donosi, kod vrlo složenih inženjerskih problema, ugrađivanje takvog znanja često je vremenski vrlo zahtjevno zbog čega ne nalazi praktičnu primjenu, osim u pogledu teorijskih ispitivanja i uspoređivanja. Bez obzira na to, postoje pripadnici EA-a, poput GP-a, kod kojih je posjedovanje ekspertnog znanja nužno kako bi se izvršio pravilan izbor operatora i operanda te definirala funkcija pogodnosti. Što se tiče selekcije, ona je kod EA-a, za razliku od inicijalizacije najčešće deterministička metoda bazirana na vjerojatnostima. Za osiguranje da se prilikom rada sagledavaju novi aspekti problema kao i da se sačuva ono što je do tada naučeno, zaslužni su varijacijski operatori. Čitavi postupak rada EA-a zaustavlja se postizanjem nekog od definiranih uvjeta završetaka. Ukoliko se traži ulazna veličina za koju neki model problema postiže optimum, najpogodnija je uporaba GA-ma međutim, traže li se i ulazna(e) veličina(e) zajedno sa modelom, tada je preporučljivo upotrijebiti GP. Teorijska podloga iznesena ovima radom trebala bi biti dovoljna da se odrede parametri potrebni za uspješan rad bilo kojeg EA-a, što je pokazano i na primjerima optimizacije funkcije GA kao i na regresijskoj analizi GP-om.

LITERATURA

[1] Banzhaf W., Nordin P., Keller E. R., Francone D. F.: Genetic Programming An Introduction, Dpunkt.verlag i Morgan Kauffmann Publishers, Inc., SAD, 1998.

[2] Yu X., Gen M.: Introduction to Evolutionary Algorithms, Springer, London, 2010.

[3] Eiben E.A., Smith E.J.: Introduction to Evolutionary Computing Second Edition, Springer, London, 2015.

[4] Golub M.: Genetski algoritam – Prvi dio, Fakultet elektrotehnike i računarstva, Zagreb, 2004.

[5] Tørensen J.: Evolutionary Algorithms – Introduction and representation, Department of Informatics, University of Oslo,

[6] Eiben E.A., Smith E.J.: Introduction to Evolutionary Computing, Springer, London, 2003.

[7] https://courses.cs.washington.edu/courses/cse473/06sp/GeneticAlgDemo/encoding.html , posljednji pristup 28.8.2017.

[8] Skorn-Kapov N.: Heurističke Metode Optimizacije: Evolucijski algoritmi, Fakultet elektrotehnike i računarstva, Zagreb, ak.god. 2014./2015.

[9] De Weck O.: A Basic Introduction to Genetic Algorithms, Massachusetts Institute of Technology, Massachusets

[10] http://www.zemris.fer.hr/~golub/ga/studenti/2012_domovic/diplomskiRad_Domovic.htm#_Toc327943751 , posljednji pristup 28.8.2017.

[11] Gen M., Cheng R.: Genetic Algorithms & Engineering Optimization, John Wiley & Sons, Inc., New York, 2000.

[12] http://mathworld.wolfram.com/GrayCode.html , posljednji pristup 28.8.2017.

[13] Kokan I.: Primjena genetskog programiranja za rješavanje problema prianjanja proteina, diplomski rad, Fakultet elektrotehnike i računarstva, Zagreb, lipanj 2010.

[14] Searson, D. GPTIPS: Genetic Programming & Symbolic Regression for MATLAB, http://gptips.sourceforge.net, 2009.

PRILOZI

I. CD-R disc

II. MATLAB kod GP-a

KOD UNUTAR GPREG.M DATOTEKE

```
%GPREG: Primjer jednostavne simbolicke regresije Kozinog kvartnog polinoma.
%
%   Ovaj primjer prikazuje jednostavnu regresiju te upotrebljava
%   funkcije kao sto su SUMMARY, RUNTREE and GPPRETTY za post analizu i
%   pojednostavljenje izraza ukoliko je instaliran Symbolic Math Toolbox.
%   Na kraju koristi DRAWTREES funkciju kako bi se vizualizirao najbolji
%   dobiveni model.
%
%
%


clc;
disp('Primjer jednostavne simbolicke regresije polinoma');
disp('-----------------------------------------');
disp('Simbolicka regresija temeljem 20 ulaznih podataka generiranih');
disp('kvartnim polinomom y=x+x^2+x^3+x^4 u rasponu -1 < x < 1');
disp('Konfiguracija koja definira rad ovog GP-a dana je u gpreg_config.m');
disp(' ');
disp('U ovom primjeru, direktni izlazni podaci evoluiranih GP stabala');
disp('koriste se kako bi se modelirali podaci.');
disp('Upotrebljeni operatori su TIMES, MINUS, PLUS i RDIVIDE.');
disp('Jedini koristeni operand je x a stabla su ogranicena');
disp('tako da imaju maksimalnu dubinu 12.');
disp(' ');
disp('Velicina populacije je 50 a ona se vrti u 100 generacija.');
disp('Pogodnost je suma apsolutnih razlika izmedju stvarnih');
disp('i predvidjenih vrijednosti y-a te GPTIPS taj iznos nastoji');
disp('minimizirati');
disp(' ');

disp('Najprije se poziva konfiguracijska datoteka gpreg_config.m');
disp('koristeci:');
disp('>>gp=rungp(@gpreg_config);');
disp('Pristisnite bilo koju tipku za nastavak');
disp(' ');
pause;
gp=rungp(@gpreg_config);

disp('Sada slijedi ispisivanje osnovnih informacija postupka upotrebom:');
disp('>>summary(gp)');
disp('Pritisnite bilo koju tipku za nastavak');
disp(' ');
pause;
summary(gp,false);

disp('Sada se najbolji dobiveni model upotrebljava na funkciji cilja:');
disp('>>runtree(gp,''best'');');
disp('Pritisnite bilo koju tipku za nastavak');
disp(' ');
pause;
runtree(gp,'best');

disp(' ');
disp('Najbolji model postupka pohranjuje se u:');
disp('gp.results.best.eval_individual{1} :');
disp(' ');
disp( gp.results.best.eval_individual{1});
disp(' ');
```

```
disp(['Ovaj model ima dubinu stabla: ' int2str(
getdepth(gp.results.best.individual{1}))]);
disp(['Pronadjen je u generaciji: ' int2str(gp.results.best.foundatgen)]);
disp(['Ima iznos pogodnosti: ' num2str(gp.results.best.fitness)]);

%Pomocu Symbolic Math toolbox-a moguce je pojednostavljenje
if gp.info.toolbox.symbolic

    disp(' ');
    disp('Koristeci symbolic math toolbox moze se pronaci');
    disp('pojednostavljena verzija izraza: ')
    disp('npr. pomocu naredbe GPPRETTY izvrsene nad najboljim modelom: ');
    disp('>>gppretty(gp,''best'') ');
    disp('Pritisnite bilo koju tipku za nastavak');
    disp(' ');
    pause;
    gppretty(gp,'best');
    disp(' ');
    disp('Dobiveni pojednostavljni izraz mogao bi biti trazeni izraz');
    disp('upotrijebljen za stvaranje trening podataka.');
    disp('NAPOMENA: Opcenito je neobicno da GP evoluira tocan oblik');
    disp('funkcije upotrijebljene za generiranje trening podataka.');
end

disp(' ');
disp('Sada slijedi vizualizacija najboljeg modela pomocu prikaza
stablom:');
disp('>>drawtrees(gp,''best'');');
disp('Pritisnite bilo koju tipku za nastavak');
disp(' ');
pause;
drawtrees(gp,'best');
```

KOD UNUTAR GPREG_CONFIG.M DATOTEKE

```
function gp = gpreg_config(gp)
%GPREG_CONFIG datoteka koja definira jednostavnu simbolicku regresiju.
%
%   Koristi se jednostavan kvartni polinom (y=x+x^2+x^3+x^4) iz knjige
%   Genetic Programming od John-a Koza-e iz 1992.
%
%   GP = GPREG_CONFIG(GP) definira parametre potrebne za rad GP-a na
%   postavljenom problemu kvartnog polinoma.
%
%   Izradjeno pomocu GPTIPS 2 toolbox-a - Copyright (c) 2009-2015 Dominic
Searson
%


%definiranje parametara velicine populacije, broja generacija te razmaka
%generacija u kojem ce se prikazivati renutni podaci tijekom rada
gp.runcontrol.pop_size = 50;
gp.runcontrol.num_gen = 100;
gp.runcontrol.verbose = 25;

%definiranje selekcije - turnirska velicine 2
gp.selection.tournament.size = 2;

%poveznica na funkciju pogodnosti
gp.fitness.fitfun = @quartic_fitfun;

%generiranje trening seta podataka, podaci se za x generiraju kao 20
brojeva
%jednakih medjusobnih razmaka, koji se nalaze izmedju -1 i 1, dok se y
%definira pomocu kvartnog polinoma
x=linspace(-1,1,20)';
gp.userdata.x = x;
gp.userdata.y = x + x.^2 +x.^3 + x.^4;
gp.userdata.name = 'Kvartni polinom';

%definira se broj operanada
gp.nodes.inputs.num_inp = 1;

%polinom koristen za ovaj primjer ne treba nikakve konstante
gp.nodes.const.p_ERC = 0;

%maksimalna dozvoljena dubina stabala
gp.treedef.max_depth = 12;

%maksimalna dubina pod-stabala generiranih mutacijom
gp.treedef.max_mutate_depth = 7;

%koristi li se vise gena
gp.genes.multigene = false;

%definiraju se operatori koji ce se koristiti
gp.nodes.functions.name = {'times','minus','plus','rdivide'};
```

KOD UNUTAR QUARTIC_FITFUN.M DATOTEKE

```
function [fitness,gp] = quartic_fitfun(evalstr,gp)
%QUARTIC_FITFUN funkcija pogodnosti koristena za primjer jednostavne
simbolicke regresije kvartnog polinoma y = x + x^2 + x^3 + x^4.
%
%   FITNESS = QUARTIC_FITFUN(EVALSTR,GP) daje vrijednost pogodnosti
%   simbolickog izraza sadrzanog unutar naredbe EVALSTR.
%
%   Copyright (c) 2009-2015 Dominic Searson
%
%   GPTIPS 2
%
%   See also GPDEMO1_CONFIG, GPDEMO1

%extract x and y data from GP struct
x1 = gp.userdata.x;
y = gp.userdata.y;

%evaluate the tree (assuming only 1 gene is suppled in this case - if the
%user specified multigene config then only the first gene encountered will
be used)
eval(['out=' evalstr{1} ';']);

%fitness is sum of absolute differences between actual and predicted y
fitness = sum( abs(out-y) );

%if this is a post run call to this function then plot graphs
if gp.state.run_completed
    figure;
    plot(x1,y,'o-','LineWidth',2,'color',[0 0.45 0.74]);grid on;hold on;
    plot(x1,out,'x-','LineWidth',2,'color',[0.85 0.33 0.1]);
    xlabel('x');ylabel('y');
    legend('y','predicted y');legend boxoff; hold off;
    title('Prediction of quartic polynomial over range [-1 1]');
end
```