1 Uvod **1**
2 Genetsko programiranje i evolucijski algoritmi **2**
2.1 Evolucija u prirodi 3
2.1.1 Minimalni uvjeti za evoluciju 4
2.1.2 DNA kao računalni program 4
2.2 Evolucijski algoritmi 5
2.3 Genetski algoritmi 6
2.4 Evolucijske strategije 7
2.5 Evolucijsko programiranje 7
2.6 Osnovna obilježja genetskog programa 8
2.6.1 Terminali i funkcije 9
2.6.2 Odabir skupa funkcija i skupa terminala 10
2.6.3 Izvršne programske strukture 11
2.6.3.1 Struktura stabla 11
2.6.3.2 Linearna struktura 12
2.6.3.3 Struktura grafa 12
2.6.4 Inicijalizacija GP populacije 12
2.6.4.1 Inicijalizacija stablastih struktura 13
2.6.4.2 Ramped half and half metoda 14
2.6.5 Genetski operatori 14
2.6.5.1 Križanje 14
2.6.5.2 Mutacija 17
2.6.5.3 Reprodukcija 17
2.6.6 Funkcija dobrote 18
2.6.7 Algoritam selekcije 19
2.6.7.1 GA scenarij 19
2.6.7.2 ES scenarij 19
2.6.8 Parametri genetskog programa 19
2.6.9 Uvjet zaustavljanja 20
2.6.10 Građa rješenja 20
2.6.11 Osnovni GP algoritam 21
3 Strojno učenje **22**
3.1 Prikaz problema 23
3.1.1 Boolean prikaz 23

3.1.2 Prikaz pragom 24
3.1.3 Prikaz slučaja 25
3.1.4 Prikaz stablom 26
3.1.5 Genetski prikaz 27
3.2 Operatori i strategije pretraživanja 28
3.2.1 Blind search pretraživanje 28
3.2.2 Hill climbing pretraživanje 29
3.2.3 Beam search pretraživanje 29
3.3 Učenje 30
3.4 Genetsko programiranje i strojno učenje 31
4 Učinkovitost GP-a u postupcima strojnog učenja 32
4.1 Strojno učenje i GP na primjeru igre šaha 33
4.1.1 KRK problem 33
4.1.1.1 Terminali i funkcije 34
4.1.1.2 Operatori i parametri GP –a 35
4.1.1.3 Rezultati 35
4.2 Problem raspoređivanja 37
4.2.1 Svojstva poslova 37
4.2.3 Ocjena rasporeda 38
4.2.4 Raspoređivanje pomoću GP-a 39
Slika 4.3: Izgled glavnog programa u ECF-u. 39
4.2.4.2 Genotip tree 39
Tablica 4.1: Popis čvorova za statički problem na jednom stroju 40
4.2.4.3 Evaluacija i funkcija dobrote 41
4.2.5 Rezultati 42
4.2.5.1 Validacija 42
4.2.5.2 Stagnacija 48
4.2.5.3 K-Fold Kros-validacija 48
5 Zaključak 51
6 Literatura 52

1 Uvod

Uporabom heurističkih algoritama možemo dobiti rješenja NP problema koja nisu optimalna, ali su prihvatljiva za naše potrebe. Genetsko programiranje (GP) je jedan od načina rješavanja takvih problema, bez davanja konkretnih uputa o postupku rješavanja, uz mogućnost definiranja načina na koji će računalo naučiti učinkovito riješiti problem. Genetsko programiranje se svrstava u širu skupinu algoritama koja se zove evolucijsko računanje (eng. *evolutionary computing*). Temelji se na Darwinovoj teoriji o postanku vrsta (razvoj svih životnih oblika procesom prirodne selekcije). Metodama križanja, mutacije i prirodne selekcije na kraju dobivamo najbolju jedinku iz populacije, odnosno dobivamo najbolje rješenje za dani problem.

GP se može efikasno upotrijebiti za rješavanje problema strojnog učenja što ćemo vidjeti u daljnjim poglavljima. GP predstavlja problem kao skup svih mogućih računalnih programa ili podskup tog skupa. Budući da se sustavi strojnog učenja mogu pokretati na računalu, odnosno prikazati kao računalni program, GP može evoluirati rješenje pronađeno pomoću takvog sustava.

Pokazat ćemo važnost testiranja algoritma GP-a, njegove sposobnosti generalizacije, na skupu podataka različitom od skupa za učenje. Ispitivanje na istom skupu je pristrano u smislu da je razumljivo da ćemo dobiti dobre rezultate pošto se algoritam već specijalizirao za taj skup. Ako koristimo dva skupa podataka, ne možemo spriječiti samu pojavu prenaučenosti, specijalizacije. Uporabom dva skupa podataka možemo vidjeti koliko je rješenje dobro, odnosno koliko dobro GP generalizira podatke koje nije vidio u procesu učenja.

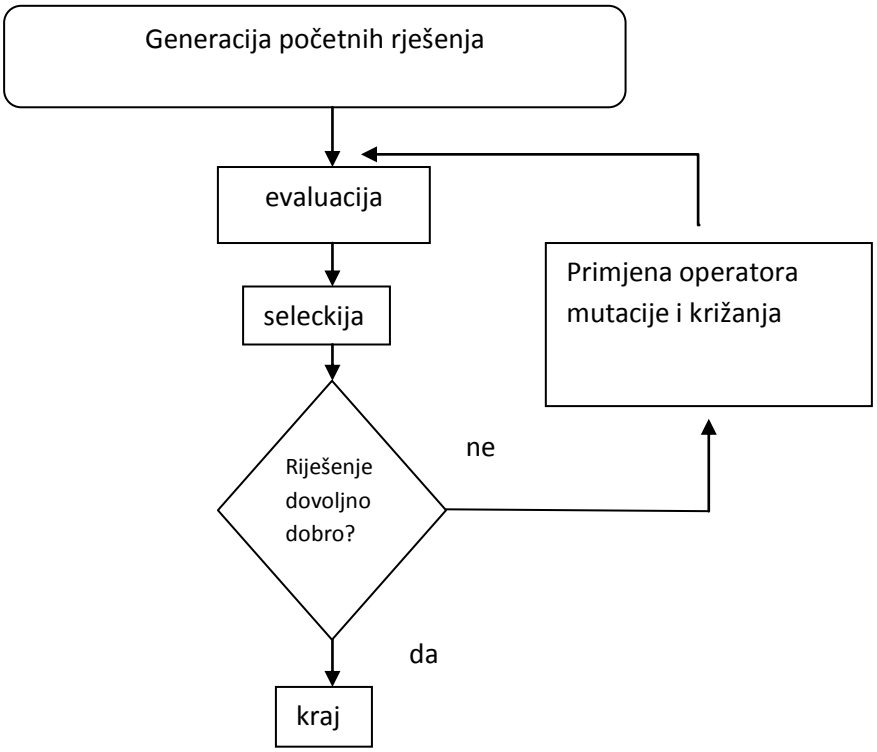
Moguće je upotrijebiti još jedan skup, skup za validaciju. Na ovome skupu algoritam ispituje kada bi trebao prestat učiti na skupu za učenje, odnosno kada se pojavila prenaučenost. Problem koji rješavamo jest otkrivanje pogodnog uvjeta zaustavljanja prilikom strojnog učenja uz pomoć genetskog programiranja. Otkrivanje uvjeta temelji se na broju generacija i uporabi validacijskog skupa. Cilj rada jest unaprijediti svojstva strojnog učenja izbjegavanjem prenaučenosti.

2 Genetsko programiranje i evolucijski algoritmi

Genetsko programiranje je jedna od mnogih tehnika za računalnu simulaciju evolucije i dio je šire skupine algoritama zajedničkim imenom nazvanim evolucijsko računanje. Evolucijsko računanje je skup postupaka koji oponašaju prirodni evolucijski proces na računalu. Ostvareni optimizacijski program koji se izvodi na računalu i oponaša evolucijski proces nazivamo evolucijskim algoritmom. Isto tako, računalni program koji izvodi postupak genetskog programiranja nazivamo genetskim programom.

Postupci evolucijskog računanja podijeljeni su na četiri glavne skupine: genetski algoritmi, genetsko programiranje, evolucijske strategije i evolucijsko programiranje. Posebnost genetskog programiranja, u odnosu na ostale navedene tehnike, je u činjenici da jedinke u populaciji genetskog programa predstavljaju računalne programe, odnosno strukture koje se mogu jednoznačno preslikati u oblik pogodan za izvođenje na računalu. Evolucijski algoritmi rade na način da se definira cilj u obliku kriterija kvalitete te se taj cilj rabi za mjerenje i usporedbu različitih kandidata rješenja. Evolucijski algoritam će pronaći optimalno ili približno optimalno rješenje nakon određenog broja iteracija.

Kriterij kvalitete je obično poznat kao funkcija dobrote te pomoću njega odlučujemo koje ćemo rješenje odabrati. U evolucijskom postupku važno je imati nekakav mehanizam varijacije kako bi stvorili razliku i bili sigurni da sljedeće generacije potomaka ne postanu identične kopije roditelja. U slučaju da takav mehanizam ne postoji daljnja poboljšanja rješenja ne bi bila moguća. Dva moguća operatora varijacije u evolucijskim algoritmima su *mutacija* i *križanje*, odnosno razmjena genetskog materijala između jedinki. Mutacija mijenja mali dio jedinke dok križanje miješa genetski materijal između dviju jedinki kako bi kreiralo potomka koji je kombinacija svojih roditelja. Osnovni evolucijski algoritam prikazan je na slici 2.1.



Slika 2.1: Prkaz osnovnog evolucijskog algoritma

Evolucijski algoritmi se upotrebljavaju za rješavanje problema čija je optimalna rješenja teško pronaći postojećim algoritmima jer povećanjem broja mogućih stanja složenost raste toliko da možemo reći da svemir nije dovoljno star u odnosu na trajanje koje nam treba da riješimo zadani problem.

2.1 Evolucija u prirodi

Prirodna evolucija je rezultat dugog procesa učenja iz kolektivnog iskustva generacija populacija organizama. Drugim riječima, svako živo biće je rezultat milijuna godina učenja svojih predaka kako preživjeti na Zemlji dovoljno dugo kako bi se mogli razmnožavati. Informacije naučene kroz biološku evoluciju smještene su u DNA. Sekvence DNA parova djeluju slično kao instrukcije u računalnom programu, zato je biološka evolucija dobar model za računalno rješavanje problema.

2.1.1 Minimalni uvjeti za evoluciju

Darwin *[C Darwin, „On the Origin of Species" 1859]* je rekao: *ako se pojavi varijacija korisna za neki organizam, sigurno će jedinke, tako karakterizirane, imati najbolju šansu da budu očuvane u borbi za život; i po principu nasljeđa proizvodit će potomstvo sličnih karakteristika.*

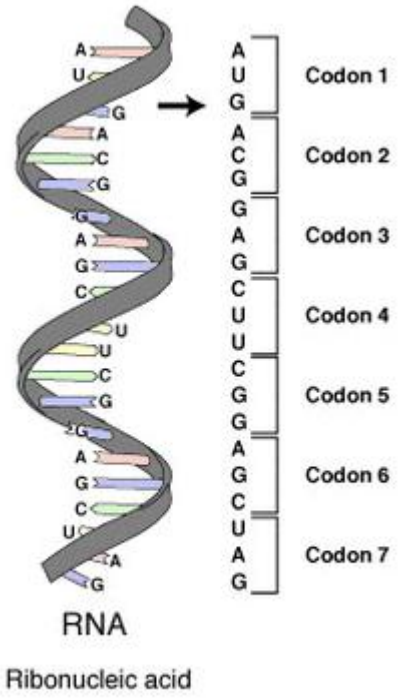
Ovaj princip očuvanja nazvao je *prirodna selekcija*.

Drugim riječima postoje četiri bitna preduvjeta za evoluciju prirodnom selekcijom:

1. reprodukcija jedinki u populaciji
2. varijacija koja utječe na vjerojatnost preživljavanja jedinki
3. nasljedstvo u reprodukciji
4. ograničena sredstva koja uzrokuju natjecanje

2.1.2 DNA kao računalni program

DNA, osnovni dio genoma, možemo gledati kao skup instrukcija za kreiranje organizma. Broj instrukcija koji se nalazi u DNA daleko nadilazi broj linija koda sveukupnog softvera koji je ikada napisan, ipak mehanizam kojim DNA pohranjuje instrukcije za kreiranje organizama je jako jednostavan. Osnovna jedinica genetskog koda je DNA par. Tri osnovna DNA para tvore takozvani **kodon** koji određuje produkciju amino kiselina. Sekvence kodona kodiraju pretvaranje amino kiselina u RNA, polipeptide i proteine. Ovi produkti su posrednici za rast i razvoj organizama. Postoje četiri različita osnovna para u DNA, adenin (A), guanin(G), citozin(C) i timin(T). U računalnom programu, *bit* je dio instrukcije procesoru. Cijela instrukcija sastoji se od sekvence bitova. Isto je tako i u DNA. Svaka instrukcija u DNA se sastoji od sekvence tri osnovna para koji čine kodon. Na slici 2.2 je prikazana ribonukleinska kiselina kao rezultat spajanja više kodona.



Slika 2.2: Rezultat spajanja više kodona, ribonukleinska kiselina

2.2 Evolucijski algoritmi

Osnovni dijelovi evolucijskih algoritama:

- populacija rješenja
- operatori mutacije
- operatori križanja
- funkcija dobrote
- selekcija

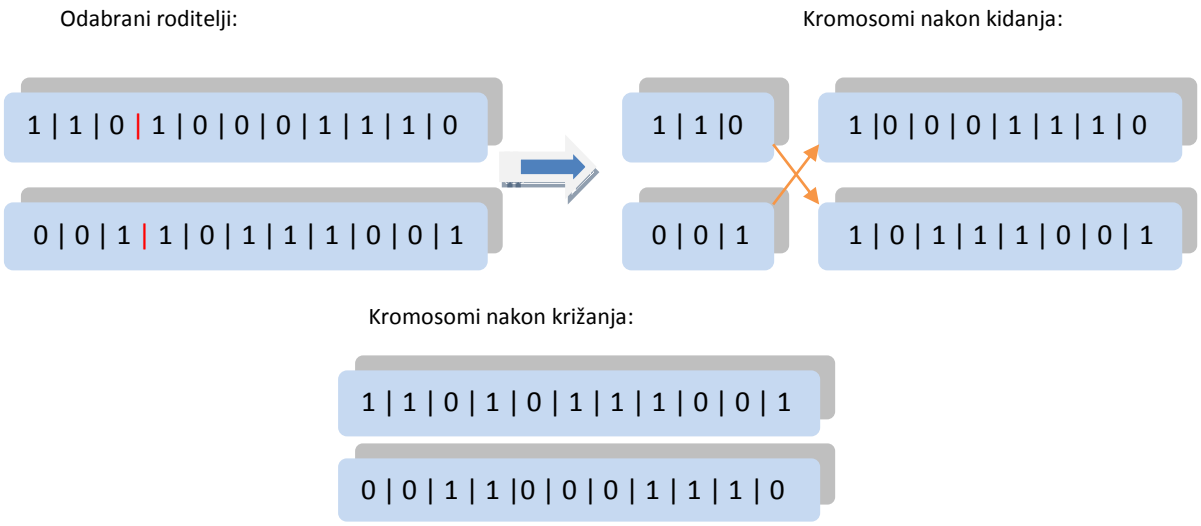
Kada promatramo optimizacijski problem prva odluka koju moramo donijeti jest kako prikazati rješenje. U evolucijskim algoritmima rješenja su prikazana genotipom, genomom ili kromosomima. Jednom kada se odlučimo za prikaz rješenja, ovisno o problemu kojeg rješavamo, procjena kandidata rješenja trebala bi biti moguća. Evolucijski algoritam radi s populacijom rješenja te iz toga proizilazi da je veličina populacije važan faktor evolucijskih algoritama. Parametar mutacije je operator inovacije koji osigurava da se istražuju ostali prostori rješenja te da se ne zaglavi u lokalno najboljim rješenjima. Operator križanja se rabi za jačanje već

naučenih svojstava od strane svih jedinki populacije. Veoma važan faktor evolucijskog algoritma je funkcija dobrote koja razlučuje bolja rješenja od ostalih. Selekcija radi na principu dobrote jedinke i odlučuje koje će jedinke nastaviti razmnožavanje a koje jedinke će biti zamijenjene.

2.3 Genetski algoritmi

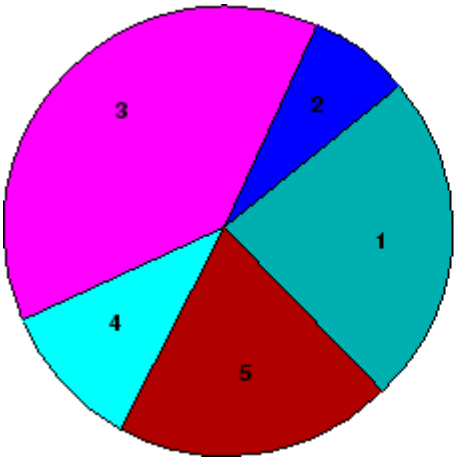
Jedan od najpoznatijih evolucijskih algoritama jest genetski algoritam (GA). Razvio ga je Holland, njegovi studenti i kolege na sveučilištu Michigan [Holland, 1992]. Originalni GA ima dvije glavne karakteristike: rabi binarnu reprezentaciju fiksne duljine i križanje.

Primjena operatora križanja nad dvjema roditeljskim jedinkama:



Slika 2.3: Križanje na binarnom kromosomu

Najobičniji oblik križanja je križanje u jednoj točki. Dva roditelja jednake duljine su poravnata te se odabire točka križanja. Repove jedinki zamijenimo od točke prekida pa nadalje. Tako dobijemo dva nova potomka. Selekcija je proporcionalna dobroti jedinke („*roulette wheel*“ selekcija, slika 2.4). Svaka jedinka dobije dio kotača proporcionalno njenoj dobroti u odnosu na prosječnu dobrotu ostalih jedinki u populaciji. Svaka jedinka ima šansu sudjelovati u stvaranju sljedeće generacije.



Population	Fitness
1	25.0
2	5.0
3	40.0
4	10.0
5	20.0

Slika 2.4: Roulette-wheel selection (slika preuzeta sa http://www.civil.iitb.ac.in/tvm/2701_dga/2701-ga-notes/gadoc/gadoc.html.)

2.4 Evolucijske strategije

Evolucijske strategije su još jedna paradigma evolucijskog računanja. Evolucijske strategije su optimizacijske tehnike bazirane na prilagođavanju i evoluciji. Njihova ideja je stvaranje jednog ili **λ** potomaka operacijom mutacije iz jednog ili μ roditelja. U sljedeću se generaciju prenosi ili μ roditelja i **λ** potomaka ili samo **λ** potomaka. Roditelji i potomci predstavljaju potencijalno rješenje optimizacijskog problema. Potencijalna rješenja prikazuju se pomoću vektora realnih brojeva. Broj na određenom mjestu unutar vektora opisuje neku karakteristiku samog rješenja. Kako bi se došlo do rješenja određenog problema koristi se operator mutacije na samim vektorima, a rjeđe se upotrebljava i operacija rekombinacije. U slijedecu generaciju odlaze samo najbolje jedinke koje se odabiru procesom selekcije.

2.5 Evolucijsko programiranje

Evolucijsko programiranje rabi operator mutacije i selekcije. Križanje, kao genetski operator, se ne koristi. Mutacija mijenja vrijednost pojedinim genima. Intenzitet mutacije odvija se prema Gaussovoj jediničnoj normalnoj razdiobi. Na temelju toga zaključuje se jesu li vjerojatnije male mutacije ili one snažnije.

Rad algoritma evolucijskog programiranja odvija se u sljedeća tri koraka:

1. Početna populacija se generira slučajnim odabirom te se evaluiraju jedinke. Veličina populacije ovisi o vrsti problema no ne postoji univerzalan način za određivanje najefikasnijeg broja jedinki.
2. Svaka jedinka se kopira u novu populaciju, gdje se mutira prema zadanoj distribuciji mutacije.
3. Stare jedinke i novi potomci se evaluiraju i prema tome odabiru za povratak u početnu populaciju kako bi ponovno prošli proces selekcije i mutacije.

2.6 Osnovna obilježja genetskog programa

Glavna obilježja koja dijele većina GP sustava su:

- Stohastičko donošenje odluka
- Programske strukture
- Genetski operatori
- Simulacija evolucije populacije selekcijom temeljenoj na dobroti

Stohastičko donošenje odluka

GP rabi pseudo-slučajne brojeve kako bi simulirao slučajnost prirodne evolucije. Kao rezultat, GP rabi stohastičke procese i vjerojatnosno donošenje odluka u nekoliko faza razvoja programa.

Programske strukture

GP tvori programe različitih veličina od osnovnih jedinica poznatih kao *funkcije* i *terminali*. Funkcije obavljaju operacije nad svojim ulazima, koji su ili terminali ili izlazi iz drugih funkcija. Stvaranje programa se radi na samom početku kada se inicijalizira populacija.

Genetski operatori

GP transformira početne programe u populaciji rabeći genetske operatore. Križanje dvije jedinke programa je jedan genetski operator. Drugi važni operatori su mutacija i reprodukcija.

Simulacija evolucije selekcijom

GP evoluira populaciju programa paralelno. Selekcija se vrši s obzirom na dobrotu jedinki. Selekcija bazirana na dobroti određuje koji će programi biti odabrani za daljnje napredovanje.

2.6.1 Terminali i funkcije

Terminali i funkcije su primitivi od kojih je izgrađen genetski program. Funkcije i terminali igraju različite uloge. Terminali predstavljaju vrijednosti dok funkcije obrađuju neku vrijednost koja se već nalazi u sustavu. Terminali i funkcije se nazivaju čvorovima.

Skup terminala

Ulazi, konstante i ostali nulti argumentni čvorovi se nazivaju terminalima ili listovima zato što završavaju granu stabla u GP baziranim na stablima. Skup konstanti se bira na početku evolucije genetskog programa. Konstante se ne mijenjaju tokom evolucije.

Skup funkcija

Skup funkcija se sastoji od naredbi u programu, operatora i funkcija koje su raspoložive GP sustavu. Skup funkcija ovisi o domeni problema koji se rješava. Neke od funkcija :

- *Boolean* funkcije: *and, or, not, xor*
- Aritmetičke funkcije: plus, minus, puta, podijeljeno
- Transcendentalne funkcije: trigonometrijske, logaritamske

  ...

2.6.2 Odabir skupa funkcija i skupa terminala

Funkcije i terminali koji se rabe za GP trebali bi biti dovoljno moćni za prikaz rješenja problema. Na primjer, skup funkcija koji se sastoji samo od operatora zbrajanja vjerojatno neće riješiti puno problema. S druge strane bolje je ne rabiti prevelik skup funkcija jer se proširuje prostor pretraživanja rješenja i može otežati pronalazak rješenja. Dobar skup funkcija bi bio na primjer:

plus, minus, puta, podijeljeno, or, and , xor.

Jedno važno svojstvo skupa funkcija jest kako bi svaka funkcija trebala biti zatvorena. Primjer funkcije koja nije zatvorena jest operator dijeljenja. Operator dijeljenja ne može prihvatiti nulu kao ulazni argument. Dijeljenje nulom će obično dovesti do rušenja sustava. Umjesto standardnog dijeljenja možemo rabiti takozvano zaštićeno dijeljenje. Zaštićeno dijeljenje je jednako kao normalno dijeljenje osim kad je dijeljenje s nulom. Tada funkcija vraća nešto drugo, npr.jako veliki broj. Sve funkcije moraju moći prihvaćati sve moguće argumente.

2.6.3 Izvršne programske strukture

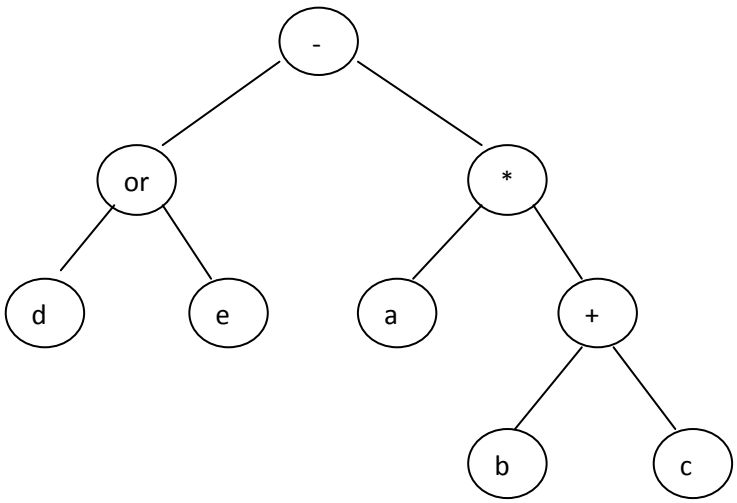
Primitivi GP-a, funkcije i terminal, nisu programi. Funkcije i terminali moraju biti sastavljeni u odgovarajuću strukturu prije nego se mogu izvršavati kao programi. Programi su strukture funkcija i terminala zajedno s pravilima i konvencijama kako i kada će se neki terminal ili funkcija izvršiti. Tri programske strukture koje se rabe u GP-u su stablo, linearne strukture i strukture grafa.

2.6.3.1 Struktura stabla

Izvršavanje stabla izvodi se tako da iznova evaluiramo najljeviji čvor za kojeg su dostupni svi argumenti. Ovakav redoslijed izvršavanja se zove postfiksni zato što operatori dolaze nakon operanada. Drugi način jest prefiksni koji je naravno suprotan od postfiksnog. Prednost prefiksnog redoslijeda jest npr.ako stablo sadrži čvorove kao *if/then* grane, tada možemo uštedjeti vrijeme izvođenja ako prvo evaluiramo treba li se izvršiti *then* grana.

Postfiksni redoslijed izvođenja čvorova za stablo na slici 2.5:

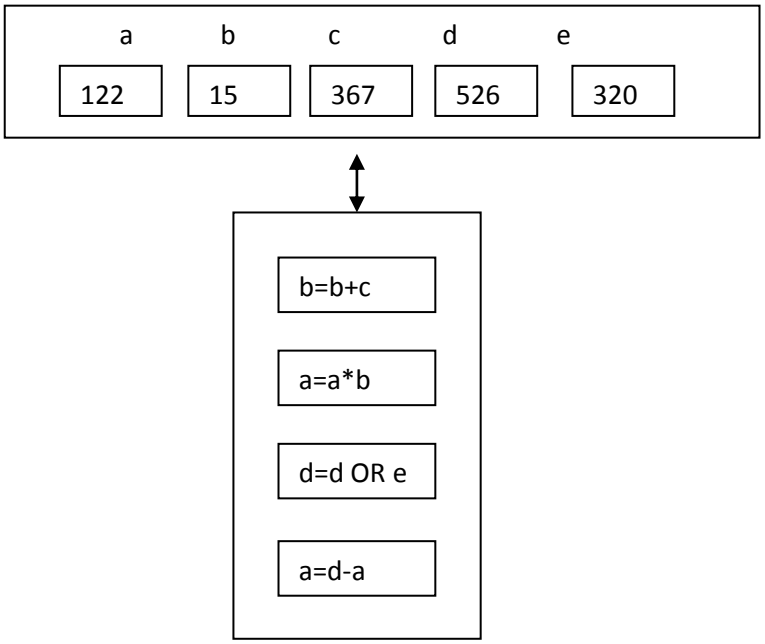
d->e->or->a->b->c->+->x->-



Slika 2.5: Prikaz strukture stabla

2.6.3.2 Linearna struktura

Linearni program na slici 2.6 je identičan kao program prikazan stablom na slici 2.5. Za linearni program je potrebna memorija u kojoj ćemo držati argumente za funkcije. Najčešće se rabe registri. Učinak funkcije prikazane na slici 2.6 je zbrojiti vrijednost registara b i c i staviti sumu u registar b. Linearni program kreće od gornje instrukcije i izvršava jednu po jednu instrukciju. Na kraju rezultat je pohranjen u registar a.



Slika 2.6: Prikaz linearnog programa

2.6.3.3 Struktura grafa

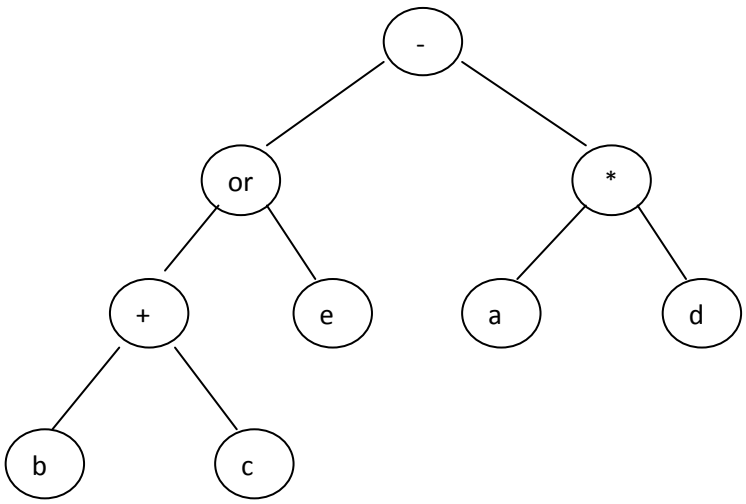
Pomoću grafova možemo prikazivati veoma složene strukture kompaktno. Struktura grafa su čvorovi povezani linijama. Možemo reći da linije između dva čvora određuju tok izvršavanja programa. Izvršavanje programa počinje čvorom start a završava čvorom kraj. Podaci između čvorova se prenose pomoću stoga.

2.6.4 Inicijalizacija GP populacije

Prvi korak u izvođenju genetskog programa je inicijalizacija populacije jedinki, odnosno stvaranje programskih struktura za evoluciju. Jedan od glavnih parametara je veličina programa. Kod stablastih struktura je to maksimalna dubina stabla ili maksimalni broj čvorova u stablu.

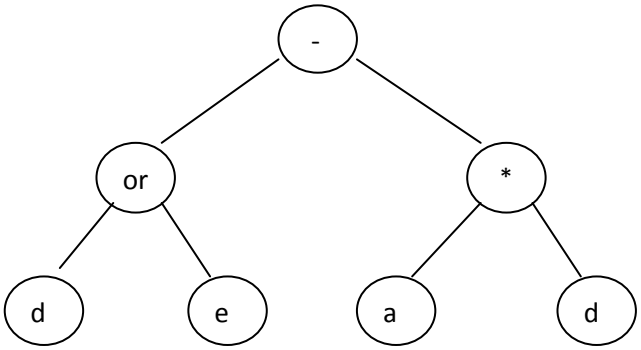
2.6.4.1 Inicijalizacija stablastih struktura

Postoje dvije metode za inicijalizaciju stablastih struktura, ***grow*** *i* ***full*** struktura. *Grow* struktura proizvodi stabla nepravilnog oblika jer se čvorovi nasumično biraju iz funkcijskog skupa i skupa terminala kroz cijelo stablo. Kada grana dobije terminalni čvor ta grana završava. Primjer stabla generiranog *grow* metodom je prikazan na slici 2.7.



Slika 2.7: Stablo generirano grow metodom

Umjesto nasumičnog biranja čvorova iz skupova terminala i funkcija, *full* metoda bira samo funkcije dok čvor ne dođe do maksimalne dubine, zatim bira samo terminale. Rezultat je takav da svaka grana stabla ide do maksimalne dubine. Ako se koristi maksimalni broj čvorova, rast staje kada se dosegne taj broj.



Slika 2.8: Stablo generirano full metodom

2.6.4.2 Ramped half and half metoda

Ova metoda se rabi kako bi se postigla raznovrsnost jer gore spomenute metode daju uniformi skup struktura početne populacije jer je postupak isti za sve jedinke. Pretpostavimo da je maksimalna dubina stabla 6. Populacija se podjeli jednako i inicijaliziraju se jedinke maksimalnih dubina 2, 3, 4, 5 i 6. Za svaku grupu pola jedinki grupe su inicijalizirane grow metodom a druga polovica full metodom.

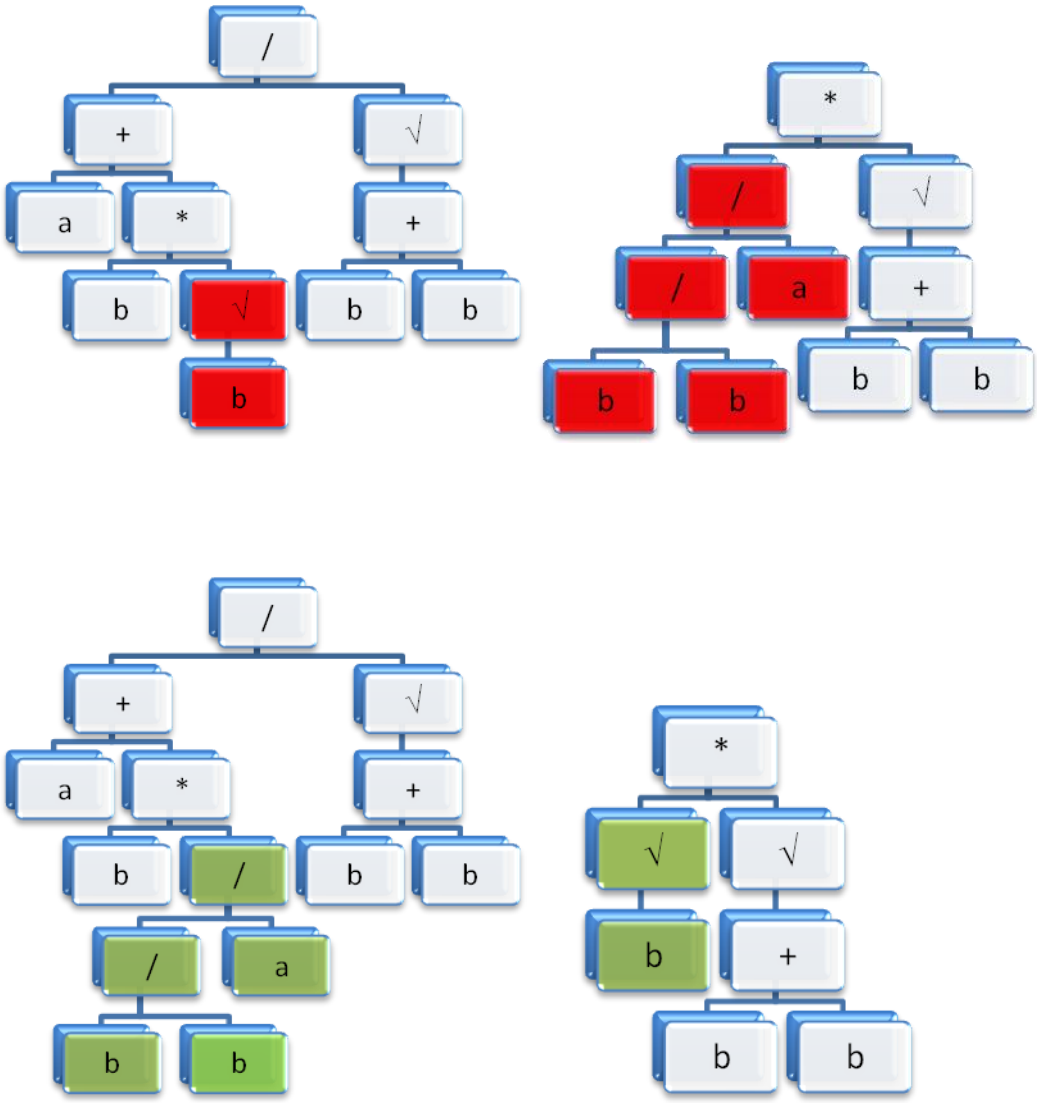
2.6.5 Genetski operatori

Početna populacija obično ima lošu dobrotu. Populacija evoluira tako da se nad jedinkama primjenjuju genetski operatori. Tri osnovna GP genetička operatora su:

- Križanje
- Mutacija
- Reprodukcija

2.6.5.1 Križanje

Operator križanja kombinira genetski materijal oba roditelja tako da dio jednog roditelja zamjeni djelom drugog roditelja. *Križanje na stablastim strukturama* je prikazano na slici 2.9.

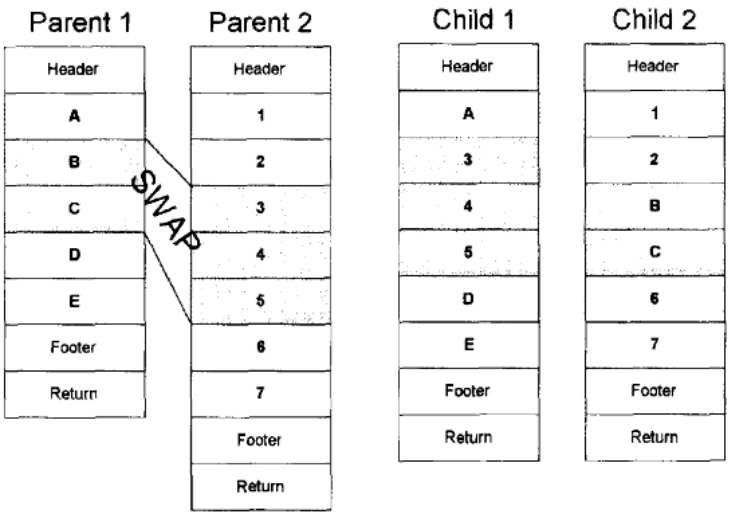


Slika 2.9: Prikaz križanja dvije jedinke

Koraci:

- Odaberi proizvoljno podstablo u svakom roditelju. Odabrana stabla su prikazana crvenom bojom.
- Zamjeni odabrana stabla u roditeljima.

Umjesto zamjene podstabala kod *linearnog križanja* zamjenjujemo segmente koda između dva roditelja. Linearno križanje je prikazano na slici 2.10.



Slika 2.10: Linearno križanje

(izvor: [Wolfgang Banzhaf] Genetic Programming An Introduction)

Koraci:

- Odaberi dvije jedinke roditelja.
- Odaberi nasumične sekvence instrukcija u oba roditelja.
- Zamjeni odabrane instrukcije između oba roditelja.

Križanje grafa je malo kompliciranije od prethodnih križanja. Sljedeću proceduru križanja grafa je predstavio Teller [Teller, 1996].

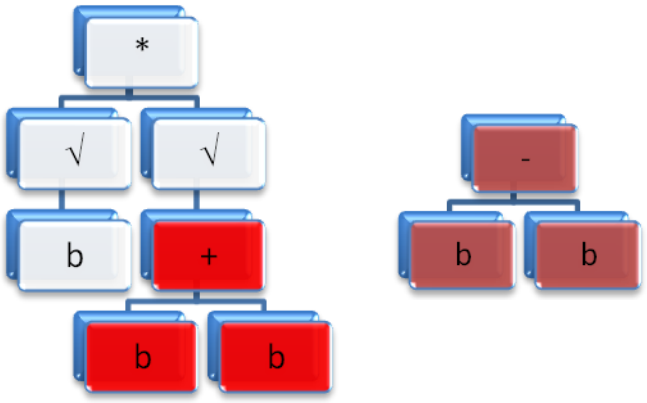
- Odaberi dvije jedinke roditelja
- Raspodjeli svaki graf u dva skupa čvorova
    - Označi sve rubove u fragmentu kao interne ako povezuju čvorove, inače označi kao eksterne
    - Označi čvorove u fragmentu kao izlazne ako su izvor eksternog ruba ili kao ulazne ako su odredište vanjskog ruba
- Zamjeni odabrane fragmente između roditelja
- Rekombiniraj rubove tako da svi eksterni rubovi u fragmentima pripadaju nasumično odabranim ulaznim čvorovima.

2.6.5.2 Mutacija

Nakon križanja svaki potomak je podvrgnut mutaciji. Vjerojatnost mutacije se određuje kao parametar genetskog programa.

Mutacija u stablastim strukturama (slika 2.11)

Kada je jedinka odabrana za mutaciju nasumično odabiremo čvor u stablu i cijelo njegovo podstablo zamijenimo slučajno generiranim stablom.



Slika 2.11: Mutacija u stablastim strukturama

Mutacija u linearnim strukturama

Kada je odabrana jedinka za mutaciju, operator mutacije prvo odabire jednu instrukciju te jedinke za mutaciju i na njoj napravi jednu ili nekoliko promjena.

Primjer r0 = r1 + r2.

Primjeri mutacije:

r1 = r1+r2 ili r0 = r2 + r2 ili r0 = r1 or r2 ili r0 = r1+r0

2.6.5.3 Reprodukcija

Reprodukcija je jednostavna. Odabere se jedinka i kopira u sljedeću generaciju u skladu s kriterijskom funkcijom. Kriterijiska funkcija se pridružuje svakoj jedinki u populaciji, pri čemu visoka vrijednost ove funkcije označava visoku kvalitetu. Kriterijska funkcija može biti bilo koja linearna, diferencijabilna ili nediferencijabilna funkcija. Reprodukcije se ostvaruje preko *„roulette wheel“* postupka.

Reprodukciju možemo formalizirati na sljedeći način:

1. Zbrojimo kriterijske funkcije svih članova populacije. Ovaj zbroj se naziva sveukupna performansa
2. Gererira se slučajan broj *n* iz intervala od nule do sveukupne performanse
3. Izabere se prvi član populacije čije je kriterijska vrijednost, zbrojena s kriterijskim vrijednostima svih prethodnih članova, veća ili jednaka broju *n*.

Tablica 2.1: Primjer reprodukcije

Redni broj	Kromosom	Vrijednost funkcije dobrote	Kumulativna vrijednost
1	01110	8	8
2	11000	15	23
3	00100	2	25
4	10010	5	30
5	01100	12	42
6	00011	8	50

Sveukupna performansa skupa prikazanog u tablici 2.1 je 50 te u cilju izbora roditelja koji će se reproducirati, generiramo brojeve iz intervala [0,50]. U tablici 2.2 je prikazan slučajan izbor 5 brojeva na temelju kojih se vrši izbor roditelja iz populacije.

Tablica 2.2: Izbor roditelja reprodukcijom

Slučajni broj	2	49	15	40	36
Izabrani kromosom	1	6	2	5	5

2.6.6 Funkcija dobrote

Cijeli proces evolucije najviše ovisi o dobroti jedinki, odnosno funkciji koja ju evaluira. Svaki individualni računalni program, svaka jedinka populacije, je izvršen i evaluiran, koristeći funkciju dobrote, kako bi odredili koliko se dobro on ponaša u danim uvjetima. Dobrota je obično određena nedosljednošću između rezultata dobivenog od jedinke i željenog rezultata. Što je pogreška manja, program je bolji.

Funkcija dobrote ne bi trebala nagrađivati samo najbolje jedinke (rješenja) u populaciji, već i sva poboljšanja pronađena tijekom evolucijskog procesa. Određivanje dobrote nekog rješenja sastoji se od ispitivanja ponašanja toga rješenja na nizu ispitnih primjera. Uočimo da je proces učenja vođen isključivo funkcijom dobrote, te sustav može iskoristiti i nedostatke toga skupa. Sustav se može specijalizirati za pojedini skup i proizvoditi jedinke koje su efikasne samo za taj skup a na ostalima ne daju prihvatljive rezultate.

2.6.7 Algoritam selekcije

Nakon što smo odredili kvalitetu jedinke pomoću funkcije dobrote, moramo odlučiti hoćemo li primijeniti genetske operatore na tu jedinku, zadržati je u populaciji ili ju zamijeniti. Selekcija je posljedica natjecanja između jedinki populacije. Postoje dva glavna scenarija selekcije: generacijska selekcija (GA) i eliminacijksa selekcija (ES).

2.6.7.1 GA scenarij

GA scenarij počinje s populacijom jedinki s poznatom dobrotom i na temelju te dobrote odabire jedinke. Odabrane jedinke su zatim podložene operatorima kao križanje i mutacija ili reprodukcija i proslijeđene u sljedeću generaciju. Sljedeća generacija obično sadrži jednak broj jedinki kao i prošla. Ponovno se računa dobrota jedinki kako bi se proces ponovio.

2.6.7.2 ES scenarij

Iz početne populacije se generira veći skup potomaka nasumično birajući roditelje. Nakon evaluacije dobrote populacija se smanjuje postupkom selekcije na veličinu početne populacije.

2.6.8 Parametri genetskog programa

Osnovni parametri genetskog programiranja su veličina populacije i maksimalni broj generacija. Osim ovih parametara proces genetskog programiranja uključuje i postavljanje vrijednosti mnogih drugih parametara. Odgovarajuće vrijednosti su

ovisne o samoj primjeni i ne postoji neki općeniti skup vrijednosti parametara koji je dobar za svaki problem. U genetskom programiranju, za ne trivijalne probleme, koriste se populacije od tisuća ili čak milijuna jedinki koje evoluiraju desecima, stotinama ili tisućama generacija.

2.6.9 Uvjet zaustavljanja

Svako pokretanje genetskog programa zahtjeva specificiranje uvjeta zaustavljanja kada ćemo prekinuti izvođenje i odrediti rezultat. Jedna od metoda za određivanje rezultata je određivanje najbolje jedinke iz bilo koje generacije populacije u tijeku izvođenja (*best-so-far individual*) kao rezultata izvođenja. Najčešći oblik uvjeta zaustavljanja jest dostizanje predefiniranog broja generacija evolucijskog procesa. U svom radu (Koza 1992) John R. Koza za sve pokuse koristi broj od 50 generacija kao uvjet zaustavljanja izvođenja postupka, ukoliko prethodno nije pronađeno rješenje problema. Nakon tog broja generacija ne pronalaze se bitno različita rješenja, odnosno poboljšanja su vrlo mala. Dobro je imati dvojaku provjeru: algoritam se zaustavlja nakon zadanog broja generacija ili se zaustavlja nakon što u određenom broju generacija nije nađeno bolje rješenje, a to znači kraj učinkovite pretrage.

2.6.10 Građa rješenja

Ako uzmemo prikaz rješenja pomoću stabla, građa rješenja se može sastojati od ADF funkcija (*automatically defined functions*). ADF funkcije su potprogrami, procedure ili subrutine koje dinamički evoluiraju tijekom izvođenja genetskog programa i mogu biti pozvane od strane glavnog programa koji također evoluira tijekom izvođenja genetskog programa. Koza u svom radu (Koza 1994) dokazuje da je pristup rješavanju pomoću ADF funkcija brži u pronalasku rješenja i bolji u rješavanju težih problema od običnog GP-a.

2.6.11 Osnovni GP algoritam

Postoje dva pristupa načinu izvođenja GP-a, generacijski pristup i eliminacijski pristup. U generacijskom GP-u, stvara se potpuno nova generacija iz stare generacije u jednom ciklusu. Nova generacija zamjenjuje staru i ciklus se nastavlja. U steady-state pristupu nema generacija.

Generacijski GP algoritam

1. Inicijaliziraj populaciju
2. Evaluiraj jedinke
3. Dok nova populacija nije popunjena ponavljaj:
    a. Odaberi jedinku ili jedinke iz populacije rabeci selekciju
    b. Primjeni genetske operacije nad odabranim jedinkama
    c. Rezultat genetskih operacija uvrsti u novu populaciju
4. Ako je zadovoljne uvjet zaustavljanja nastavi, inače zamjeni staru populaciju novom i ponavljaj korake od 2 do 4.
5. Kao izlaz algoritma je najbolja jedinka

Steady-state algoritam

1. Inicijaliziraj populaciju
2. Nasumično odaberi podskup populacije koji će sudjelovati u natjecanju
3. Evaluiraj dobrotu jedinki iz podskupa
4. Odaberi pobjednika/pobjednike rabeći selekciju
5. Primjeni genetske operatore nad pobjednikom/pobjednicima
6. Zamjeni gubitnike natjecanja s rezultatima genetskih operacija nad pobjednikom/pobjednicima
7. Ponavljaj korake od 2 do 7 dok nije zadovoljen uvjet zaustavljanja
8. Odaberi najbolju jedinku kao izlaz algoritma

3 Strojno učenje

Strojno učenje (engl. *machine learning*, ML) je proces koji počinje s identifikacijom domene i završava s ispitivanjem i korištenjem rezultata učenja. Osnovni dijelovi ovog procesa su *domena učenja, skup za učenje, sustav učenja i ispitivanje* rezultata učenja. Domena učenja je bilo kakav problem ili skup činjenica gdje možemo identificirati osobine domene koje se mogu mjeriti i rezultat koji želimo predvidjeti.

U genetskom programu osobina bi bila ulaz u sustav a rezultat izlaz. Kada odaberemo ulaze iz domene učenja, oni definiraju okruženje u kojem će sustav učiti. Sâmo strojno učenje odvija se treniranjem na skupu za učenje kojeg moramo definirati. U slučaju GP-a ovo znači da sustav mora naučiti računalni program da bude sposoban predvidjeti izlaze skupa za učenje na temelju ulaza u sustav. Nakon učenja odabiremo najbolje rješenje i ispitujemo ga na skupu za ispitivanje kako bi vidjeli koliko je rješenje dobro.

Kao što smo vidjeli GP sustavi rabe algoritam temeljen na prirodnoj evoluciji. Višeslojne neuronske mreže se temelje na biološkim živčanim sustavima, Bayes sustavi se temelje na statistici. Klasifikaciju algoritama za učenje napravit ćemo s obzirom na četiri osnovna pitanja [Wolfgang Banzhaf, Genetic Programming ,1998]:

1. Kako su prikazana rješenja?
2. Koje operatore pretraživanja koristi algoritam kako bi se kretao u prostoru rješenja?
3. Koja vrsta pretraživanja se koristi?
4. Jeli učenje nadgledano ili nenadgledano?

3.1 Prikaz problema

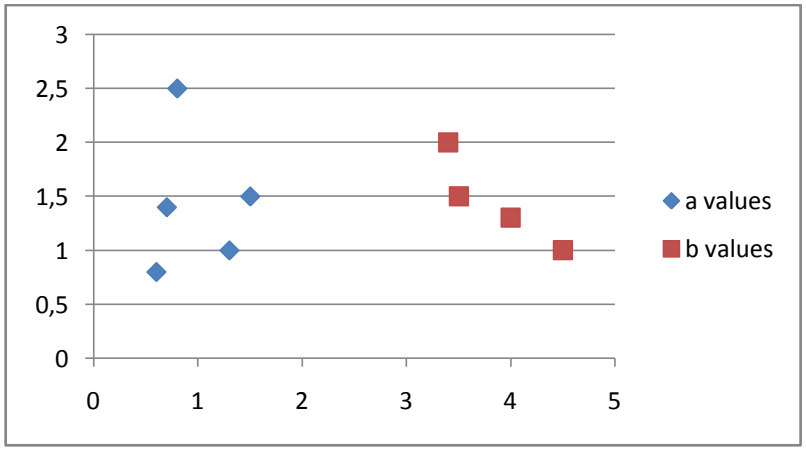
Prikaz problema u sustavu strojnog učenja (ML) definira kako izgledaju moguća rješenja, odnosno kakav tip ulaznih parametara sustav prihvaća te kako ulazi oblikuju izlaz sustava. Možemo reći da prikaz problema definira skup svih mogućih rješenja koje sustav može pronaći za dani problem. Jednadžba 3.1 prikazuje jednostavan primjer prikaza problema. Želimo predvidjeti vrijednost varijable y s obzirom na varijablu x. Y je izlaz a x je ulaz u sustav.

$$y = ax^2 + bx + c \tag{3.1}$$

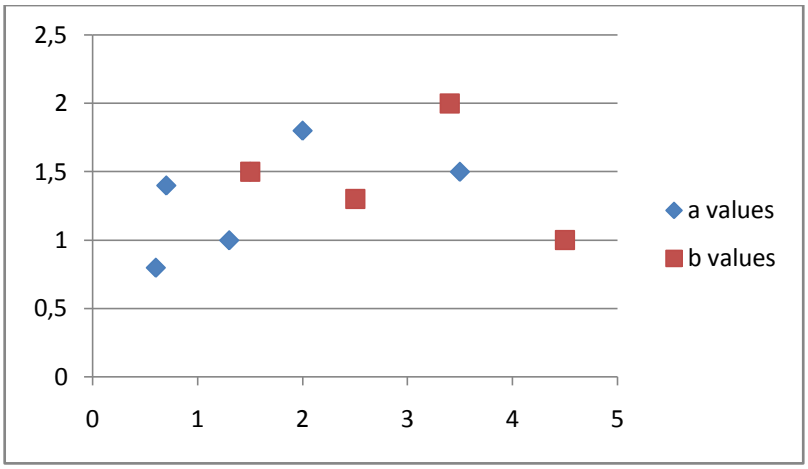
Prikaz problema definira i ograničava prostor pretraživanja rješenja koje sustav može pronaći. U navedenom primjeru skup za učenje bio parovi brojeva ulaza i izlaza, vrijednost ulaza *x* za vrijednost izlaza *y*.

3.1.1 Boolean prikaz

Svaki primjer za učenje u *boolean* prikazu za određeni ulaz ima definiran izlaz koji može biti istinit ili lažan. U pravom boolean sustavu ulazi su prikazani u boolean izrazima. Takav sustav opisuje koncepte koje je naučio preko Boolean konjukcija i disjunkcija danih ulaza u sustav. Konjuktivni i disjunktivni sustavi mogu potpuno opisati domenu koja linearno separabilna. Pod ovim mislimo da sustav može prikazati sva rješenja koja se mogu pronaći u nekoj domeni rješenja. Za domenu kažemo da je linearno separabilan, ako možemo povući pravac koji će razdijeliti sve klase rješenja. Dvije klase (razreda) rješenja koje su linearno separabilne prikazane su na slici 3.1. Na slici 3.2 je su prikazana rješenja koja nisu linearno separabilna.



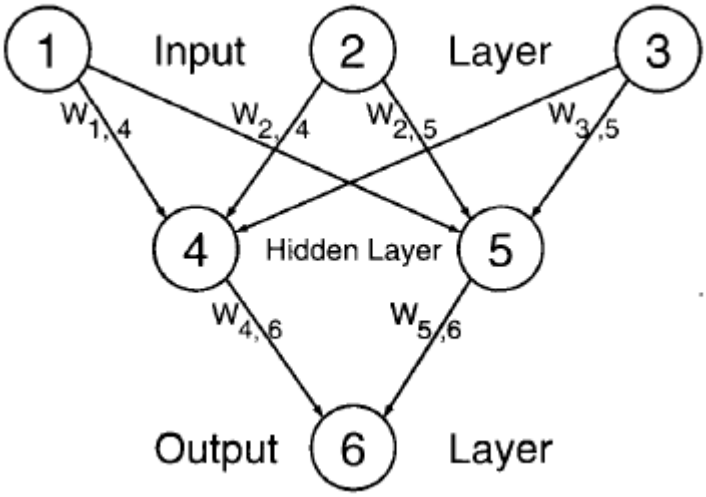
Slika 3.1: Linearno separabilne klase rješenja



Slika 3.2: Rješenja koja nisu linearno separabilna

3.1.2 Prikaz pragom

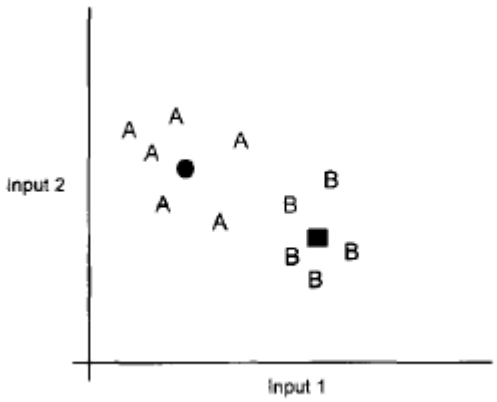
Ulaz ili jedinica u sustavu će proizvesti izlaz samo ako njeni ulazi prelaze nekakav prag vrijednosti (eng: *treshold representation*). Ako za izlaze kažemo da su boolean vrijednosti tada će izlaz jedinice biti true samo ako ulaz u tu jedinicu prelazi prag. Ovakav prikaz problema rabi se višeslojnim neuronskom mrežama. Neuronska mreža (slika 3.3) je sastavljena od jedinica koje zovemo neuroni. Svaki neuron zbraja sve svoje ulaze i zatim određuje prelazi li taj zbroj njegov prag. Ako ne prelazi izlaz iz neurona je 0, inače je neka pozitivna vrijednost veća od 0.



Slika 3.3: Neuronska mreža

3.1.3 Prikaz slučaja

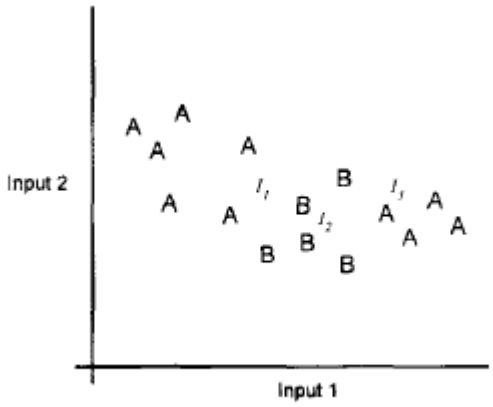
Sustav strojnog učenja koji koristi prikaz slučaja (eng: *case-based representation*) primjere za učenje predstavlja kao klase. Jednostavan primjer ovakvog sustava strojnog učenja za dvije klase, A i B, je prikazan na slici 3.4.



Slika 3.4: Prikaz slučaja za dvije klase

Svaka klasa ima dva ulaza, x i y vrijednost. Prosječna vrijednost ulaza A klase je krug, a prosječna vrijednost klase B je kvadrat. Kako bi klasificirao novi ulaz sustav izračuna koliko su vrijednosti novog ulaza blizu prosječnim vrijednostima klase A i B te ga svrstava u bližu klasu.

Ovakav sustav ne može ispravno klasificirati ako klase nisu linearno separabilne. Još jedan sustav baziran na prikazu slučaja koji može klasificirati klase koje nisu linearno separabilne je sustav koji koristi princip K najbližih susjeda. Uzmimo na primjer da sustav gleda 3 najbliža susjeda. Ako su 2 od 3 susjeda u klasi A, tada će sustav pridijeliti novi ulaz klasi A. Na slici 3.5 je prikazana klasifikacija tri nova ulaza I1, I2 i I3. Sustav ispravno klasificira primjere u A{I1,I3} i B{I2}.

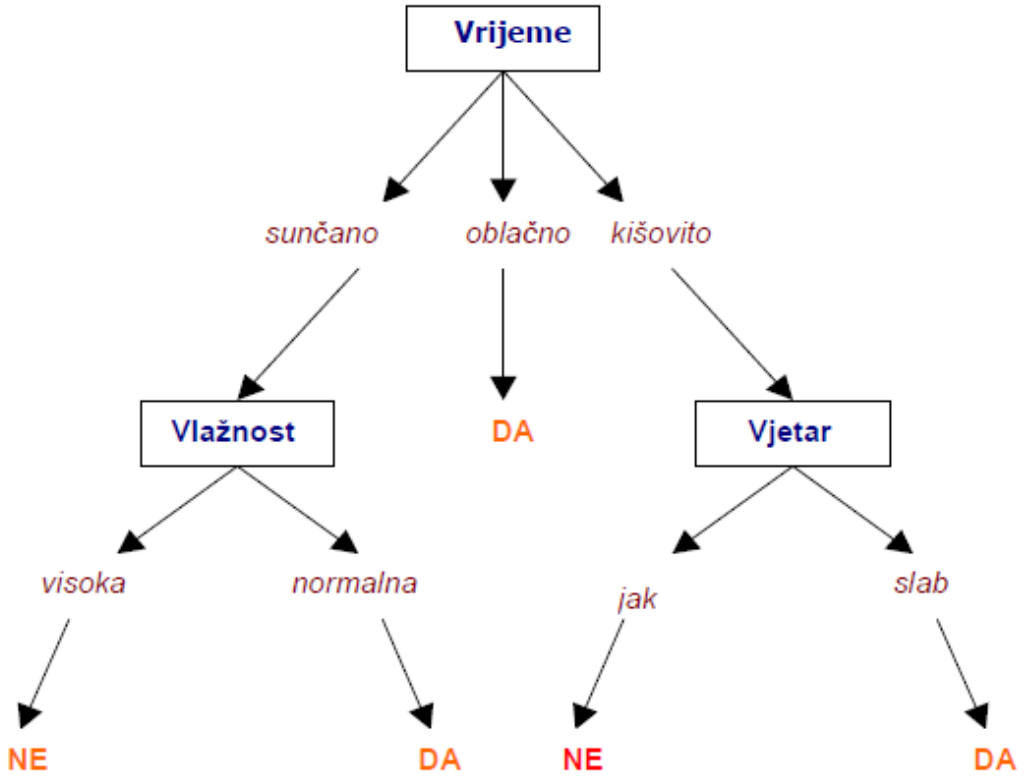


Slika 3.5: Klasifikacija po principu K najbližih susjeda

3.1.4 Prikaz stablom

Temeljni algoritam stabla odluke Quinlan [Quilan, 1979] je nazvao IDE3 (***I**nduction of **D**ecision **T**rees*) algoritam. Stablo odluke je tip usmjerenog grafa. Svaki unutarnji čvor u stablu je neko svojstvo domene. Svaki rub u grafu predstavlja moguće vrijednosti atributa čvora iznad. Svaki čvor list je klasifikacija.

Primjer klasifikacije **DA/NE –** „jeli subotnje jutro pogodno za tenis?“ je prikazan na slici 3.6.



(**Vrijeme** = sunčano, **Temperatura** = vruće, **Vlažnost** = visoka, **Vjetar**= jak)
(Klasifikacija, **Igranje_tenisa** = **NE**)

Slika 3.6: Klasifikacija stablom odluke

3.1.5 Genetski prikaz

Genetsko programiranje svoje koncepte i klasifikator predstavlja kao računalne programe. Svi sustavi strojnog učenja se mogu pokretati na računalu, odnosno mogu se prikazati kao računalni programi. GP može evoluirati rješenje pronađeno prethodno spomenutim prikazima.

- GP sustavi mogu uključivati boolean operatore
- Funkcija praga je obična IF/THEN funkcija
- Uvjetno grananje kao IF/THEN ili SWITCH omogućuje GP sustavu evoluciju stabala odluke
- GP sustav možemo implementirati na način da osiguramo memoriju u koju ćemo spremati atribute skupa za učenje te evoluirati prikaz slučaja.

Možemo reći da je GP prikaz nadskup svih ostalih prikaza sustava strojnog učenja i da možemo, ispravno definiranim GP sustavom, evoluirati i pronaći rješenje koje bi bilo koji drugi sustav strojnog učenja mogao pronaći.

3.2 Operatori i strategije pretraživanja

Operatori pretraživanja definiraju kako sustav strojnog učenja odabire rješenja koje će provjeravati te kojim redom. Operatori pretraživanja definiraju područje koje će biti pretraživano.

Vrste operatora:

- Generalizacija i specijalizacija (Boolean prikaz i prikaz praga)
- Gradijentni spust (neuronske mreže)
- Operatori GP-a, mutacija i križanje

Strategije pretraživanja opisuju kako se provodi pretraživanje prostora u kojem pronalazimo rješenje.

Neke vrste pretraživanja prostora rješenja:

- *Blind search*
- *Hill climbing*
- *Beam search*

3.2.1 Blind search pretraživanje

Blind search je pretraživanje prostora rješenja i odabir rješenja bez informacija o strukturi problema ili rezultata iz prijašnjih koraka pretrage. Blind pretraživanje se obično kreće kroz stablo prostora rješenja. U tom stablu svaki čvor predstavlja kandidata za rješenje a rubovi predstavljaju dopuštene skokove u prostoru pretraživanja. Postoje dvije vrste pretraživanja, pretraživanje u dubinu i širinu. Pretraživanje u širinu pretražuje stablo razinu po razinu dok ne pronađe dobro rješenje. Pretraživanje u dubinu ide prvo do najdubljeg čvora u stablu kroz jednu granu, zatim se vraća do sljedeće neprovjerene grane.

3.2.2 Hill climbing pretraživanje

Hill climbing pretraživanje kreće u jednoj točki prostora pretraživanja i zadržava novo rješenje ako je bolje od prethodno pronađenog. Ako novo rješenje nije bolje, odbacuje se i originalno rješenje se ponovno transformira i evaluira. Jedan od primjera uporabe hill climbing pretraživanja su neuronske mreže.

3.2.3 Beam search pretraživanje

U *beam search* pretraživanju rabimo evaluacijsku matricu za odabir nekoliko najboljih rješenja za daljnju transformaciju. Sva ostala rješenja odbacujemo. Rješenja koja odaberemo čine „*beam*“. Evaluacijska matrica je zapravo funkcija dobrote u GP sustavu, a „*beam*“ je populacija nad kojom GP traži rješenje.

3.3 Učenje

Postoje tri vrste učenja :

1. Nadgledano učenje (*eng: supervised learning*)
2. Nenadgledano učenje (*eng: unsupervised learning*)
3. Podržano učenje (*eng: reinforcement learning*)

Nadgledano učenje je učenje u kojem za svaki ulaz postoji odgovarajući izlaz. Izlaz ovako pronađenog rješenja se uspoređuje s točnim izlazom. U GP sustavima se često koristi ova tehnika gdje funkcija dobrote uspoređuje izlaz programa sa željenim izlazom.

Nenadgledano učenje je učenje u kojemu za svaki ulaz kod primjera za učenje nije definiran izlaz. Sustav samostalno traži uzorke u ulaznim podacima.

Podržano učenje je nešto između nadgledanog i nenadgledanog učenja. Iako nisu definirani izlazi za odgovarajuće ulaze, ipak se prenosi nekakva informacija o kvaliteti izlaza nazad u algoritam učenja.

3.4 Genetsko programiranje i strojno učenje

Genetski program možemo gledati kao jedan oblik strojnog učenja i opisujemo ga na sljedeći način:

- GP predstavlja problem kao skup svih mogućih računalnih programa ili podskup tog skupa. Za GP prikaz problema možemo reći da je nadskup svih ostalih prikaza u strojnom učenju.

- GP rabi križanje i mutaciju kao operatore transformacije kako bi promijenio trenutna potencijalna rješenja u nova potencijalna rješenja.

- GP rabi *beam search* gdje veličina populacije određuje koliko će biti veliki *beam*, a funkcija dobrote predstavlja evaluacijsku matricu.

- GP je obično implementiran kao oblik nadgledanog učenja, ali se može rabiti i podržano kao i nenadgledano učenje.

4 Učinkovitost GP-a u postupcima strojnog učenja

GP se može efikasno upotrijebiti za rješavanje problema strojnog učenja minimiziranjem pogreške između dobivenog i željenog rješenja za ograničeni skup primjeraka za ispitivanje, odnosno skupa za učenje.

U ranim fazama razvoja genetskog programiranja, problemi strojnog učenja rješavali su se na način da se rabio samo jedan skup podataka. Učenje i ispitivanje se obavljalo na istom skupu. Problemi su bili jednostavni pa nije niti bilo potrebe za složenijim pristupom. Za rješavanje složenijih problema ubrzo se uvidjelo da je ovaj pristup pogrešan jer može doći do pojave prenaučenosti. Prenaučenost je pojava kada se algoritam previše prilagodi na skup podataka na kojem uči te pokazuje slabije rezultate na ne viđenim skupovima podataka.

Važno je testirati algoritam, njegovu sposobnost generalizacije, na skupu podataka različitom od skupa za učenje.

Primijetimo da, iako koristimo dva skupa podataka, ne možemo spriječiti samu pojavu prenaučenosti. Uporabom dva skupa podataka možemo vidjeti koliko je rješenje dobro jer ispitivanje na istom skupu je pristrano u smislu da je razumljivo da ćemo dobiti dobre rezultate pošto se algoritam već specijalizirao za taj skup.

U postupku strojnog učenja uvodimo još jedan skup podataka, skup za validaciju. Ovaj skup služi da „kaže“ algoritmu kada bi trebao prestat učiti na skupu za učenje, odnosno kada se pojavila prenaučenost.

Što je bogatiji skup podataka za učenje to je veća šansa da se nađe bolje rješenje. Ovaj pristup ima jednu manu, a to je smanjenje dostupnog skupa primjera jer se skup sada dijeli, ne više na dva, nego na tri dijela.

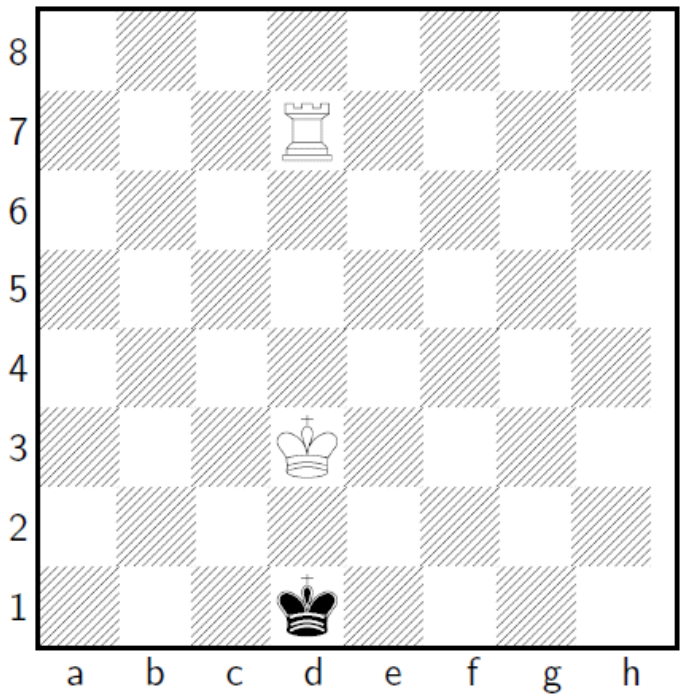
U ovom poglavlju prvo ćemo navesti primjer uporabe GP-a u strojnom učenju, a zatim detaljno opisati i prikazati pokuse i rezultate uporabe GP-a u rješavanju problema raspoređivanja.

4.1 Strojno učenje i GP na primjeru igre šaha

Igra šaha je problem aproksimacije, odnosno simulacije odabira najboljeg poteza iz velikog prostora pretraživanja. Čak niti stroj ne može igrati savršeni šah jer se procjenjuje da se stablo pretraživanja sastoji od 10^{120} čvorova. Računalima koja mogu evaluirati 10^{16} pozicija u sekundi bilo bi potrebno 10^{95} godina kako bi se potpuno evaluiralo stablo pretraživanja. Slijedi primjer KRK problema detaljno opisanog u radu *„Machine learning in Computer Chess: Genetic Programming and KRK“* ,[David Gleich 2003].

4.1.1 KRK problem

KRK (king-rook-king) problem je jedan od primjera završnica (engl. *endgame*) u šahu. Sastoji se od pozicija koje mogu zauzeti bijeli kralj, bijeli top i crni kralj. KRK problem je prikazan na slici 4.1.



Slika 4.1: KRK problem

Rješenje problema zahtjeva program koji uzima poziciju iz KRK baze podataka i vraća broj poteza do šah-mata. KRK baza podataka sastoji se od 28 056 pozicija čiji sažetak je prikazan u tablici 4.1. Podaci su preuzeti iz *University of California*

Irvine's Machine Learning baze podataka. U tablici 4.1 s iznimkom *draw* klase, ostale klase predstavljaju dubinu do šah-mata. Na primjer, imamo 2796 *draw*, odnosno neriješenih pozicija, i klasu od 1433 pozicije sa 8 poteza do šah-mata (klasa C_8 - klasa sa 8 poteza do šah-mata). Jednadžba (4.1) predstavlja KRK problem u obliku matematičke funkcije.

$$krk(wk_r, wk_f, wr_r, wr_f, bk_r, bk_f) = broj\ poeza\ do\ šah - mata \qquad (4.1)$$

U jednadžbi (4.1), wk_r predstavlja poziciju bijelog kralja, dok wk_f predstavlja polje. Isto tako pozicija bijelog topa je određena varijablama wr_r, wr_f, a pozicija crnog kralja je određena varijablama bk_r, bk_f.

Depth	Positions	Depth	Positions
draw	2796	8	1433
0	27	9	1712
1	78	10	1985
2	246	11	2854
3	81	12	3597
4	198	13	4194
5	471	14	4553
6	592	15	2166
7	683	16	390

Tablica 4.1: Sažetak svih pozicija u KRK problemu

4.1.1.1 Terminali i funkcije

Za KRK problem u „*Machine learning in Computer Chess: Genetic Programming and KRK*“ [David Gleich 2003], odabrani su sljedeći terminali i funkcije:

- Edge(i) – vraća 1 ako je i=8 ili i=1, inače vraća 2
- Distance(i,j) – vraća apsolutnu vrijednost od i-j (udaljenost između *i* i *j*)
- Ifthen(i,j,k) – ako je i=1 vraća j, inače vraća k

- Compare(i.j – vraća 1 ako je i<j, inače vraća 2

4.1.1.2 Operatori i parametri GP –a

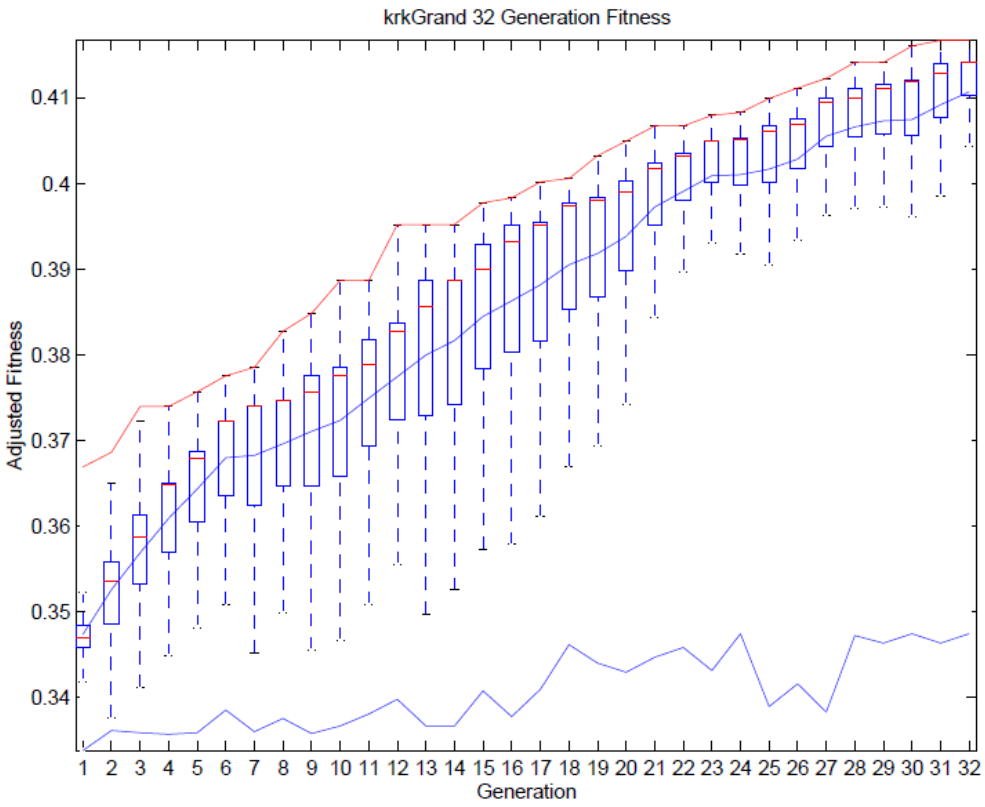
Prostor pretraživanja GP –a je veličine 5000 jedinki. Učestalost križanja 75%, a mutacije 15% i reprodukcije 10%. Jedinke su evaluirane nad svim podacima iz baze.

Standardna dobrota KRK problema je prikazana jednadžbom (4.2). U standardnoj dobroti najbolja jedinka ima vrjiednost 0.0. U jednadžbi (4.2) $correct_i$ predstavlja broj točnih odgovora u klasi C_i; $incorrect_i$ je broj netočnih odgovora u klasi C_i, a num_i je ukupan broj pozicija u klasi C_i.

$$krkFit = 1 - \frac{\sum_{i=0}^{16} \frac{correct\ _i - incorrect\ _i}{num\ _i}}{17} \qquad (4.2)$$

4.1.1.3 Rezultati

Na slici 4.2 su prikazani rezulati provedenih pokusa [*David Gleich„Machine learning in Computer Chess: Genetic Programming and KRK“ ,2003*.].



Slika 4.2: Rezultati GP-a u rješavanju KRK problema

Na slici 4.2 prikazana je dobrota jedinki u 32 generacije. Crvena linija predstavlja maksimalnu dobrotu u svakoj generaciji. Srednja plava linija je prosječna dobrota a donja plava linija je minimalna dobrota. Jedinka s najboljom dobrotom ima vrijednost 1.0.

Tablica 4.2 prikazuje broj pozicija u svakoj C_i klasi koje je najbolja jedinka, nakon 32 generacije, ispravno klasificirala. Pod stupcem *„Learned“* prikazano je koliko je pozicija najbolja jedinka naučila od 1. do 32. generacije.

Class	Correct	Learned	Class	Correct	Learned
C_0	27/27	+10	C_9	0/1712	0
C_1	59/78	-1	C_{10}	0/1985	0
C_2	175/246	+131	C_{11}	264/2854	+264
C_3	32/81	+21	C_{12}	1053/3597	+1053
C_4	58/198	+51	C_{13}	2003/4194	+2003
C_5	12/471	0	C_{14}	2812/4553	+2812
C_6	162/592	+66	C_{15}	281/2166	+281
C_7	10/683	-89	C_{16}	0/390	0
C_8	29/1433	-331			

Tablica 4.2: Kvaliteta najbolje jedinke GP-a na KRK problemu

4.2 Problem raspoređivanja

Pod pojmom raspoređivanje podrazumijevamo određivanje redoslijeda aktivnosti nad sredstvima koja su potrebna za njihovo izvršavanje. Ovdje razmatramo okruženje u kojem se vrši raspoređivanje aktivnosti nad samo jednim sredstvom.

.

4.2.1 Svojstva poslova

Kod raspoređivanja na jednom stroju, smatramo da se posao sastoji od samo jedne nedjeljive aktivnosti. Skup svih poslova možemo označiti sa J, a pojedini posao sa J_j.

Trajanje izvođenja (p_j) – vrijeme koje je potrebno da se izvrši aktivnost j od posla J_j .

Vrijeme željenog završetka (d_j) – (eng. *due date*) trenutak do kojeg se očekuje da će posao završiti. Posao može završiti i nakon isteka ovoga roka, ali se u tome slučaju stvara određeni trošak (npr. zbog kašnjenja isporuke naručitelju).

Težina (w_j) – težina (eng. *weight*) određuje prioritet posla u sustavu. Koristi se kod vrednovanja rasporeda, gdje predstavlja mjeru stvarne kvalitete rasporeda.

4.2.3 Ocjena rasporeda

Ocjena dobrote jedinki, a ujedno i učinkovitost pravila raspoređivanja obavlja se ovisno o okruženju i definiranom mjerilu vrednovanja rasporeda na većem broju ispitnih primjera. Kriteriji vrednovanja rasporeda u velikoj mjeri utječu na izbor odgovarajućeg algoritma raspoređivanja. Kao kriterij vrednovanja će se koristi težinsko zaostajanje.

C_j – vrijeme završetka (eng. completion time), trenutak u kojemu aktivnost j završava svoje izvođenje.

L_j – kašnjenje (eng. lateness), razlika između vremena završetka i vremena željenog završetka:

$$L_j = C_j - d_j$$

T_j - zaostajanje (eng. *tardiness*) – pozitivni iznos kašnjenja neke aktivnosti. Ako je kašnjenje negativno, zaostajanje je jednako nuli.

$$T_j = \max(0, L_j)$$

Težinsko zaostajanje (T_w) – težinsko zaostajanje (eng. *weighted tardiness*) jednako je težinskoj sumi zaostajanja svih poslova:

$$T\mathbf{w} = \sum_j w\mathbf{j}T\mathbf{j}$$

4.2.4 Raspoređivanje pomoću GP-a

Glavni dio našeg programa jest ECF (eng. *evolutionary computing framework* http://gp.zemris.fer.hr/ecf/). ECF je razvojno okruženje namijenjeno za bilo koji tip evolucijskog računanja. Od nekoliko podržanih algoritama mi ćemo koristiti *steady state tournament* algoritam koji je turnirski algoritam. Dakle, odabiremo *k* jedinki iz populacije, izračunamo dobrotu svake od njih, zatim onu jedinku sa najmanjiom dobrotom izbacujemo, a njeno mjesto zauzima nova jedinka nastala križanjem dviju jedinki koje su preživjele turnir. Osnovni dio ECF-a je *genotip*. Svaka jedinka u populaciji može imati jedan ili više genotipa, a najjednostavniji način uporabe jest navesti ih u konfiguracijskoj datoteci koju predajemo programu. Komponenta *state* se brine za sve ostalo: veličinu populacije, uvjet zaustavljanja itd. Genotip, o kojem ćemo malo više reći i kojeg koristimo u ovom radu je *tree.*

```
int main(int argc, char **argv)
{
 StateP state = static_cast<StateP> (new State);
 state->setEvalOp(static_cast<EvaluateOpP> (new MyEvalOperator));

 state->initialize(argc, argv);
 state->run();
 return 0;
}
```

Slika 4.3: Izgled glavnog programa u ECF-u.

Na slici 4.3 možemo vidjeti kako izgleda glavni program ECF-a. Definiramo novu komponentu *state* kojoj predajemo svoju evaluacijsku funkciju, funkciju koja će izračunavati dobrotu. Genotip, algoritam, veličinu populacije, broj generacija i ostalo definiramo preko konfiguracijske datoteke.

4.2.4.2 Genotip tree

Prikaz rješenja raspoređivanja po pravilima za statičke uvjete je stablo koje predstavlja funkciju prioriteta.

Križanje i mutacija stabla objašnjeni su u prethodnim poglavljima, stoga ćemo samo spomenuti princip. Križanjem uzimamo dvije roditeljske jedinke i slučajno odaberemo neko podstablo u svakoj jedinki te ih međusobno zamijenimo. Mutacija

se provodi na način da u nekoj jedinki odaberemo podstablo koje će biti zamijenjeno novim nasumično generiranim stablom i provedemo zamjenu.

Također je spomenuto da stablo sadrži podatkovne i funkcijske čvorove. Nazive tih čvorova moramo definirati kako bi stablo znalo kako raditi s njima. Preko konfiguracijske datoteke, pod tipom genotipa koji koristimo, definiramo maksimalnu i minimalnu veličinu stabla. Razlog zašto uvodimo ova ograničenja je čisto optimizacijski, jer premala stabla vjerojatno nisu dovoljno dobra da riješe problem, a prevelika opet kombiniraju velik broj računskih operacija i tako postižemo nepotrebnu složenost. ECF podržava razne matematičke operacije koje mogu poslužiti kao funkcijski čvorovi, a to su: +, -, *, /, cos, sin i slično. Odabir elemenata u skupu čvorova načinjen je ručno za svako pojedino okruženje. Nastojmo odabrati one elemente koji su značajni za specifično okruženje i uvjete raspoređivanja. Relevantni podaci koji bi u svakako trebali biti uključeni su trajanje poslova, težina i željena vremena završetaka. U tablici 3.1 možemo vidjeti popis čvorova za statički problem raspoređivanja na jednom stroju.

Tablica 4.1: Popis čvorova za statički problem na jednom stroju

Oznaka funkcijskog čvora	definicija
ADD	binarni operator zbrajanja
SUB	bnarni operator oduzimanja
MUL	binarni operator množenja
DIV	zaštićeno dijeljenje: $DIV\ (a,b) = \begin{cases} 1, ako \lvert b \rvert < 0.000001 \\ \frac{a}{b}, inače \end{cases}$
POS	unarni operator '+': POS(a)=max{a, 0}
Oznaka podatkovnog čvora	definicija vrijednosti podatkovnog čvora
pt	trajanje obrade (p_j)
dd	željeno vrijeme završetka (p_j)
w	težina (w_j)
SL	pozitivna odgoda, max{$d_j - p_j$ – time, 0}

4.2.4.3 Evaluacija i funkcija dobrote

Funkcija dobrote, iliti fitness funkcija, određena je težinskim zaostajanjem nad skupom primjera. Primjeri za učenje, kao i primjeri za ocjenjivanje, učitavaju se iz vanjskih datoteka (primjeri su preuzeti iz OR-library, http://people.brunel.ac.uk/~mastjjb/jeb/info.html). Dane su tri datoteke, i u svakoj je određeno jedno svojstvo svakog posla u skupu za učenje, ocjenu. Prva datoteka sadrži trajanja poslova. Trajanje svakog posla je vrijednost u intervalu `[1..100]`. Druga datoteka sadrži težinu poslova. Vrijednosti težine se kreću u intervalu `[1..10]`. Treća datoteka sadrži vrijeme željenog završetka poslova (eng. *due date*). Vrijednosti završetka se kreću u intervalu `[P(1-TF-RDD/2), P(1-TF+RDD/2)]`.

- RDD predstavlja relativni interval vremena željenog završetka (RDD=0.2, 0.4, 0.6, 0.8, 1.0).
- TF je prosječni faktor zakašnjelosti (TF=0.2, 0.4, 0.6, 0.8, 1.0).
- P je definiran kao zbroj trajanja svih poslova: P=SUM{ j=1, ...,n}p(j).

Prioritet svakog posla se određuje na temelju tih parametara, prema funkciji određenoj stablom koje predstavlja jedinku za koju se izračunava funkcija dobrote. Evolucijski proces se odvija turnirskim odabirom, gdje najlošiju jedinku eliminiramo. Veličina turnira je 3. U sljedećem poglavlju navesti ćemo načine testiranja te prikazati dobivene rezultate.

4.2.5 Rezultati

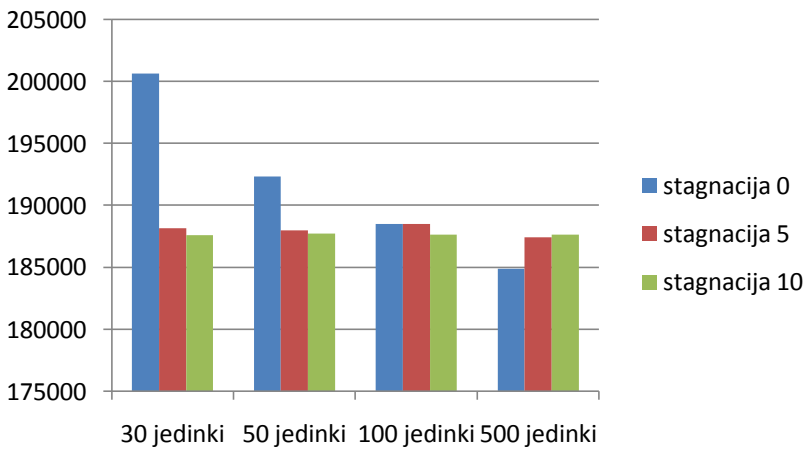
U tablici 4.2 opisano je okruženje ispitivanja. Veličina skupa je broj različitih rasporeda poslova na nekom stroju. U postupku učenja određen postotak najboljih jedinki provjerava se, osim na skupu za učenje, i na skupu za validaciju. Za svaki postotak najboljih jedinki koje se ispituju na skupu za validaciju, zbrajaju se njihove vrijednosti dobivene nad tim skupom te se tako dobivena vrijednost normalizira dijeljenjem s brojem jedinki koje sudjeluju u ispitivanju nad skupom za validaciju. Ako se tako dobivena vrijednost nad skupom za validaciju ne poboljša u određenom broju generacija, dolazi do pojave stagnacije te zaustavljamo učenje. U našim pokusima, ovisno o slučaju, dopuštamo stagnaciju od 0, 5 ili 10 generacija kako je prikazano u tablici 4.2.

Tablica 4.2: Opis okruženja za učenje

Broj jedinki	30, 50, 100 i 500
Veličina skupa za učenje	80
Veličina skupa za validiranje	20
Veličina skupa za ispitivanje	25
Minimalna veličina stabla	2
Maksimalna veličina stabla	6
Vjerojatnost mutacije	0.3
Stagnacija	0, 5, 10 generacija
Postotak najboljih jedinki za ispitivanje na skupu za validaciju	5%, 10%, 15%

4.2.5.1 Validacija

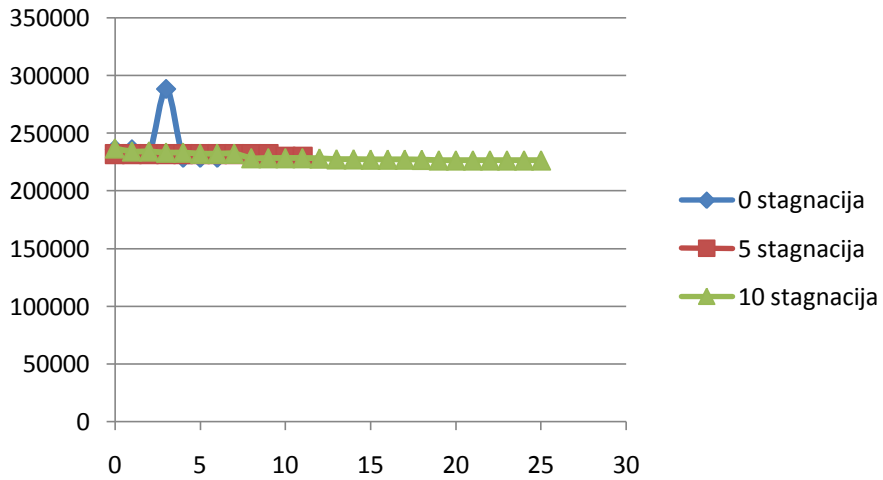
Na slici 4.4 prikazani su rezultati na skupu za ispitivanje. Tijekom učenja uzimali smo 5% najboljih jedinki za ispitivanje na skupu za validaciju.



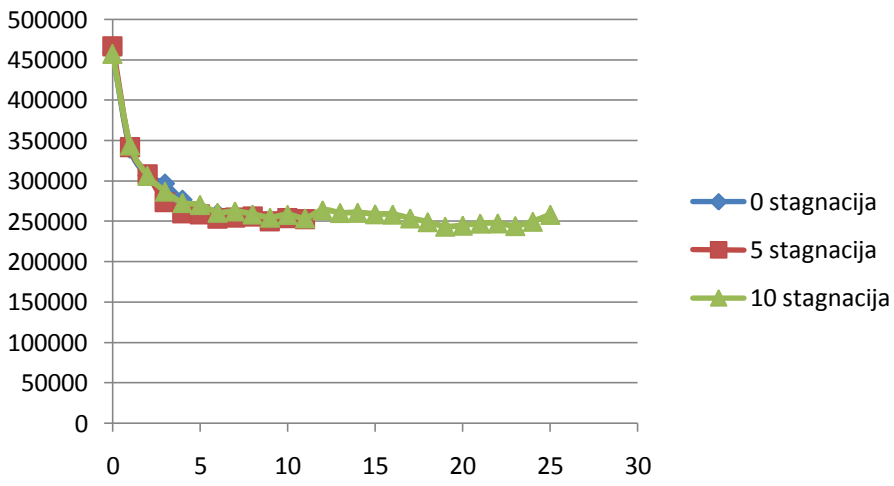
Slika 4.4: Rezultati na skupu za ispitivanje.
Za validiranje je odabrano 5% najboljih jedinki iz skupa za učenje

Najbolje rješenje dobiveno je za populaciju od 500 jedinki i stagnaciju 0 generacija.

Najbolja rješenja dobivamo za 500 jedinki. Na slici 4.5 je prikaz minimalne, a na slici 4.6 prosječne dobrote, kroz generacije za sva tri slučaja stagnacije.

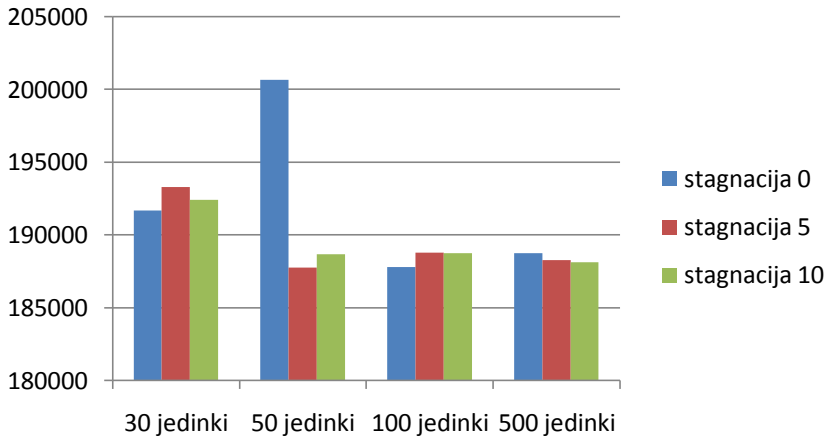


Slika 4.5: Prikaz minimalne dobrote kroz generacije za sva tri slučaja stagnacije



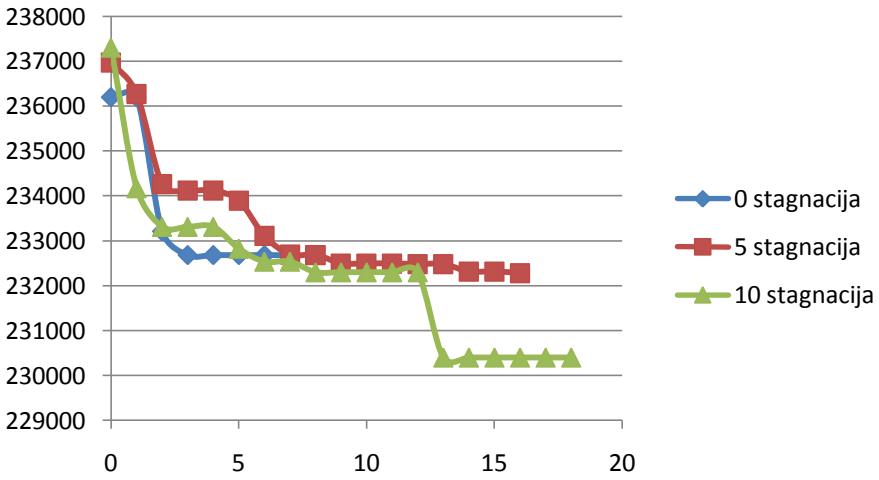
Slika 4.6: Prikaz prosječne dobrote kroz generacije za sva tri slučaja stagnacije

Na slici 4.7 prikazani su rezultati na skupu za ispitivanje. Tijekom učenja uzimali smo 10% najboljih jedinki za ispitivanje na skupu za validaciju.

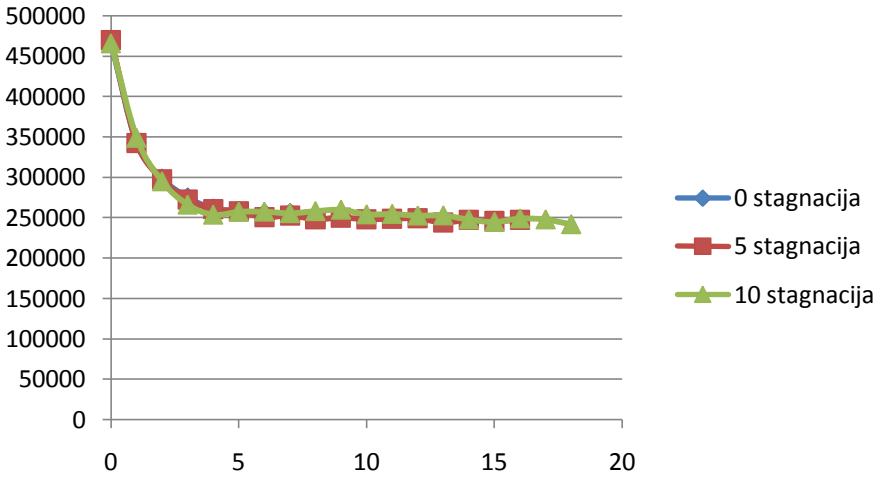


Slika 4.7: Rezultati na skupu za ispitivanje.
Za validiranje je odabrano 10% najboljih jedinki iz skupa za učenje

Najbolje rješenje dobiveno je za populaciju od 100 jedinki i stagnaciju 0 generacija. U prosjeku najbolja rješenja dobivamo za 500 jedinki. Na slici 4.8 je prikaz minimalne, a na slici 4.9 prosječne dobrote, kroz generacije za sva tri slučaja stagnacije.

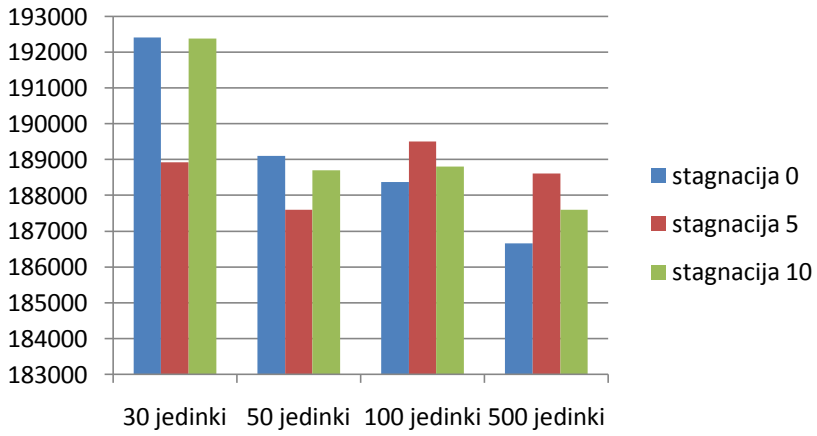


Slika 4.8: Prikaz minimalne dobrote kroz generacije za sva tri slučaja stagnacije



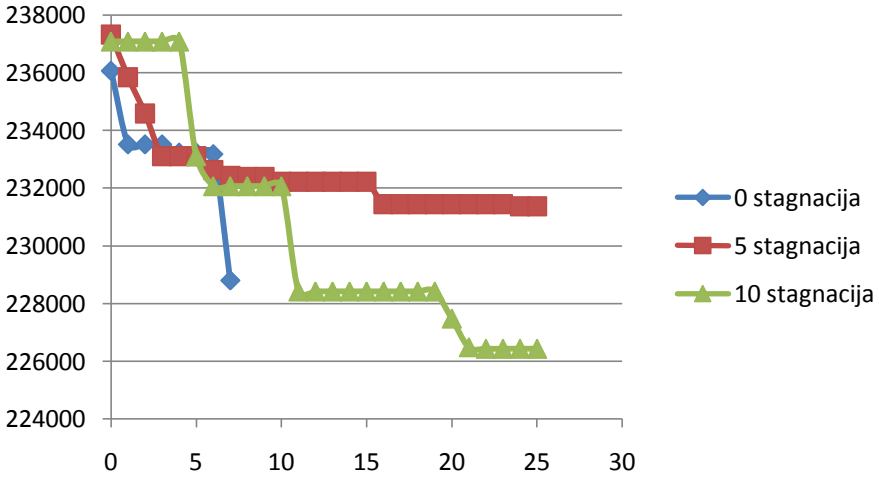
Slika 4.9: Prikaz prosječne dobrote kroz generacije za sva tri slučaja stagnacije

Na slici 4.10 prikazani su rezultati na skupu za ispitivanje. Tijekom učenja uzimali smo 15% najboljih jedinki za ispitivanje na skupu za validaciju.

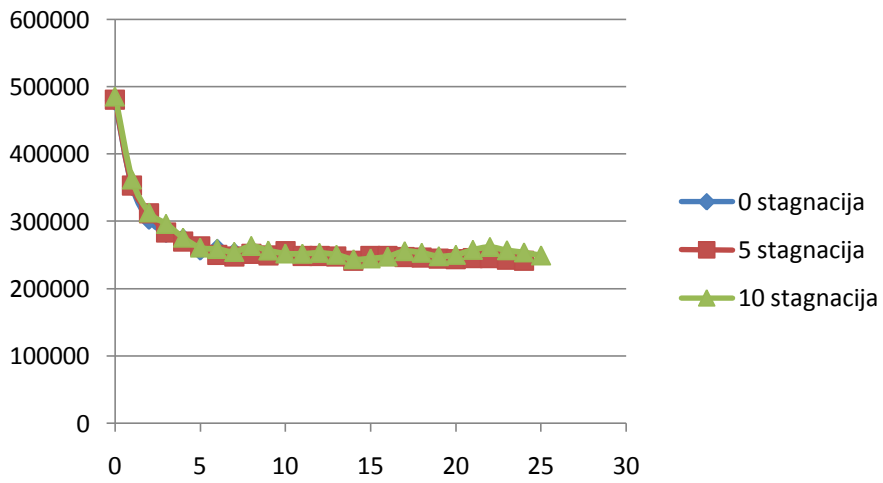


Slika 4.10: Rezultati na skupu za ispitivanje.
Za validiranje je odabrano 15% najboljih jedinki iz skupa za učenje

Najbolje rješenje dobiveno je za populaciju od 500 jedinki i stagnaciju 0 generacija. Na slici 4.11 je prikaz minimalne, a na slici 4.12 prosječne dobrote, kroz generacije za sva tri slučaja stagnacije.

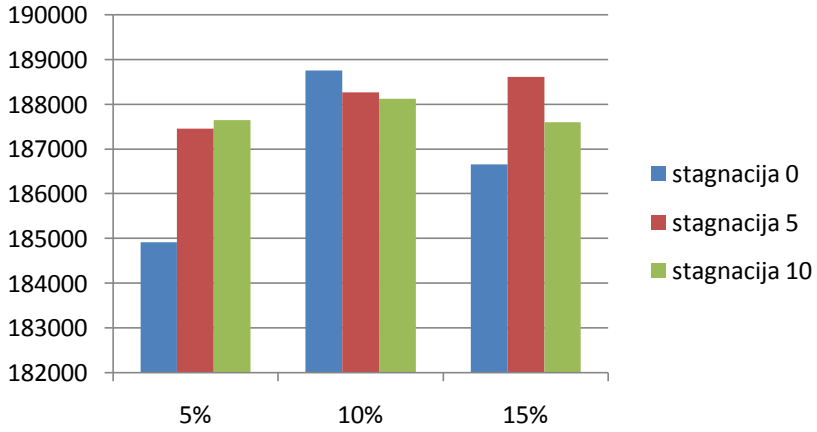


Slika 4.11: Prikaz minimalne dobrote kroz generacije za sva tri slučaja stagnacije



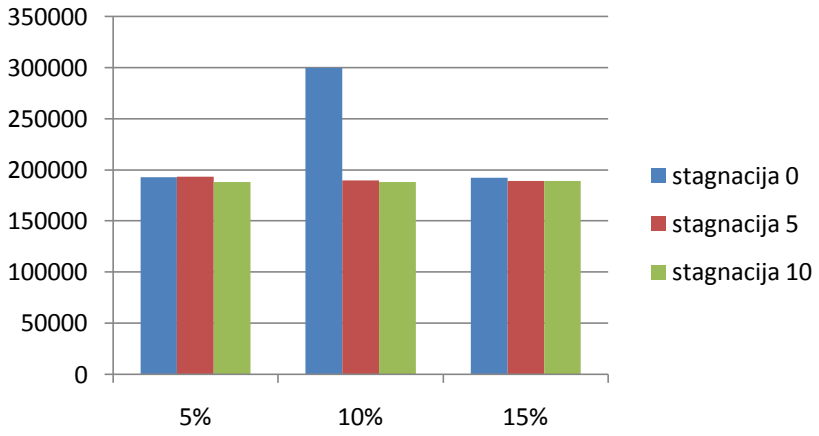
Slika 4.12: Prikaz prosječne dobrote kroz generacije za sva tri slučaja stagnacije

Na slici 4.13 uspoređeni su rezultati promatranja 5%, 10% i 15% najboljih jedinki za ispitivanje na skupu za validaciju. Veličina populacije je 500, a stagnacija 0, 5 i 10 generacija.



Slika 4.13: Usporedba rezultata korištenjem validacije uz promatranje 5% 10% i 15% najboljih jedinki iz skupa za učenje. Veličina populacije je 500.

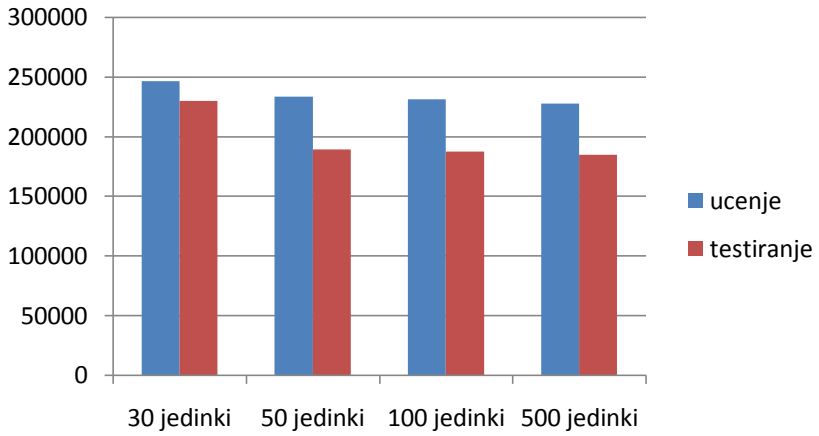
Na slici 4.14 uspoređeni su rezultati promatranja 5%, 10% i 15% najboljih jedinki za ispitivanje na skupu za validaciju. Veličina populacije je 500, a stagnacija 0, 5 i 10 generacija. Na skupu za ispitivanje ocjenjuje se samo jedna jedinka – ona koja ima najbolju ocjenu na skupu za validaciju. Ovu jedinku bi u stvarnim uvjetima uzeli kao konačno rješenje.



Slika 4.14: Usporedba rezultata korištenjem validacije uz promatranje 5% 10% i 15% najboljih jedinki iz skupa za učenje. Na skupu za ispitivanje ispitivali smo najbolju jedinku iz skupa za validaciju. Veličina populacije je 500.

4.2.5.2 Stagnacija

Na slici 4.15 prikazani su rezultati na skupu za ispitivanje. U ovome slučaju nismo rabili validaciju već je zaustavljanje učenja određeno stagnacijom populacije, odnosno brojem generacija bez promjene, pronalaska boljeg rješenja. Stagnacija je postavljena na 5 generacija.

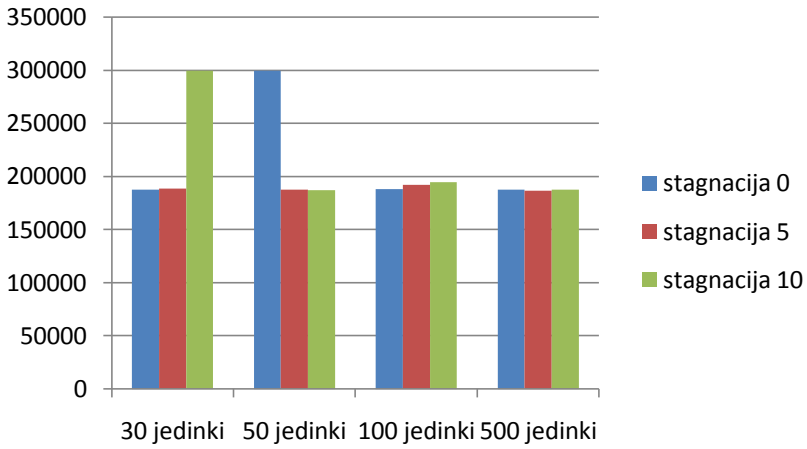


Slika 4.15: Rezultati uz uporabu stagnacije kao kriterija zaustavljanja učenja. Prikazani su rezultati na skupu za učenje kao i na skupu za ispitivanje.

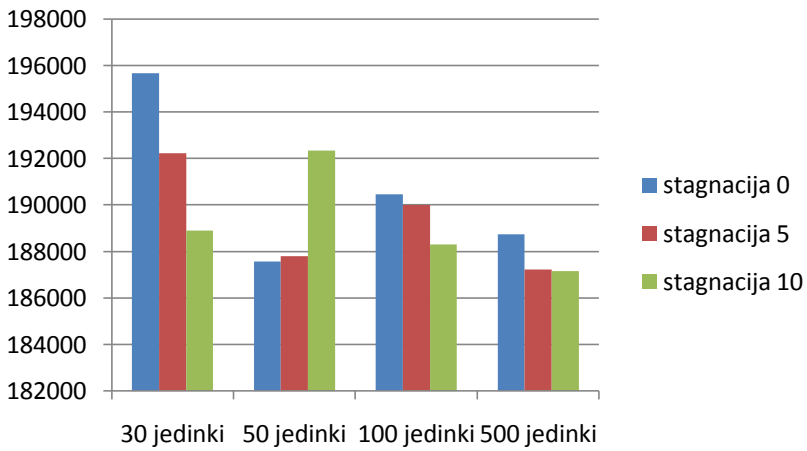
4.2.5.3 K-Fold Kros-validacija

K-Fold Kros-validacija je postupak sličan običnoj validaciji. Skup za učenje se dijeli na K dijelova. Za svaki od K pokusa, rabimo K-1 dijelova za učenje, a preostali dio

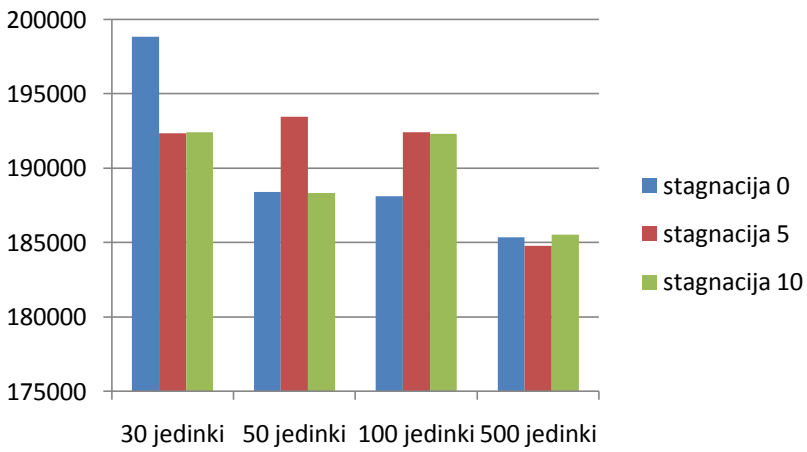
za validaciju. Za potrebe pokusa uzeli smo da je K=5. Na slikama 4.16, 4.17 i 4.18 prikazani su rezultati na skupu za ispitivanje. Tijekom učenja uzimali smo 5%, 10% i 15% najboljih jedinki za ispitivanje na skupu za validaciju.



Slika 4.16: Rezultati na skupu za ispitivanje.
Za validiranje je odabrano 5% najboljih jedinki iz skupa za učenje

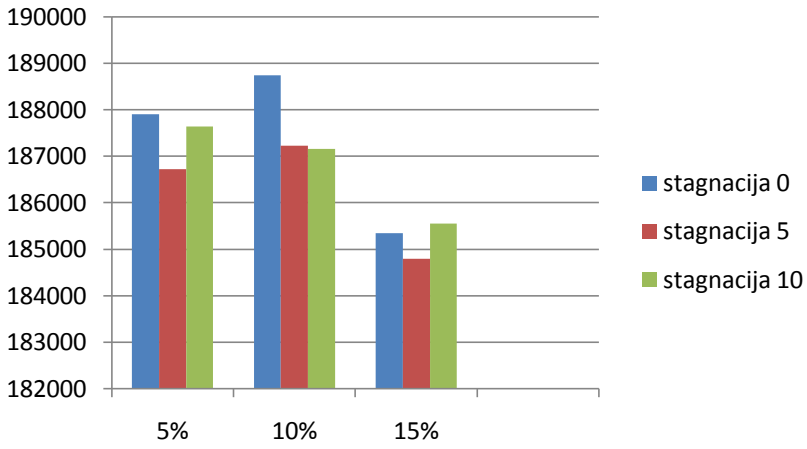


Slika 4.17: Rezultati na skupu za ispitivanje.
Za validiranje je odabrano 10% najboljih jedinki iz skupa za učenje



Slika 4.18: Rezultati na skupu za ispitivanje.
Za validiranje je odabrano 15% najboljih jedinki iz skupa za učenje

Na slici 4.19 uspoređeni su rezultati promatranja 5%, 10% i 15% najboljih jedinki za ispitivanje na skupu za validaciju. Veličina populacije je 500, a stagnacija 0, 5 i 10 generacija.



Slika 4.18: Usporedba rezultata korištenjem krosvalidacije uz promatranje 5% 10% i 15% najboljih jedinki iz skupa za učenje. Veličina populacije je 500.

5 Zaključak

Genetsko programiranje se može upotrijebiti za rješavanje problema iz područja strojnog učenja minimizirajući pogrešku između dobivenog i željenog rješenja problema kojeg rješavamo. GP predstavlja problem kao skup svih mogućih računalnih programa ili podskup tog skupa. Budući da se sustavi strojnog učenja mogu pokretati na računalu, odnosno prikazati kao računalni program, GP može evaluirati rješenje pronađeno pomoću takvog sustava.

Za rješavanje kompleksnih problema bitno je koristiti različite skupove podataka za učenje i za ispitivanje genetskog algoritma. Ako želimo dobiti još kvalitetnija rješenja potrebno je koristiti još jedan dodatni skup podataka, skup za validaciju. Učenjem na posebnom skupu podataka i validiranjem jedinki populacije genetskog programa na drugom skupu podataka sprječavamo pojavu prenaučenosti, odnosno ne dopuštamo GP-u da se previše specijalizira na jedan skup podataka dajući tako lošija rješenja za ostale, ne viđene, skupove podataka.

Dobiveni rezultati pokazali su da korištenje skupa za validaciju može dati bolja rješenja od ostalih uvjeta zaustavljanja učenja GP algoritma. Također je potrebno dalje eksperimentirati kako bi se pronašla najprikladnija konfiguracija veličine populacije, stabla i ostalih parametara.

6 Literatura

http://gp.zemris.fer.hr/ecf/index.html

http://people.brunel.ac.uk/~mastjjb/jeb/info.html

Jakobović, D. *„Raspoređivanje zasnovano na prilagodljivim pravilima“.* Doktorska disertacija. FER, 2005.

Koza, J. R. *Genetic Programming – On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.

David Gleich *„Machine learning in Computer Chess: Genetic Programming and KRK“ ,*[2003].

Darwin, C - „On the Origin of Species” 1859.

Wolfgang Banzhaf,Peter Nordin, Robert E. Keller, Frank D. Francone – „Genetic Programming An Introduction“ , Morgan Kaufman Publishers, Inc. 1998

Tom M. Mitchell - Machine Learning, McGraw-Hill Science/Engineering/Math; (March 1, 1997)

Ocjena učinkovitosti genetskog programiranja u postupcima strojnog učenja

Sažetak

U ovome radu opisan je mehanizam genetskog programiranja te su nabrojani i opisani osnovni elementi i operatiori genetskog programa. Definiran je i detaljno opisan pojam sustava strojnog učenja. Pokazana je važnost testiranja algoritma genetskog programa, njegove sposobnosti generalizacije, na skupu podataka različitom od skupa za učenje. Upotrijebili smo skup za validaciju na kojem algoritam ispituje kada bi trebao prestati učiti na skupu za učenje, odnosno kada se pojavila prenaučenost.

Genetsko programiranje prikazano je kao paradigma strojnog učenja i rabimo ga kako bi ocjenili njegovu učinkovitost u postupcima strojnog učenja. Problem koji rješavamo jest otkrivanje pogodnog uvjeta zaustavljanja prilikom strojnog učenja. Otkrivanje uvjeta temelji se na broju generacija i uporabi validacijskog skupa.

Navedeno je nekoliko primjera uporabe genetskog programiranja za rješavanje problema iz područja strojnog učenja. KRK (king-rook-king) problem kao jedan od primjera završnica u šahu te problem raspoređivanja poslova na jednom stroju.

Ključne riječi: evolucija, evolucijski algoritmi, genetski algoritmi, genetsko programiranje, genetski operatori, populacija, funkcija dobrote, strojno učenje, validacija, krosvalidacija.

Genetic programming efficiency in machine learning

Abstract

This paper describes the genetic programming mechanism and lists and describes basic elements and operators of the genetic program. It also defines and describes the notion of the machine learning system in detail. It demonstrates the importance of testing of the genetic program algorithm, of its ability of generalization on a set of data different from the learning set. We have used a set for validation on which the algorithm examines the time when it should stop learning on the learning set, that is, when the over-fitting phase occurs.

Genetic programming has been depicted as a machine learning paradigm and it is used in order to assess the effectiveness in the machine learning procedures. The problem we are trying to solve refers to the discovery of suitable condition to stop in the process of machine learning. The condition discovery is based on the number of generations and the validation set use.

This paper enumerates a few examples of the genetic programming use in order to solve problems related to machine learning, such as KRK (king-rook-king) problem as one of the examples of the chess finals and the problem of assigning tasks on one machine.

Key words: evolution, evolution algorithms, genetic algorithms, genetic programming, genetic operators, population, goodness function, machine learning, validation, cross validation.