UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Jana Jovičić

METODE ZA REŠAVANJE PROBLEMA SIMBOLIČKE REGRESIJE

master rad

Beograd, 2022.

Mentor:

dr Aleksandar KARTELJ, docent
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

prof. dr Nenad MITIĆ, redovni profesor
Univerzitet u Beogradu, Matematički fakultet

dr Milana GRBIĆ, docent
Univerzitet u Banjoj Luci, Prirodno-matematički fakultet

Datum odbrane: _____

Naslov master rada: Metode za rešavanje problema simboličke regresije

Rezime:

Simbolička regresija predstavlja tip regresione analize. Njen cilj je pronalazak modela kojim se najbolje opisuje dati skup podataka. Proces formiranja modela se sastoji od pretrage prostora matematičkih (odnosno simboličkih) izraza i konstruisanja izraza koji najbolje uspostavlja vezu između ulaznih promenljivih i ciljne promenljive. Tokom pretrage, istovremeno se optimizuju i struktura modela i njegovi parametri. U ovom radu su predstavljena tri načina rešavanja ovog problema - jedan pomoću metode grube sile i dva metaheuristička pristupa. Od metaheurističkih pristupa korišćeni su metoda promenljivih okolina i dve varijante genetskog programiranja, koje se međusobno razlikuju po načinu ukrštanja jedinki. Kod jedne varijante genetskog programiranja koristi se standardni operator ukrštanja, a kod druge operator ukrštanja koji je zasnovan na semantičkoj sličnosti. Kvalitet razvijenih metaheurističkih metoda je upoređen preko rezultata dobijenih testiranjem na instancama problema različitih dimenzija. Na instancama manjih dimenzija, metaheurističke metode su upoređene i sa rezultatima dobijenim algoritmom grube sile. Za evaluaciju su korišćene slučajno generisane instance izraza koji se često razmatraju u literaturi vezanoj za ovaj problem, kao i jedan od javno dostupnih skupova podataka za regresiju.

Ključne reči: simbolička regresija, metaheuristike, genetsko programiranje, metoda promenljivih okolina, optimizacija

Sadržaj

Glava 1

Uvod

Osnovu bilo kog naučnog procesa obično čini modelovanje različitih osobina fizičkih sistema, koji su predmet njihovog istraživanja, pomoću promenljivih, kao i razumevanje odnosa između tih promenljivih. Simbolička regresija nastoji da otkrije te odnose tako što pretražuje prostor matematičkih izraza kako bi pronašla onaj koji najbolje opisuje dostupni skup podataka. Simbolička regresija predstavlja tip regresione analize koji istovremeno uči i o strukturi modela kojim se opisuju dati podaci, i njegove parametre. Kako ne zahteva prethodnu specifikaciju strukture modela, simbolička regresija je pod manjim uticajem ljudske greške ili nedovoljnog domenskog znanja, u poređenju sa drugim metodama koje već unapred pretpostavljaju formu modela.

Problem simboličke regresije se smatra problemom kombinatorne optimizacije. Pošto je instance velikih dimenzija praktično nemoguće rešiti usled ograničenja vremenskih i memorijskih resursa, najčešće se traže aproksimativna rešenja problema, uglavnom pomoću razlčitih metaheurističkih metoda. Zbog ovoga se smatra da je simbolička regresija NP-težak problem, međutim to još uvek nije formalno dokazano.

U ovom radu će biti prikazane dve metaheurističke metode i jedna metoda grube sile za rešavanje ovog problema. Rad je podeljen u pet poglavlja. U drugom poglavlju (2) izložen je kratak pregled najčešće korišćenih regresionih tehnika, prikazan je način evaluacije regresionih modela i predstavljen je problem simboličke regresije. U trećem poglavlju (3) je prikazan algoritam grube sile za rešavanje problema simboličke regresije. Poglavlje 4 se bavi metaheurisičkim metodama za rešavanje ovog problema. U njemu su predstavljene metoda promenljivih okolina i dve varijante genetskog programiranja, koje se međusobno razlikuju po načinu ukrštanja jedinki. Kod jedne varijante genetskog programiranja koristi se standardni operator ukrštanja, a kod druge operator ukrštanja koji je zasnovan na semantičkoj sličnosti. Eksperimentalni rezultati

su prikazani u poglavlju 5.

Glava 2

Pojam simboličke regresije

2.1 Regresija

Regresija predstavlja tehniku za modelovanje veze između zavisne (ciljne) promenljive i jedne ili više nezavisnih promenljivih (atributa). Pripada grupi algoritama mašinskog učenja koji se zasnivaju na nadgledanom učenju.

Kod algoritama nadgledanog učenja za svaki uzorak iz skupa za trening data je i tačna izlazna vrednost za taj uzorak. Podaci o izlaznim vrednostima koriste se u obuci algoritama koji će za neki novi neobeleženi uzorak predvideti vrednost izlaza. Problemi nadgledanog učenja se nazivaju regresionim ukoliko je izlazna promenljiva kontinualnog tipa. U ovom slučaju cilj je formiranje modela koji će na osnovu dostupnih uzoraka za trening i odgovarajućih izlaznih vrednosti predvidati vrednost kontinualne izlazne promenljive za novi uzorak.

Pošto se regesija uglavnom koristi za predviđanje vrednosti, modelovanje vremenskih serija i određivanje uzročno-posledičnih veza između promenljivih, ima primenu u različitim oblastima, poput finansija, marketinga, fizike i biologije.

Postoje različiti tipovi regresione analize, poput linearne regresije, polinomijalne regresije i simboličke regresije.

Linearna regresija

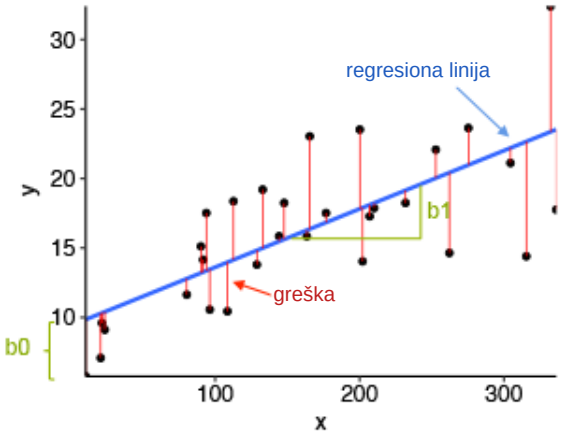
Jedan od najjednostavnijih i najčešće korišćenih tipova regresije je linearna regresija. Linearna regresija predstavlja metodu nadgledanog učenja koja se koristi za predviđanje vrednosti kontinualne izlazne promenljive uz pretpostavku da se ta vrednost može dobiti kao linearna kombinacija vrednosti ulaznih obeležja. Formiranje modela

linearne regresije svodi se na određivanje parametara pretpostavljene linearne zavisnosti.

Kako model linearne regresije pretpostavlja linearnu zavisnost po parametrima β_0, β_1, ..., β_m između atributa x_1, x_2, ..., x_m i ciljne promenljive y, onda se y predstavlja u obliku

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_m x_m.$$

Geometrijski posmatrano, cilj linearne regresije je pronalazak prave koja najbolje opisuje date podatke. Na slici 2.1 se može videti primer proste linarne regresije predstavljene modelom $f(x) = \beta_0 + \beta_1 x$.



Slika 2.1: Linearna regresija

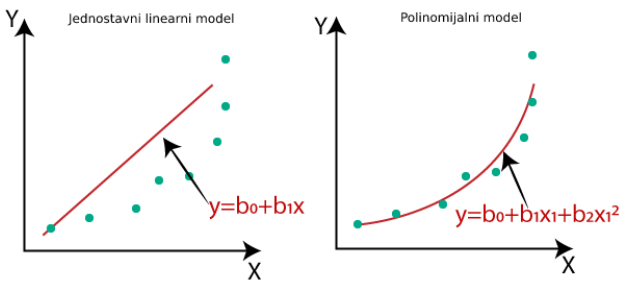
Polinomijalna regresija

Polinomijalna regresija je tip regresione analize kod koje se veza između atributa i ciljne promenljive modeluje pomoću polinoma n-tog stepena. U slučaju jedne nezavisne promenljive, ciljna promenljiva je oblika

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + ... + \beta_m x_1^m$$

Na slici 2.2 može se videti uporedni prikaz proste linearne i polinomijalne regresije.

Ovakav model se i dalje smatra linearnim, jer su težine pridružene atributima linearne. Samo je kriva koju modelujemo polinomijalnog obilka.



Slika 2.2: Polinomijalna regresija

Simbolička regresija

Simbolička regresija predstavlja generalizaciju linearne ili polinomijalne regresije. Dok one pretpostavljaju linearnu ili polinomijalnu formu modela i optimizuju samo numeričke parametre tog predefinisanog modela, simbolička regresija istovremeno uči i o strukturi modela i njegove parametre. Upravo zbog toga što ne zahteva prethodnu specifikaciju strukture modela, na simboličku regresiju manje utiču ljudske greške i nedovoljno domensko znanje.

U suštini, cilj simboličke regresije je pronalazak matematičkog izraza u simboličkoj formi, koji dobro modeluje vezu između ciljne promenljive i nezavisnih promenljivih.

Formalno, ako je dat skup podataka (X_i, y_i), $i = 1, ..., n$, gde $X_i \in \mathbb{R}^n$ predstavlja i-ti skup atributa, a $y_i \in \mathbb{R}$ i-tu ciljnu promenljivu, cilj simboličke regresije je pronalazak funkcije $f : \mathbb{R}^n \to \mathbb{R}$ koja najbolje odgovara skupu podataka, odnosno za koju važi $y_i \approx f(X_i), i = 1, ..., n$.

Za formiranje modela, koji pronalazi tu vezu, mogu se koristiti i različite metode dubokog učenja. Međutim, modeli formirani tim tehnikama su u vidu crne kutije i teški su za interpretaciju. Prednost simboličke regresije je u tome što je naučeni model lakše interpretirati, što može biti značajno u nekim oblastima primene koje zahtevaju mogućnost objašnjenja odluke koja je doneta pomoću tog modela. Na primer, ako se u bankarstvu odluka o izdavanju kredita donosi na osnovu modela, može biti potrebno obrazloženje donete odluke, pogotovo ako je korisnik odbijen.

Takođe, model dobijen simboličkom regresijom se može dalje analizirati. Pošto se radi o matematičkoj funkciji, moguće je, na primer, odrediti njenu periodičnost, asimptotsko ponašanje ili bilo koju drugu osobinu koja može biti od značaja za posmatranu oblast.

2.2 Evaluacija regresionih modela

Za evaluaciju regresionih modela, pa tako i modela simboličke regresije, mogu se koristiti različite metrike, poput srednje kvadratne greške i koeficijenta određenosti R^2, takođe poznatog i pod nazivom koeficijent determinacije (eng. *coefficient of determination*). Srednje kvadratna greška se izražava u terminima veličine ciljne promenljive, dok je vrednost koeficijenta određenosti normirana.

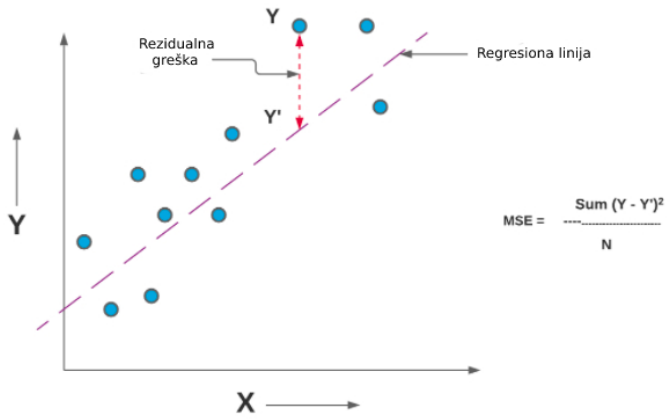
Srednje kvadratna greška (MSE, eng. *Mean Squared Error*) se izražava kao

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2,$$

gde je n broj uzoraka, y_i stvarna vrednost ciljne promenljive za i-ti uzorak, a $\hat{y}_i$ predviđena vrednost.

Vrednost MSE je uvek pozitivna, pri čemu vrednost bliža nuli označava kvalitetniji model.

Vizualna reprezentacija srednje kvadratne greške je prikazana na slici 2.3.



Slika 2.3: Vizualna reprezentacija srednje kvadratne greške

Koeficijent određenosti se izračunava kao količnik zbira kvadrata regresije (SSR, eng. *Sum of Squares Regression*) i ukupnog zbira kvadrata (SST, eng. *Sum of Squares Total*). SSR predstavlja varijansu predviđenih vrednosti koje se nalaze na regresionoj pravoj ili ravni u odnosu na srednju vrednost ciljne promenljive iz uzorka. SST predstavlja varijansu stvarnih vrednosti ciljne promenljive u odnosu na srednju vrednost ciljne promenljive iz uzorka. Odnosno, koeficijent određenosti se izračunava kao

$$R^2 = \frac{SSR}{SST} = \frac{\sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2},$$

gde je $\bar{y}$ srednja vrednost ciljne promenljive iz uzorka, $\hat{y}_i$ predviđena vrednost ciljne promenljive, a y_i stvarna vrednost ciljne promenljive.

Što je veća R^2 vrednost, to je i naučeni model bolji, jer je njime objašenjen veći udeo varijanse ciljne promenljive u odnosu na srednju vrednost.

R^2 se može izraziti i preko SSE greške (eng. *Sum of Squares Error*). SSE predstavlja deo varijanse koja nije obuhvaćena modelom. U tom slučaju važi

$$\begin{aligned} R^2 &= 1 - \frac{SSE}{SST} \\ &= 1 - \frac{\sum_{1=1}^{n}\left(y_i - \hat{y}_i\right)^2}{\sum_{i=1}^{n}\left(y_i - \bar{y}\right)^2} \\ &= 1 - \frac{\frac{1}{n}\sum_{1=1}^{n}\left(y_i - \hat{y}_i\right)^2}{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \bar{y}\right)^2}. \\ &= 1 - \frac{MSE}{\mathrm{Var}(y)} \end{aligned}$$

Lako se uočava da veća vrednost R^2 znači manju MSE vrednost.

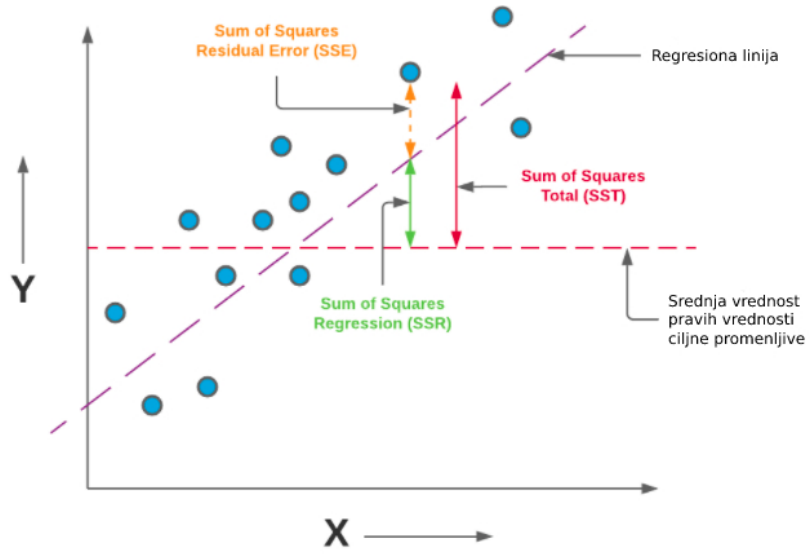
Dok na skupu podataka za trening R^2 uzima vrednosti iz intervala [0, 1], na skupu podataka za test može da se desi da vrednost bude i negativna (u slučaju kada je SSE veće od SST), što ukazuje da se radi o vrlo lošem modelu.

Vizualna reprezentacija koeficijenta određenosti je prikazana na slici 2.4.

2.3 Reprezentacija izraza kod simboličke regresije

Postoje različiti načini na koje mogu da se predstave matematički izrazi sa kojima se operiše u okviru simboličke regresije. Na primer, izraz se može konstruisati pomoću pravila kontekstno-slobodnih gramatika [6, 7, 10] ili se može predstaviti u vidu sintaksnog stabla [8, 18, 2, 21]. Pošto će u ovom radu biti korišćena reprezentacija pomoću sintaksnog stabla, u nastavku će biti prikazan samo taj pristup.

Pomoću sintaksnog stabla izraz se obično predstavlja u prefiksnoj notaciji. Listovi sintaksnog stabla mogu da sadrže samo terminale izraza, odnosno mogu predstavljati konstante i nezavisne promenljive iz datog skupa podataka. Unutrašnjim čvorovima stabla su predstavljene unarne i binarne funkcije (npr. +, -, sin, cos, log, ...). U okviru jednog stabla ista funkcija se može pojaviti veći broj puta. Isto važi i za konstante i

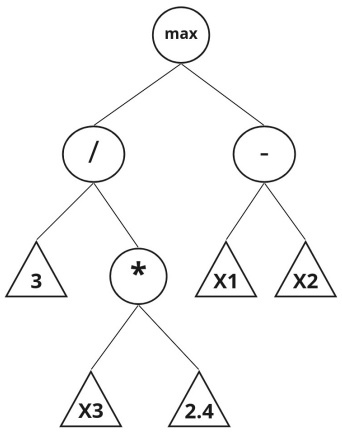


Slika 2.4: Vizualna reprezentacija koeficijenta određenosti

promenljive. Dodatno, skupovi funcija i terminala su fiksirani u skladu sa trenutnim problemom koji se rešava.

Slika 2.5 ilustruje primer sintaksnog stabla za izraz

$$\max\left\{\frac{3}{x_3 * 2.4}; x_1 - x_2\right\}$$



Slika 2.5: Primer sintaksnog stabla

Prilikom ocenjivanja tačnosti regresionog modela treba uzeti u obzir da se jedna ista funkcija može izraziti pomoću više simbolički različitih zapisa. Na primer, ako je

ciljna funkcija f jednaka $(u+v)/(1+uv/c^2)$ onda se i simbolički različit zapis $(v+u)/(1+uv/c^2)$ treba smatrati tačnim rešenjem. Odnosno, smatra se da je ciljna funkcija f ispravno određena kandidatskom funkcijom f' ako algebarska simplifikacija izraza $f'-f$ daje simbol „0" [20].

2.4 Simbolička regresija kao NP-težak problem

Razlikujemo sledeće klase složenosti:

- **Klasa P.** Ovoj klasi pripadaju svi problemi za koje postoji algoritam polinomske složenosti koji ga rešava.
- **Klasa NP.** Ovoj klasi pripadaju svi problemi za koje se može izvršiti provera rešenja u polinomskom vremenu. Jasno je da je $P \subset NP$, međutim za sada nije poznato da li je i $NP \subset P$. Problem određivanja da li je $P \neq NP$ je jedan od najpoznatijih otvorenih problema u teorijskom računarstvu.
- **NP-teški problemi.** Problem je NP-težak ako se svaki problem iz klase NP može u polinomskom vremenu redukovati na njega. Da bi se dokazalo da je problem NP-težak, dovoljno je dokazati da postoji polinomska redukcija bar jednog NP-teškog problema na njega.
- **NP-kompletni problemi.** Problem je NP-kompletan ukoliko pripada klasi NP i ukoliko je NP-težak.

Problem simboličke regresije se smatra problemom kombinatorne optimizacije.

Kako je prostor matematičkih izraza diskretan (u smislu strukture modela) i kontinualan (u smislu parametara modela, npr. konstante mogu biti realni brojevi), prostor pretrage se može eksponencijalno povećati sa povećanjem dužine izraza.

Zbog navedenih karakteristika, postavljena je hipoteza da je simbolička regresija NP-težak problem, međutim to još uvek nije formalno dokazano [17, 3]. Ali, pošto je instance velikih dimenzija praktično nemoguće rešiti usled ograničenja vremenskih i memorijskih resursa, najčešće se traže aproksimativna rešenja poroblema, uglavnom pomoću razlčitih metaheurističkih metoda.

Glava 3

Algoritam grube sile za rešavanje problema simboličke regresije

Radi dobijanja tačnih rešenja na instancama manjih dimenzija, kao i za proveru tačnosti metaheurističkih metoda, u radu je implementiran i jedan algoritam grube sile, koji se zasniva na sistematičnoj pretrazi prostora matematičkih izraza. Pretraga podrazumeva isprobavanje svih mogućih kombinacija podizraza sve dok se ne stigne do zadovoljavajućeg rešenja.

Po ugledu na algoritam grube sile koji se koristi kao deo metode razvijene u [20], potraga za rešenjem je implementirana iterativno po visini sintaksnog stabla izraza.

Prvo se proveravaju sva stabla visine 1 koja su generisana na osnovu svih datih funkcija i promenljivih. Ako među njima postoji izraz koji nad datim skupom podataka daje MSE grešku manju od definisanog parametra $\epsilon = 10^{-6}$ (dodatno pooštren kriterijum iz [20]), smatra se da je pronađeno tačno rešenje i pretraga se prekida. Ako takav izraz nije pronađen generišu se sva stabla visine 2. Postupak se ponavlja sve dok se ne pronađe tačno rešenje ili dok se ne dostigne definisano vremensko ograničenje. Pseudokod ovog postupka se može videti u algoritmima 1 i 2.

Kod procedure za konstruisanje stabala, za svaku od ulaznih funkcija se formiraju sva moguća stabla tako da se data funkcija f nalazi u korenu, dok podstabla čine do sada generisani terminali.

Pri prvom prolazu kroz proceduru, terminali su samo promenljive formirane na osnovu atributa datog skupa podataka (definisano u alg. 1, linije 1 i 3). Pri svakom narednom prolasku, u skup terminala se dodaju i stabla kreirana u prethodnoj iteraciji (alg. 1, linija 8).

Od terminala su generisane permutacije sa ponavljanjem dužine 2, tako da se dobi-

Algoritam 1 Algoritam grube sile za simboličku regresiju

Ulaz: X - skup vrednosti atributa, y - skup vrednosti ciljne promenljive, F - skup funkcija koje je moguće koristiti za kreiranje stabla izraza, ϵ - dozvoljena epsilon okolina greške, $vreme$ - maksimalan broj sati izvršavanja programa;

Izlaz: $resenje$ - izraz koji predstavlja tačno rešenje problema, $najblizeResenje$ - izraz koji daje najmanju grešku, $uspeh$ - indikator koji govori da li je pronadjeno tačno resenje;

1: **Inicijalizacija:** t = lista promenljivih // formiranih na osnovu ulaznog skupa podataka;
2: **while** vremenski kriterijum zaustavljanja nije ispunjen **do**
3: T = svi mogući parovi terminala iz t; // *generisani kao permutacije dužine 2*
4: $uspeh, resenje, S, najblizeResenje = konstruisiStabla(T, F, X, y, \epsilon)$;
5: **if** $uspeh$ **then**
6: **return** $uspeh, resenje, najblizeResenje$;
7: **end if**
8: $t = t \cup S$;
9: **end while**
10: **return** $uspeh, resenje, najblizeResenje$;

ju sva moguća uparivanja terminala, koja će se dalje koristiti kao podstabla za trenutnu korenu funkciju f (alg. 1, linija 3).

Za svaku korenu funkciju f se prolazi kroz sve generisane parove terminala (alg. 2, linije 2 i 3).

Ako je f unarna funkcija, proverava se da li je prvi član trenutnog para isti kao i prvi član prethodno obrađenog para. Ako jeste, prelazi se na naredni par (alg. 2, linije 4-8). Kako su parovi generisani iz skupa terminala kao permutacije dužine 2, u formiranom skupu biće grupisani po prvom paru, tako da će se dešavati da uzastopni parovi imaju isti prvi član. U tom slučaju, kako je f unarna funkcija, već je u prethodnoj iteraciji bilo kreirano podstablo pomoću istog terminala, pa ga nema potrebe ponovo kreirati.

Kao levo podstablo funkcije f postavlja se prvi član iz para, a u sličaju da je f binarna funkcija, kao desno podstablo postavlja se drugi član iz para (alg. 2, linije 9-12).

Tako formirana funkcija f se dodaje u skup stabala koja su generisana u ovoj iteraciji, kako bi pri narednoj mogla da se iskoristi kao terminal (alg. 2, linija 13).

Zatim se računaju vrednosti funkcije f u datim tačkama X i određuje se greška u odnosu na date stvarne vrednosti ciljne promenljive y (alg. 2, linije 14 i 15). Ukoliko je greška manja od trenutno najmanje greške, f se pamti kao nova najbolja funkcija (alg. 2, linije 16-19). Ako je greška manja od prosleđene vrednosti ϵ, f se uzima kao rešenje problema i dalja pretraga se prekida (alg. 2, linije 20-24). Najbliže rešenje se pamti kako

Algoritam 2 konstruisiStabla()

Opis: Algoritam za konstruciju novih stabala i njihovu evaluaciju pri potrazi za rešenjem.

Ulaz: T - skup parova terminala, F - skup funkcija koje je moguće koristiti za kreiranje stabla izraza, X - skup vrednosti atributa, y - skup vrednosti ciljne promenljive, ϵ - dozvoljena epsilon okolina greške;

Izlaz: $uspeh$ - indikator koji govori da li je pronadjeno tačno resenje, $resenje$ - tačno rešenje čija je greška manja od ϵ, S - skup generisanih stabala, tj. skup izraza čije stablo ima visinu jednaku trenutnoj iteraciji, $najblizeResenje$ - izraz koji daje najmanju grešku, ali ne nužno manju od ϵ;

Inicijalizacija: $minGreska = inf$; $S = []$; $resenje = ''$; $uspeh = False$;
for $f \in F$ **do**
  for $par \in T$ **do**
    if f je unarna funkcija **then**
      if prvi član $para$ je jednak prvom članu prethodnog para **then**
        continue;
      end if
    end if
    $f.levo\_podstablo = par[0]$;
    if f je binarna funkcija **then**
      $f.desno\_podstablo = par[1]$;
    end if
    $S = S \cup f$
    $y_{pred} = f(X)$;
    $greska = MSE(y_{pred}, y)$;
    if $greska < minGreska$ **then**
      $najblizeResenje = f$;
      $minGreska = greska$;
    end if
    if $greska < \epsilon$ **then**
      $uspeh = True$;
      $resenje = f$;
      break;
    end if
  end for
  if $uspeh$ **then**
    break;
  end if
end for
return $uspeh, resenje, S, najblizeResenje$;

bi se i u slučaju dostizanja maksimalnog dozvoljenog vremena izvršavanja programa dobilo do tad najbolje pronađeno rešenje.

Može se desiti da algoritam grube sile ne uspe da stigne do tačnog rešenja ne samo zbog vremenskog ograničenja, već i zbog ograničenja memorijskih resursa. U svakoj iteraciji se povećava broj stabala koje je potrebno čuvati, jer ih je u narednoj iteraciji potrebno iskoristiti kao terminale. Ako je n broj funkcijskih simbola, a m broj parova terminala u trenutnoj iteraciji, broj stabala generisanih u toj iteraciji biće, u najgorem slučaju, jednak $n * m$.

Glava 4

Metaheurističke metode za rešavanje problema simboličke regresije

Kako simbolička regresija pripada grupi problema kombinatorne optimizacije, najčešći pristup njenom rešavanju je pomoću metaheuristika. U radu će biti predstavljeni pristupi rešavanja pomoću genetskog programiranja (uz različite vidove ukrštanja jedinki) i metode promenljivih okolina.

4.1 Genetsko programiranje

Genetsko programiranje pripada grupi algoritama evolutivnog izračunavanja (EC, eng. *Evolutionary Computation*) koji su zasnovani na Darvinovoj teoriji evolucije. Tokom 1970-ih, Holand je predložio genetski algoritam kao jedan vid evolutivnog izračunavanja [5], dok je njegov učenik Džon Koza 1988. patentirao genetsko programiranje. Tokom 1990-ih i ranih 2000-ih Koza je izdao veliki broj radova i knjiga o genetskom programiranju, od kojih je najznačajnija knjiga [8] koja predstavlja prvu knjigu iz ove oblasti, a sadrži i princip primene te metode na problem simboličke regresije.

Osnovna ideja genetskih algoritama (GA) i genetskog programiranja (GP) je simulacija procesa evolucije. Prema Darvinu, bolje prilagođene jedinke imaju veću šansu da prežive i putem razmnožavanja prenesu svoj genetski materijal u narednu generaciju. Na taj način, celokupna vrsta se kontinuirano usavršava. Generalno, genetski algoritam se sastoji u iterativnoj popravci inicijalne populacije rešenja koje se često nazivaju jedinke ili hromozomi. Svaka jedinka odgovara jednom rešenju u pretraživačkom prostoru. Jedinke se evaluiraju na osnovu funkcije prilagođenosti. U svakom koraku algoritma, nad jedinkama se primenjuju različiti genetski operatori. Operator selekcije

obezbeđuje da verovatnoća sa kojom jedinka prelazi u narednu generaciju zavisi od njene prilagođenosti. Primena operatora ukrštanja i mutacije na selektovane jedinke simulira proces reprodukcije. Razlika između GA i GP je u tome što su jedinke kod GA predstavljene linearnim struktuama poput nizova, dok GP radi nad stabloidnim strukturama. Algoritam 3 predstavlja uopštenu šemu genetskog programiranja. Gotovo svi koraci se mogu na veliki broj različitih načina prilagoditi konkretnoj primeni.

Algoritam 3 Osnovna struktura GP metode

Ulaz: GP parametri i kriterijum zaustavljanja;
Izlaz: najbolja jedinka tj. izraz koji predstavlja rešenje problema;

```
1: Generisati početnu populaciju jedinki;
2: Izračunati prilagođenost svake jedinke u populaciji;
3: while nije zadovoljen kriterijum zaustavljanja do
4:    Izabrati iz populacije skup jedinki za reprodukciju;
5:    Primenom operatora ukrštanja kreirati nove jedinke;
6:    if ispunjen uslov za mutaciju then
7:       Primeniti mutaciju nad novim jedinkama;
8:    end if
9:    Izračunati prilagođenost novih jedinki;
10:   Formirati novu generaciju na osnovu novih jedinki;
11: end while
12: return najbolja jedinka;
```

Reprezentacija jedinki i generisanje početne populacije

Kako jedinka treba da odgovara rešenju problema, u slučaju simboličke regresije pomoću jedne jedinke će biti predstavljen jedan izraz koji je u obliku sintaksnog stabla.

Kao skup funkcija koje se mogu naći u unutrašnjim čvorovima stabla, najčešće se koriste: +, -, *, /, exp, sin, cos, log.

Kao terminali se koriste promenljive definisane na osnovu dostupnog skupa podataka i efemerne slučajne konstante. Efemerna slučajna konstanta je slučajno generisani broj određenog tipa iz definisanog intervala. Kod problema čije su nezavisne promenljive realni brojevi, prirodno je odabrati da efemerne konstante takođe budu realnog tipa i da budu iz nekog prikladnog opsega (često je to [-1, 1]) [8].

Početna populacija jedinki se generiše na slučajan način. Taj proces se može implementirati na više različitih načina, čime se postiže formiranje inicijalnih stabala različitih veličina i oblika. Dva osnovna načina su „full” i „grow” metode [8].

Oba pristupa se zasnivaju na dubini stabla. Dubina stabla se definiše kao put od korena do najudaljenijeg lista.

„Full" metoda podrazumeva generisanje potpunog stabla, tj. stabla kod koga je put između korena i svakog lista jednak definisanoj maksimalnoj dubini. Ovo se postiže ograničavanjem izbora vrste čvora u zavisnosti od trenutne dubine. Ako se trenutni čvor koji se generiše nalazi na dubini koja je manja od maksimalne, može se izabrati samo čvor koji predstavlja funkciju. U suprotnom, ako se trenutni čvor koji se generiše nalazi na dubini koja je jednaka maksimalnoj, onda se može odabrati samo terminal.

„Grow" metoda podrazumeva generisanje stabala čiji oblici variraju. Od značaja je samo da dužina puta od korena do bilo kog lista ne bude veća od definisane maksimalne dubine, a dozvoljeno je da bude kraća. Ako se trenutni čvor koji se generiše nalazi na dubini koja je manja od maksimalne, može se izabrati ili čvor koji predstavlja funkciju ili čvor koji predstavlja terminal, pri čemu se izbor vrši na slučajan način. Ako se trenutni čvor koji se generiše nalazi na dubini koja je jednaka maksimalnoj, onda se može odabrati samo terminal.

Kod obe metode važi sledeće: Ako je trenutni čvor koji se generiše na dubini koja je manja od minimalne, moguće je odabrati samo funkciju.

Na osnovu [8], najbolji pristup generisanju inicijalne populacije bila bi „ramped half-and-half" metoda koja omogućava da se kreira širok spektar stabala različitih visina i oblika. Ona predstavlja kombinaciju „full" i „grow" metoda. Takođe, obezbeđuje i kreiranje podjednakog broja stabala po svakom nivou dubine od nivoa 2 do nivoa definisanog parametrom maksimalne dubine. Na primer, ako je definisana maksimalna dubina jednaka 6, onda će 20% stabala imati dubinu 2, 20% dubinu 3, ..., 20% dubinu 6. Dalje, za svaki nivo dubine 50% stabala se kreira pomoću „full", a 50% pomoću „grow" metode.

Funkcija prilagođenosti

Funkcija prilagođenosti (eng. *fitness function*) daje ocenu kvaliteta jedinke i utiče na verovatnoću izbora te jedinke za proces formiranja nove generacije. Svakoj jedinki je pridružena skalarna vrednost koja predstavlja njenu prilagođenost i rezultat je neke eksplicitno definisane funkcije. Ta funkcija npr. može biti MSE greška na skupu podataka za trening i u tom slučaju prilagođenije jedinke imaju manju vrednost fitnes funkcije.

U knjizi [8], Koza definiše 4 vrste funkcija prilagođenosti:

- „Raw" funkcija prilagođenosti (eng. *raw fitness*)
- Standardizovana funkcija prilagođenosti (eng. *standardized fitness*)
- „Adjusted" funkcija prilagođenosti (eng. *adjusted fitness*)
- Normalizovana funkcija prilagođenosti (eng. *normalized fitness*)

Definicije ovih funkcija nisu međusobno isključive, već se svaka od funkcija definiše na osnovu prethodno uvedene.

„Raw" funkcija prilagođenosti

„Raw" funkcija prilagođenosti se izražava u terminima problema koji se rešava. Kod simboličke regresije „raw" funkcija prilagođenosti se može posmatrati kao funkcija greške. Njena vrednost za neku jedinku odgovara zbiru distanci izmedju vrednosti koje daje izraz predstavljen tom jedinkom i pravih ciljnih vrednosti svih uzoraka iz trening skupa. Distanca se računa kao apsolutna vrednost razlike između predviđenih i pravih vrednosti. Formalno, „raw" funkcija prilagođenosti $r(i,t)$ za i-tu jedinku iz generacije t se računa kao

$$r(i,t) = \sum_{i=1}^{N} |y_i - \hat{y}_i|$$

gde je N broj uzoraka u trening skupu, y_i prava vrednost cljne promenljive, a $\hat{y}_i$ predviđena vrednost cljne promenljive pomoću i-te jedinke.

U zavisnosti od definicije problema, prilagođenije jedinke mogu imati ili manju ili veću $r(i,t)$ vrednost. Pošto „raw" prilagođenost kod simboličke regresije odgovara grešci, onda bolje jedinke imaju manju vrednost date funkcije.

Standardizovana funkcija prilagođenosti

Standardizovana funkcija prilagođenosti $s(i,t)$ redefiniše „raw" funkciju tako da njena vrednost za bolje jedinke uvek bude manja, bez obzira na vrstu problema koji se rešava.

Kako je kod simboličke regresije „raw" funkcija već definisana tako da manje vrednosti reprezentuju bolje jedinke, standardizovana funkcija će biti jednaka „raw" funkciji. Odnosno,

$$s(i,t) = r(i,t).$$

Kod problema kod kojih je „raw" prilagođenost definisana tako da se boljim jedinkama smatraju one sa većom vrednošću te funkcije, standardizovana funkcija bi bila jednaka razlici maksimalne moguće vrednosti „raw" funkcije (označene sa r_{max}) i vrednosti „raw" funkcije za datu jedinku

$$s(i,t) = r_{max} - r(i,t).$$

„Adjusted" funkcija prilagođenosti

„Adjusted" funkcija prilagođenosti $a(i,t)$ se računa na osnovu standardizovane funkcije na sledeći način

$$a(i,t) = \frac{1}{1+s(i,t)}.$$

Vrednost $a(i,t)$ pripada intervalu [0,1] i veća je za bolje jedinke.

Prednost ove funkcije prilagođenosti je u tome što naglašava razliku između dobrih i vrlo dobrih jedinki. Na primer, ako imamo dve loše jedinke čije su vrednosti standardizovane funkcije redom 64 i 65, njihove vrednosti prilagođene funkcije će biti 0.0154 i 0.0156. U oba slučaja, razlika između prilagođenosti ove dve loše jedinke nije velika. A ako imamo dve dobre jedinke čije su vrednosti standardizovane funkcije redom 4 i 3, njihove vrednosti prilagođene funkcije će biti 0.20 i 0.25. Ovde, iako su jedinke na osnovu vrednosti standardizovane funkcije bliske, na osnovu „adjusted" funkcije se dodatno ističe bolja od dve posmatrane dobre jedinke.

Normalizovana funkcija prilagođenosti

Normalizovana funkcija prilagođenosti $n(i,t)$ se računa na osnovu prilagođene funkcije na sledeći način

$$n(i,t) = \frac{a(i,t)}{\sum_{k=1}^{M} a(k,t)},$$

gde je M veličina populacije.

Odnosno, za jedniku i „adjusted" funkcija prilagođenosti se normalizuje u skladu sa „adjusted" funkcijama ostalih jedinki iz populacije.

Normalizovana funkcija prilagođenosti ima tri poželjne karakteristike:

- Njena vrednost pripada intervalu [0, 1].
- Veća je za bolje jedinke u populaciji.

- Zbir normalizovanih vrednosti prilagođenosti svih jedinki je jednak 1.

Nedostatak ove vrste funkcije prilagođenosti jeste u tome što se u populaciji može nalaziti neka jedinka čiji izraz nije definisan nad svim tačkama iz skupa podataka, pa će to uticati i na vrednost funkcije prilagođenosti ostalih jedinki. Naime, ako neka jedinka nije definisana nad svim tačkama iz skupa podataka, vrednost njene „raw" funkcije će takođe biti nedefinisana, a samim tim i vrednost „adjusted" funkcije. A to će onemogućiti izračunavanje normalizovane funkcije prilagođenosti čak i za neku drugu jedinku koja je definisana nad svim tačkama.

U ovom radu će biti korišćena „adjusted" funkcija prilagođenosti, po ugledu na [8].

Selekcija

Selekcija obezbeđuje čuvanje i prenošenje dobrih osobina populacije (tj. dobrog genetskog materijala) na sledeću generaciju. U svakoj generaciji, deo jedinki se izdvaja za reprodukciju i generisanje nove generacije. Izdvajanje jedinki koje će učestvovati u reprodukciji zasniva se na funkciji prilagođenosti i, generalno, prilagođenije jedinke imaju veću verovatnoću da budu izabrane.

Proces selekcije se može realizovati putem ruletske selekcije, turnirske selekcije, rangiranjem prema funkciji prilagođenosti ili na neki drugi način. U ovom radu je primenjena turnirska selekcija koja funkcioniše na sledeći način. Iz tekuće populacije se na slučajan način bira unapred zadati broj jedinki. Taj broj je određen parametrom k koji predstavlja veličinu turnira. Od izabranih k jedinki se zatim bira najbolje prilagođena jedinka. Time se i slabije prilagođenim jedinkama daje određena šansa da budu izabrane za reprodukciju.

Operatori ukrštanja

Za potrebe rešavanja problema simboličke regresije razvijani su različiti operatori ukrštanja. U ovom radu će biti predstavljene dve varijante genetskog programiranja - jedna koja koristi standardni operator uktštanja i druga koja koristi operator ukrštanja zasnovan na semantičkoj sličnosti.

Standardni operator ukrštanja

Operator ukrštanja (eng. *crossover*) se koristi u procesu reprodukcije u kom učestvuju jedinke izabrane procesom selekcije. U ukrštanju učestvuju dve jedinke koje se

nazivaju roditeljima. Rezultat ukrštanja su dve nove jedinke koje se nazivaju decom ili neposrednim potomcima. Očekivano je da deca nasleđuju osobine roditelja, uključujući njihovu prilagođenost, pa i da imaju bolju prilagođenost od njih.

Kod simboličke regresije, ukrštanje dve jedinke podrazumeva razmenu njihovih slučajno odabranih podstabala. Kod oba roditelja se odabere po jedan čvor, slučajnim izborom iz uniformne raspodele, koji će predstavljati tačku ukrštanja. Fragment ukrštanja svakog roditelja je podstablo određeno tim čvorom kao korenom i svim čvorovima koji se nalaze ispod njega. Zatim je potrebno samo razmeniti odabrane fragmente između te dve jedinke.

Primer standardnog ukrštanja se može videti na slici 4.1. Na slikama (a) i (b) su prikazane dve jedinke koje su odabrane za proces ukrštanja i na svakoj od njih su obeleženi čvorovi koji predstavljaju tačku ukrštanja, kao i podstabla određena tim čvorovima. Na slikama (c) i (d) su prikazane dve jedinke koje nastaju nakon razmene podstabala definisanih izabranim tačkama ukrštanja.

Treba imati u vidu da su jedinke koje su odabrane za ukrštanje obično različite veličine i da se vrlo brzo može desiti da jedinke koje nastaju ukrštanjem dostignu ogromne razmere (npr. slučajevima kada se u jednom stablu kao čvor ukrštanja izabere terminal, a u drugom stablu čvor koji određuje visoko podstablo). Kako bi se ovo onemogućilo, uveden je parameter kojim se ograničava veličina jedinki koje nastaju ukrštanjem. Ako se formira jedinka nedozvoljene veličine, umesto nje će u narednu generaciju ići jedan od roditelja.

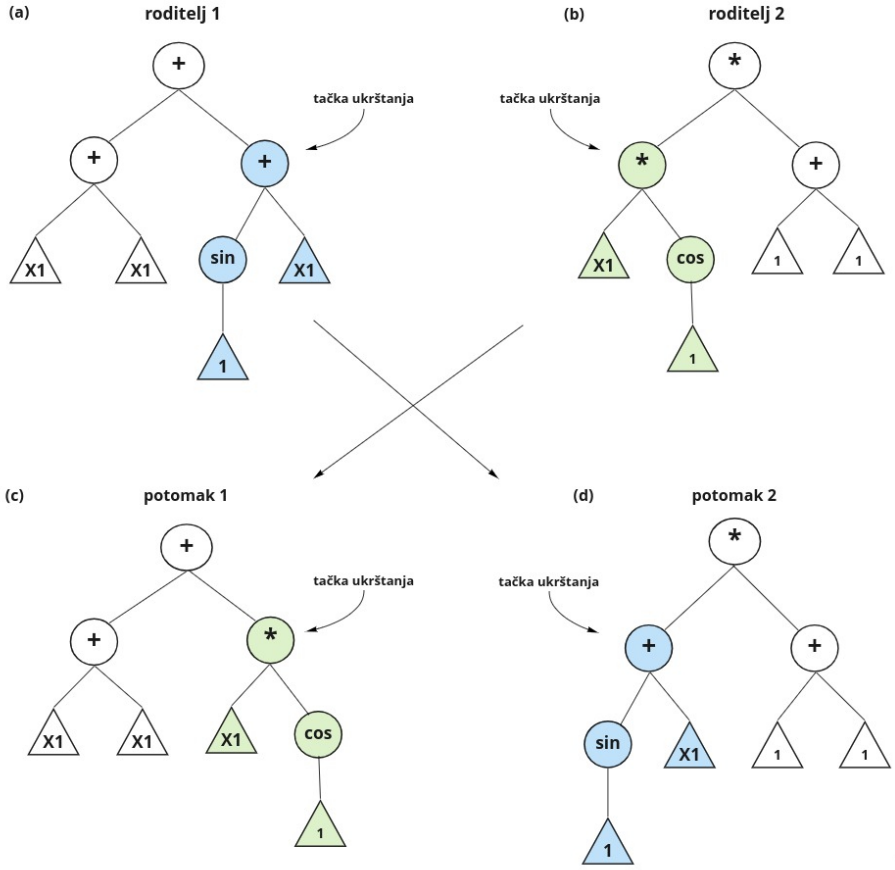
Operatori ukrštanja zasnovani na semantici

Pored osnovnog operatora ukrštanja, tokom godina su razvijani i drugi pristupi koji bi dali kvalitetnije jedinke. Neki od njih su i operatori ukrštanja koji su zasnovani na semantici [18].

Semantika nekog podstabla se aproksimira pomoću vrednosti dobijenih evaluacijom tog podstabla na predefinisanom skupu tačaka iz domena problema. Ovo se naziva *semantikom uzorkovanja* (SS, eng. *Sampling Semantics*). Formalno, *semantika uzorkovanja* nekog stabla (ili podstabla) se definiše na sledeći način:

Neka je F funkcija koja je izražena pomoću (pod)stabla T na domenu D i neka je P skup tačaka iz domena D, $P = \{p_1, p_2, ..., p_N\}$. Tada je *semantika uzorkovanja* stabla T na skupu P u domenu D, skup $S = \{s_1, s_2, ..., s_N\}$ takav da je $s_i = F(p_i), i = 1, 2, .., N$.

Vrednost broja N zavisi od problema. Ako se u skupu podataka nalazi veoma malo tačaka, SS može biti neprecizna, a ako ih ima mnogo, izračunavanje SS može biti



Slika 4.1: Primer standardnog ukrštanja dva stabla

vremenski zahtevno. Izbor tačaka P je takođe bitan. Ako su tačke suviše bliske skupu funkcija koje se koriste u GP (npr. π za trigonometrijske funkcije, e za logaritamske funkcije), semantika može biti pogrešno protumačena. Zbog toga se skup P bira na slučajan način.

Na osnovu SS definiše se *rastojanje semantike uzorkovanja* (SSD, *Sampling Semantics Distance*) između dva podstabla. Neka je $P = \{p_1, p_2, ..., p_N\}$ *semantika uzorkovanja* podstabla St_1, a $Q = \{q_1, q_2, ..., q_N\}$ *semantika uzorkovanja* podstabla St_2. Onda se SSD između St_1 i St_2 definiše kao

$$SSD(St_1, St_2) = \frac{1}{N}(|p_1 - q_1| + |p_2 - q_2| + ... + |p_N - q_N|).$$

Pomoću SSD se mogu definisati dve vrste semantičkih veza između podstabala - *semantička ekvivalentnost* i *semantička sličnost.*

Dva podstabla su *semantički ekvivalentna* (SE, eng. *Semantically Equivalent*) na domenu ako je njihova SSD vrednost dovoljno mala

$$SE(St_1, St_2) = \begin{cases} true, & \text{ako je } SSD(St_1, St_2) < \epsilon \\ false, & \text{inače} \end{cases}$$

Parametar ϵ predstavlja *semantičku osetljivost* (eng. *semantic sensitivity*). Na osnovu eksperimenata iz [11], za ϵ je najbolje uzeti neku vrednost iz skupa {0.01, 0.02, 0.04, 0.05, 0.06, 0.08, 0.1}.

Dva podstabla su *semantički slična* (SS_i, eng. *Semantically Similar*) na domenu ako njihova SSD vrednost leži na nekom pozitivnom intervalu:

$$SS_i(St_1, St_2) = \begin{cases} true, & \text{ako je } \alpha < SSD(St_1, St_2) < \beta \\ false, & \text{inače} \end{cases}$$

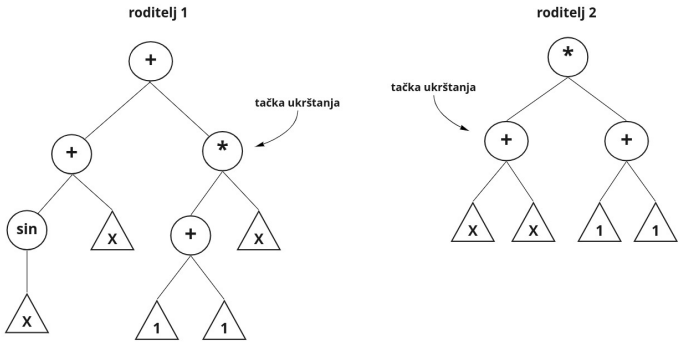
gde su α i β donja i gornja granica semantičke osetljivosti (LBSS, eng. *Lower Bound Semantic Sensitivity* i UBSS, eng. *Upper Bound Semantic Sensitivity*). Najbolje vrednosti za ove parametre variraju od problema do problema. U radu [18] se pokazalo da su za simboličku regresiju, nad jednačinama koje su korišćene i u ovom radu, najbolje rezultate dale vrednosti između 0.4 i 0.6 za $UBSS$, a vrednosti 10^{-2} ili manje za $LBSS$.

Intuicija korišćenja semantičke sličnosti je u tome što je verovatnije da će razmena podstabala biti korisnija ukoliko se odvija između dve jedinke koje nisu semantički identične, ali nisu ni semantički suviše različite. Motivacija za ovo je uzeta iz bioloških procesa i sličnosti MHC gena (koji imaju ulogu u imunom odgovoru) između dve jedinke. Za potomke je najbolje kada one imaju različite MHC gene, ali ne previše različite.

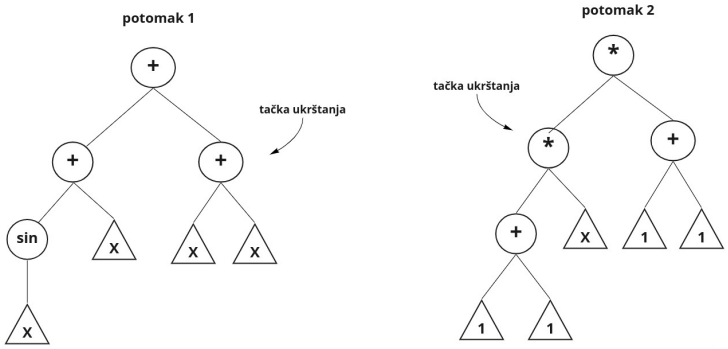
Operator ukrštanja koji je svestan semantike

Operator ukrštanja koji je svestan semantike (SAC, eng. *Semantics Aware Crossover*) je prvobitno predložen u radu [12]. Motivacija za uvođenje ovog operatora je bila mogućnost razmene semantički ekvivalentnih podstabala kod običnog operatora ukrštanja zasnovanog na slučajnom izboru, što rezultuje kreiranjem potomaka koji su identični svojim roditeljima. Na primer, ako su data dva stabla kao na slici 4.2 i ako su za razmenu odabrana označena podstabla, onda će kao rezultat ukrštanja nastati stabla kao na slici 4.3 . Iako su sintaksno različita, podstabla koja su izabrana za razmenu su semantički ekvivalentna (oba imaju semantiku $2X$). Ovakvim ukrštanjem ništa ni-

je postignuto - u populaciji i dalje ostaju dva stabla koja imaju istu vrednost funkcije prilagođenosti.



Slika 4.2: Selekcija semantički ekvivalentnih podstabala



Slika 4.3: Deca generisana ukrštanjem semantički ekvivalentnih podstabala

SAC sprečava razmenu ovakvih, semantički ekvivalentnih, podstabala. Svaki put kada se odaberu dva podstabla za razmenu, proverava se njihova ekvivalentnost pomoću formule za *SE*. Ako su ekvivalentna, ponovo se na slučajan način biraju tačke ukrštanja.

Operator ukrštanja koji je zasnovan na semantičkoj sličnosti

Operator ukrštanja koji je zasnovan na semantičkoj sličnosti (SSC, eng. *Semantic Similarity-based Crossover*) predstavlja proširenje SAC operatora. Kod ove metode ukrštanja, kada se odaberu podstabla, umesto semantičke ekvivalencije, proverava se nji-

hova semantička sličnost. Pošto je semantičku sličnost mnogo teže zadovoljiti u odnosu na semantičku ne-ekvivalentnost, verovatno je da će doći do uzastopnih neuspešnih pokušaja potrage za takvim podstablima. Zbog toga, SSC metoda koristi veći broj pokušaja da nađe semantički sličan par, a okreće se ka slučajnom izboru ako pređe dozvoljeni broj pokušaja. Algoritam 4 prikazuje SSC metodu.

Algoritam 4 Algoritam ukrštanja zasnovanog na semantičkoj sličnosti

Ulaz: $skupJedinki$ - skup jedinki iz kog je potrebno izabrati dve jedinke za ukrštanje, α - donja granica semantičke osetljivosti, β - gornja granica semantičke osetljivosti, $maxBrojPokusaja$ - maksimalan broj pokušaja odabira semantički sličnog para jedinki

Izlaz: ;

1: Izabrati roditelja P_1 iz $skupaJedinki$;
2: Izabrati roditelja P_2 iz $skupaJedinki$;
3: $brojPokusaja = 0$;
4: **while** $brojPokusaja < maxBrojPokusaja$ **do**
5: Na slučajan način izabrati podstablo St_1 u roditelju P_1;
6: Na slučajan način izabrati podstablo St_2 u roditelju P_2;
7: Na slučajan način generisati skup tačaka P iz domena D;
8: Izračunati SSD između St_1 i St_2 na tačkama P;
9: **if** $\alpha < SSD(\mathrm{St}_1, St_2) < \beta$ **then**
10: *// St_1, St_2 su semantički slični*
11: Izvršiti ukrštanje;
12: Dodati decu u novu populaciju;
13: **return** true;
14: **end if**
15: $brojPokusaja = brojPokusaja + 1$
16: **end while**
17: **if** $brojPokusaja == maxBrojPokusaja$ **then**
18: Na slučajan način izabrati podstablo St_1 u roditelju P_1;
19: Na slučajan način izabrati podstablo St_2 u roditelju P_2;
20: Izvršiti ukrštanje;
21: **return** true;
22: **end if**

Operatori mutacije

Mutacija se primenjuje nakon procesa ukrštanja. Operator mutacije sa određenom (obično veoma malom) verovatnoćom menja jedan deo jedinke na određeni način. Uloga mutacije je da spreči da jedinke u populaciji postanu suviše slične i da pomogne u obnavljanju izgubljenog genetskog materijala. Time sprečava konvergenciju ka

lokalnom ekstremumu i omogućava razmatranje novih delova prostora pretrage u kojima se možda nalazi globalni ekstremum. Dovoljno je da se jedna jedinka približi globalnom ekstremumu pa da kroz nekoliko generacija veliki broj jedinki bude u tom delu prostora pretrage.

Postoje različiti načini mutacije stabloidnih jedinki. Neki od njih su mutacija pojedinačnih čvorova stabla i mutacija celog podstabla [23].

Mutacija pojedinačnog čvora (eng. *point mutation*) podrazumeva zamenu odabranog čvora nekim drugim čvorom. Terminal može biti zamenjen drugim slučajno izabranim terminalom, a funkcija može biti zamenjena drugom funkcijom iste arnosti.

Mutacija celog podstabla (eng. *subtree mutation*) obuhvata slučajni izbor čvora koji će predstavljati koren podstabla koje treba zameniti, generisanje novog stabla na slučajan način i njegovo postavljanje na mesto odabranog podstabla.

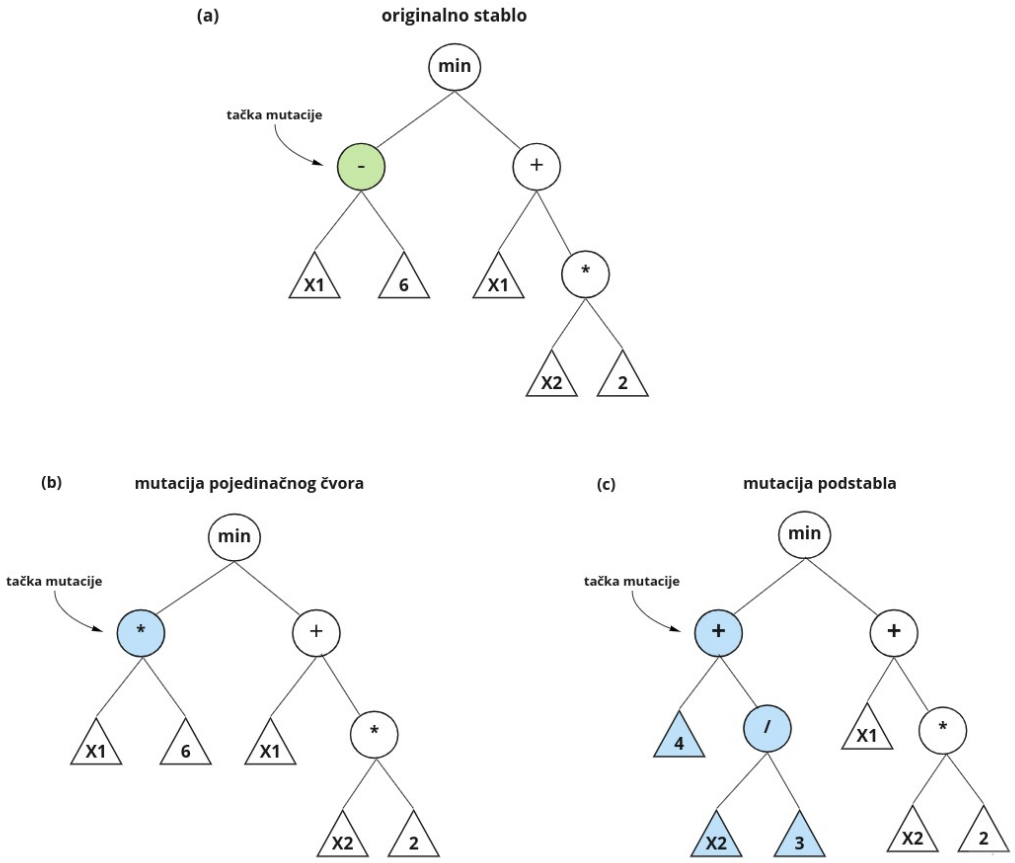
Na slici 4.4 se može videti primer obe vrste mutacije. Na slici (a) je predstavljeno originalno stablo nad kojim treba da se izvrši mutacija. Zelenom bojom obeležen je čvor koji predstavlja izabranu tačku mutacije. Na slici (b) je prikazano stablo koje nastaje kao rezultat primene mutacije pojedinačnog čvora, a na slici (c) stablo koje nastaje kao rezultat primene mutacije celog podstabla.

Zaustavljanje

Evolutivni proces stvaranja novih generacija se ponavlja sve dok nije zadovoljen neki uslov zaustavljanja [16]:

1. Pronađeno je rešenje koje zadovoljava unapred zadati kriterijum.
2. Dostignut je zadati broj generacija.
3. Dostignut je definisani vremenski kriterijum zaustavljanja.
4. Funkcija prilagođenosti je izračunata zadati broj puta.

U ovom radu su korišćeni uslovi 1, 2 i 3. u navedenom redosledu, pri čemu se u uslovu 1. smatra da je pronađeno zadovoljavajuće rešenje ako mu je „adjusted” funkcija prilagođenosti veća od 0.9.



Slika 4.4: Primer dve vrste mutacije stabla

4.2 Metoda promenljivih okolina

Metoda promenljivih okolina (VNS, eng. *Variable Neighbourhood Search*) je S-metaheuristika (*Single-solution-based* metaheuristika) zasnovana na lokalnoj pretrazi i unapređivanju jednog rešenja, za razliku od genetskog algoritma koji istovremeno radi nad velikim brojem rešenja. Lokalna pretraga polazi od početnog rešenja i u njegovoj okolini pokušava da pronađe bolje rešenje. Zatim se postupak pretrage nastavlja od tog rešenja, i ponavlja se sve dok se ne pronađe bolje rešenje. VNS metaheuristika je prvobitno predložena za rešavanje problema trgovačkog putnika u radu [14] koji su objavili Pjer Hansen i Nenad Mladenović. Metoda promenljivih okolina zasnovana je na tri činjenice:

1. Lokalni optimum u odnosu na jednu okolinu ne mora da predstavlja lokalni optimum u odnosu na drugu.

2. Globalni optimum je lokalni optimum u odnosu na sve moguće okoline.

3. Za mnoge probleme lokalni optimumi u odnosu na različite okoline su relativno blizu.

Poslednja činjenica ukazuje na to da lokalni optimum često daje neke informacije o globalnom. Zato se okoline lokalnog optimuma istražuju u potrazi za boljim rešenjem u nadi da će pretraga dovesti do globalnog optimuma [13].

Osnovnu varijantu metode promenljivih okolina (Basic VNS, eng. *Basic Variable Neighbourhood Search*) čine lokalna pretraga i stohastička procedura razmrdavanja (eng. *shaking*). Cilj ove procedura je da spreči da se pretraga zaglavi u nekom lokalnom optimumu. Osnovni koraci ove metode su:

1. Inicijalizacija: Izbor skupa okolina $N_k, k = 1, ..., k_{max}$; Konstruisanje početnog rešenja x; Izbor kriterijuma zaustavljanja.

2. Ponavljanje narednih koraka sve dok se ne ispuni kriterijum zaustavljanja:

   a) Postaviti k = 1

   b) Ponavljati naredne korake sve dok je $k \leq k_{max}$

      i. *Razmrdavanje* - Generisanje slučajnog rešenja x' iz okoline $N_k(x)$;

      ii. *Lokalna pretraga* - Primeniti neku metodu lokalne pretrage sa početnim rešenjem x'. Rezultat pretrage označiti sa x'';

      iii. *Prihvatanje rešenja i promena okoline* - Ako je tako dobijeno rešenje x'' bolje od x, postaviti $x = x''$ i $k = 1$; Inače, postaviti $k = k + 1$;

Kako su kod simboličke regresije rešenja predstavljena pomoću stabala, svaki od ovih koraka je bilo potrebno definisati u skladu sa takvim strukturama podataka. U radu [21] je predložena osnovna VNP (eng. *Basic Variable Neighbourhood Programming*) metoda koja je usklađena sa svim zahtevima simboličke regresije.

Tipovi okolina

Kod osnovne VNP metode se koriste tri vrste okolina:

- $N(T)$ označava strukturu susedstva koje se koristi tokom lokalne pretrage. Članovi ove vrste susedstva se formiraju elementarnim transformacijama stabla, koje će biti predstavljene nešto kasnije.

- $N_1(T)$ označava strukturu susedstva čiji se članovi dobijaju zamenom samo jednog čvora početnog stabla T. Čvorovi koji predstavljaju funkcije se mogu zameniti samo čvrovima koji predstavljaju funkcije iste arnosti. Slično, terminali se mogu zameniti samo terminalima. Dobijeni susedi imaju isti oblik i veličinu kao stablo T. Ovaj operator odgovara operatoru mutacije pojedinačnog čvora (eng. *point mutation*) kod genetskog programiranja.

  Na primer, ako T predstavlja izraz $\max\left\{\frac{3}{x_3*2.4}; x_1 - x_2\right\}$, a skupovi funkcija i terminala redom sadrže $\{*, -, +\}$ i $\{3, 2.4, 0.66, x1, x2, x3, x4\}$, onda su neki od suseda stabla T: $\max\left\{\frac{2.4}{x_3*2.4}; x_1 - x_2\right\}$, $\max\left\{\frac{x_1}{x_3*2.4}; x_1 - x_2\right\}$, $\max\{3 + x_3 * 2.4; x_1 - x_2\}$, itd.

- $N_2(T)$ predstavlja operator zamene (eng. *Swap operator*). U ovoj okolini, susedi od T se generišu tako što se u T nasumično odabre jedan čvor koji predstavlja koren podstabla koje treba da bude zamenjeno. Na njegovo mesto se postavlja novo, slučajno generisano stablo. Oblik i veličina generisanog suseda se razlikuju od početnog stabla T. Pomoću ovog operatora može se generisati beskonačan broj suseda. *Swap operator* odgovara operatoru mutacije celog podstabla (eng. *subtree mutation*) kod genetskog programiranja.

  Okoline $N_1(T)$ i $N_2(T)$ se koriste u proceduri razmrdavanja.

Razmrdavanje

Procedura razmrdavanja je prikazana u algoritmu 5. Ovom procedurom se dobija k-ti sused stabla T, primenom istog poteza k puta. Odnosno, prvo je potrebno nasumično izabrati okolinu $N_1(T)$ ili $N_2(T)$, a zatim taj operator primeniti k puta nad datim stablom.

Algoritam 5 Shake(T, k)

Ulaz: T - inicijalno stablo, k - broj ponavljanja primene operatora susedstva;
Izlaz: T' - stablo koje je sused stabla T;

```
1: s ← slučajna vrednost iz {1,2};
2: for i = 1, k do
3:     Na slučajan način izabrati T' ∈ N_s(T) ;
4:     T ← T';
5: end for
6: return T;
```

Elementarne transformacije i lokalna pretraga

Susedi koji se razmatraju tokom lokalne pretrage, generišu se elementarnim transformacijama trenutno najboljeg rešenja.

Elementarne transformacije stabla (ETT, eng. *Elementary Tree Transformation*) se definišu na sledeći način. Neka je $G(V,E)$ neusmereni graf sa skupom čvorova V i skupom grana E i neka je $T(V,A)$ neko razapinjuće stablo grafa G. ETT transformiše stablo T u stablo T' (u oznaci $T' = ETT(T)$) sledećim koracima:

1. U stablo T dodati granu a, takvu da $a \in E \setminus A$.
2. Detektovati formirani ciklus i ukloniti bilo koju granu (osim one koja je dodata u prethodnom koraku) iz njega kako bi se dobio podgraf T', koji takođe predstavlja razapinjuće stablo grafa G.

Ovaj postupak je opisan algoritmom 6. Ulazni parametar je stablo $T(r,F,H,E,d)$, definisano korenom r, skupom čvorova koji predstavljaju funkcije F, skupom čvorova koji predstavljaju terminale H, skupom grana E i nizom $d = (d_1,...,d_n)$ koji označava stepen svakog čvora stabla T. Stepen d_i označava broj grana koje su incidentne sa čvorom i. Ako čvor predstavlja binarnu funkciju, njegov stepen je 3, a ako predstavlja unarnu funkciju, njegov stepen je 2. Stepen čvora terminala je 1.

Naredbe iz algoritma 6 se mogu grupisati u četiri grupe (dve vrste dodavanja grane i dve vrste uklanjanja grane):

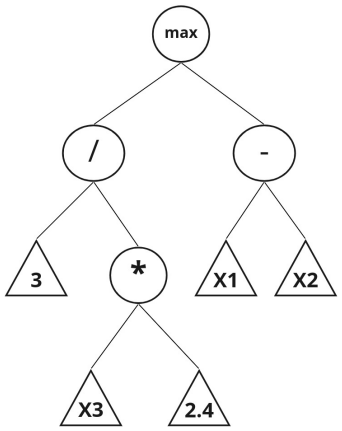
1. *Dodavanje grane (tip I)* (linije 4,5). Grana (i,j) se dodaje u trenutno stablo T, pri čemu ni i ni j ne smeju biti koren stabla. Dodatno, ili i ili j mora da bude funkcija. Nakon primene ovog koraka se formira ciklus u trenutnom stablu.
2. *Uklanjanje grane (tip I)* (linije 8,9,23,24). Ovde postoje dva slučaja. Ako su i i i j funkcije (slučaj 1 u algoritmu), onda se može ukloniti neka od grana $(i, parent(i))$, $(j, parent(j))$ kako bi se uklonio ciklus. Ako je jedan od čvorova terminal, npr. neka to bude j, onda se uklanja grana $(j, parent(j))$ (slučaj 2 u algoritmu).
3. *Dodavanje grane (tip II)* (linije 11,12,16,17,25,26). Ako je obrisana grana $(i, parent(i))$, onda se dodaje grana $(parent(i), child(j))$. Ako je obrisana grana $(j, parent(j))$, onda se dodaje grana $(parent(j), child(i))$.
4. *Uklanjanje grane (tip II)* (linije 13,14,18,19,27,28). Ove grane se uklanjaju kako bi stepen čvorova ostao zadovoljavajući.

Algoritam 6 ETT(T(r, F, H, E, d))

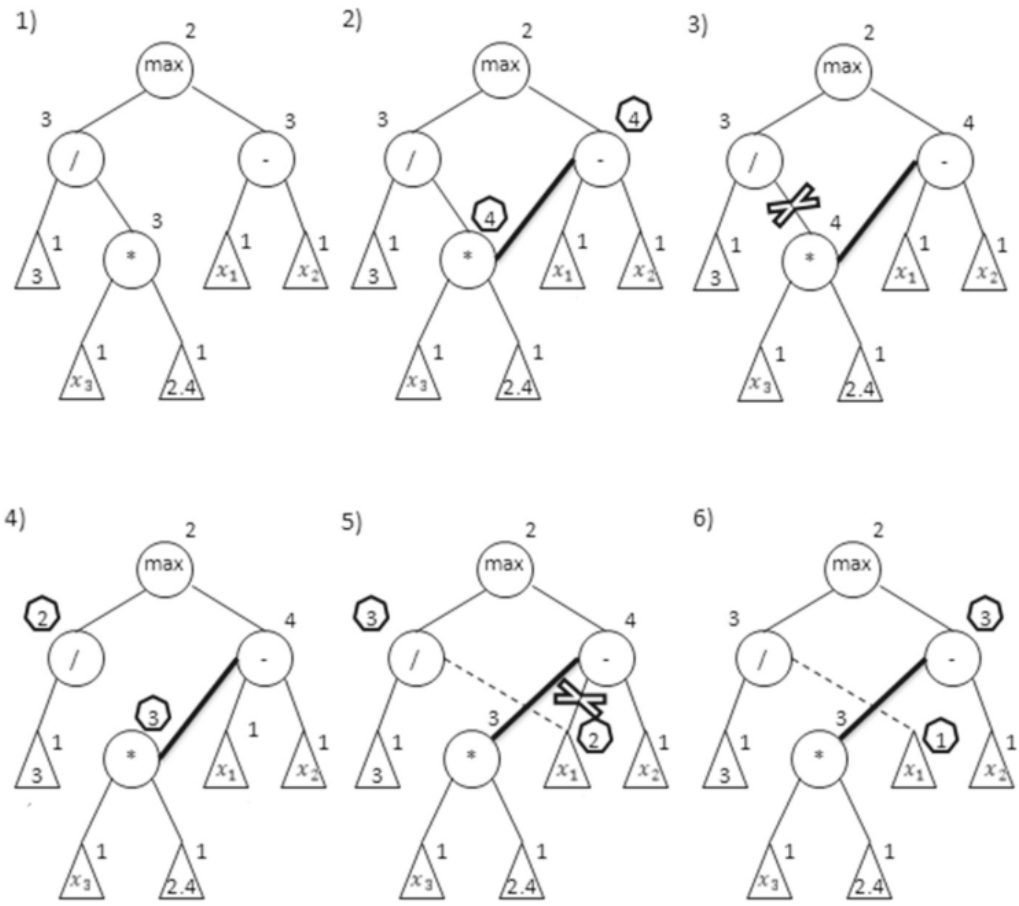
$Terminate \leftarrow false$;
for each node $i \in F$ **do**
  for each node $j \in F \cup H \setminus \{i, parent(i), child(i)\}$ **do**
    $E' \leftarrow E \cup \{(i, j)\}$; // ciklus je formiran
    $d_i \leftarrow d_i + 1; d_j \leftarrow d_j + 1$;
    if $j \in F$ **then**
      // slučaj 1
      $E' \leftarrow E' \setminus \{(x, b)\} / x \in \{i, j\}, b \in \{\text{parent}(i), \text{parent}(j)\}$;
      $d_b \leftarrow d_b - 1$;
      if $x = i$ **then**
        $d_i \leftarrow d_i - 1; E' \leftarrow E' \cup \{(b, \text{child}(j))\}$;
        $d_b \leftarrow d_b + 1; d_{\text{child}(j)} \leftarrow d_{\text{child}(j)} + 1$;
        $E' \leftarrow E' \setminus \{(j, \text{child}(j))\}$;
        $d_j \leftarrow d_j - 1; d_{\text{child}(j)} \leftarrow d_{\text{child}(j)} - 1$;
      else
        $d_j \leftarrow d_j - 1; E' \leftarrow E' \cup \{(b, \text{child}(i))\}$;
        $d_b \leftarrow d_b + 1; d_{child}(i) \leftarrow d_{\text{child}(i)} + 1$;
        $E' \leftarrow E' \setminus \{(i, \text{child}(i))\}$;
        $d_i \leftarrow d_i - 1; d_{\text{child}(i)} \leftarrow d_{\text{child}(i)} - 1$;
      end if
    else
      // slučaj 2
      $b \leftarrow \text{parent}(j); E \leftarrow E' \setminus \{(j, b)\}$;
      $d_j \leftarrow d_j - 1; d_b \leftarrow d_b - 1$;
      $E' \leftarrow E' \cup \{(b, child(i))\}$;
      $d_b \leftarrow d_b + 1; d_{\text{child}(i)} \leftarrow d_{\text{child}(i)} + 1$;
      $E' \leftarrow E' \setminus \{(i, \text{child}\ (i))\}$;
      $d_i \leftarrow d_i - 1; d_{\text{child}(i)} \leftarrow d_{\text{child}(i)} - 1$;
    end if
    $T' \leftarrow T\left(r, F, H, E', d\right)$;
    if $f\left(T\left(r, F, H, E', d\right)\right) > f(T(r, F, H, E, d))$ **then**
      $E \leftarrow E'$; $Terminate \leftarrow true$; Break;
    end if
  end for
  if $Terminate = true$ **then**
    Break;
  end if
end for

Na slici 4.6 je prikazan primer formiranja jednog suseda stabla sa slike 4.5 pomoću elementarnih transformacija. Koraci algoritma 6 na ovom primeru bi bili sledeći:

- $Terminate = false$
- $i \in F = \{-, /, *\}$: Npr. uzima se da je $i = '-'$. Kako je stepen funkcijskih čvorova jednak 3, važi $d_i = 3$.
- $j \in F \cup H \setminus \{i, parent(i), child(i)\}$: Npr. $j = '*', d_j = 3$
- $E' \leftarrow E \cup \{(i, j)\}$: U trenutno stablo se dodaje grana $('-', '*')$ (Slika 4.6, korak 2). Time se dobija ciklus koji povezuje čvorove $'-', '*', '/', 'max'$. Ovde se, takođe, povećavaju stepeni čvorova $'-'$ i $'*'$. Sada važi $d_i = 4$ i $d_j = 4$.
- Kako je odabano $j = '*'$, znači da $j \in F$, što znači da se izvršava prvi slučaj algoritma.
- $E' \leftarrow E' \setminus \{(x, b)\} / x \in \{i, j\}, b \in \{\text{parent}(i), \text{parent}(j)\}$: Ovde se može obrisati grana $(i, parent(i))$, što je $('-', 'max')$, ili grana $(j, parent(j))$, što je $('*', '/')$. Odabraće se npr. grana $('*', '/')$ (Slika 4.6, korak 3). Znači, $x = j$ i $b = parent(j)$, pa se dalje izvršava blok koda od 16. do 19. linije.
- Smanjuje se stepen čvorova j i $parent(j)$: $d_j = 3$, $d_{parent(j)} = 2$ (Slika 4.6, korak 4).
- $E' \leftarrow E' \cup \{(b, \text{ child}(i))\}$: Sada se dodaje grana $(parent(j), child(i))$, što može biti $('/', 'x_1')$ ili $('/', 'x_2')$. Neka je npr. odabrana $('/', 'x_1')$ (Slika 4.6, korak 5).
- Sada se povećava stepen čvora $child(i)$, što je $'x_1'$, i čvora $parent(j)$, što je $'/'$: $d_{parent(j)} = 3$, $d_{child(i)} = 2$.
- $E' \leftarrow E' \setminus \{(i, \text{ child}(i))\}$: Konačno, briše se grana $('-', 'x_1')$ i ažuriraju se stepeni $d_i = 3$ i $d_{child(i)} = 1$ (Slika 4.6, korak 6). Primećujemo da novo stablo $T(r, F, H, E', d)$ čuva stepen svakog čvora iz originalnog stabla $T(r, F, H, E, d)$.
- Ako je funkcija prilagođenosti novog stabla $T(r, F, H, E', d)$ bolja od funkcije prilagođenosti stabla $T(r, F, H, E, d)$, onda se postavlja $Terminate = true$.
- Algoritam se zaustavlja kada se naiđe na prvo bolje stablo ili kada se istraže sve strukture stabala u datom susedstvu.



Slika 4.5: Početno stablo



Slika 4.6: Primer primene ETT transformacija

Pomoću ETT je implementirana i lokalna pretraga koja koristi strategiju *prvog poboljšanja*. Kada se naiđe na prvo stablo koje daje bolje rezultate na datom skupu podataka, ono se vraća kao trenutno najbolje rešenje iz čijeg susedstva će se dalje nastaviti pretraga. Kao funkcija na osnovu koje se poredi kvaliet stabala, uzet je koeficijent determinacije R^2.

Osnovni VNP algoritam

Uzimajući u obzir sve prethodno definisane pojmove, osnovna VNP procedura za simboličku regresiju se može predstaviti algoritmom 7.

Algoritam 7 Basic VNP(T, k$_{max}$)

1: **repeat**
2: $k \leftarrow 1$;
3: **while** $k < k_{max}$ **do**
4: $T' \leftarrow Shake(T, k)$;
5: $T'' \leftarrow ETT(T')$;
6: $Neighborhood\_change(T, T'', k)$;
7: **end while**
8: **until** nije ispunjen kriterijum zaustavljanja;

Kriterijum zaustavljanja osnovne VNP procedure može biti pronalazak tačnog rešenja, maksimalan dozvoljeni broj iteracija ili maksimalno vreme izvršavanja.

Glava 5

Eksperimentalni rezultati

U ovom poglavlju će biti izloženi eksperimentalni rezultati predstavljenih metoda za rešavanje simboličke regresije. Sva testiranja su izvršena na računaru sa Intel Core i7 procesorom od 16GB i SSD-om, a sve metode su implementirane u programskom jeziku Python.

Podaci

Predstavljene metode su testirane pomoću tri vrste skupova podataka.

Jedna vrsta skupova je generisana na osnovu funkcija koje se često razmatraju u literaturi [21], [18], [2]. Instance ovog skupa su generisane na osnovu slučajno odabranih vrednosti nezavisnih promenljiih. Za svaku funkciju je generisano 100 instanci. Funkcije i intervali vrednosti iz kojeg su birane tačke su:

$F_1 = x^3 + x^2 + x, \quad x \in [-1, 1]$

$F_2 = x^4 + x^3 + x^2 + x, \quad x \in [-1, 1]$

$F_3 = x^5 + x^4 + x^3 + x^2 + x, \quad x \in [-1, 1]$

$F_4 = x^6 + x^5 + x^4 + x^3 + x^2 + x, \quad x \in [-1, 1]$

$F_5 = sin(x^2)cos(x) - 1, \quad x \in [-1, 1]$

$F_6 = sin(x) + sin(x + x^2), \quad x \in [-1, 1]$

$F_7 = log(x + 1) + log(x^2 + 1), \quad x \in [0, 2]$

$F_8 = sin(x_0) + sin(x_1^2), \quad x_0, x_1 \in [-1, 1]$

$F_9 = 2sin(x_0)cos(x_1), \quad x_0, x_1 \in [-1, 1]$

Drugu vrstu testa čini evaluacija implementiranih metoda nad nekim od javno dostupnih skupova podataka za regresiju. Ovde je iskorišćen „Yacht Hydrodynamics" skup [19], kod koga je cilj predvideti vrednost rezidualnog otpora jahte na osnovu nje-

Tabela 5.1: Vreme izvršavanja (izraženo u sekundama) svih metoda na poblemima manjih dimenzija

	Algoritam grube sile	**GP**	**GP sa SSC**	**VNP**
F_{01}	1	12	19	4
F_{02}	<1	7	13	6
F_{03}	<1	6	12	6
F_{04}	<1	12	18	5
F_{05}	<1	7	18	6
F_1	<1	15	24	7
F_2	22	13	26	8

nih karakteristika. Skup sadrži 308 instanci koje su određene pomoću 6 nezavisnih i jedne ciljne promenljive. Sve vrednosti su realnog tipa.

Dodatno, generisani su i skupovi podataka na osnovu nekih jednostavnijih funkcija koje su uzete radi upoređivanja metaheurističkih metoda sa metodom grube sile, jer metoda grube sile nije mogla uspešno da se nađe tačno rešenje na velikom broju prethodno pomenutih skupova. Te funkcije su:

$F_{01} = x_0 x_1 + x_1$

$F_{02} = x_1 + x_1^2 + x_0$

$F_{03} = x_0 x_1 + cos(x_0)$

$F_{04} = x_0 - x_1 x_1$

$F_{05} = x_0 - x_1 x_1 + x_1$

Evaluacija algoritma grube sile i poređenje sa metaheurističkim metodama

Algoritam grube sile je uspešno mogao da dođe do optimalnog rešenja za primere $F_{01}, ..., F_{05}$ i F_1 i F_2. U ostalim slučajevima je, nakon više sati izvršavanja, dolazilo do nedostatka memorijskih resursa, bez pronalaska dovoljno dobrog rešenja.

Svi metaheuristički pristupi su, takođe, uspešno pronašli optimalno rešenje za primere $F_{01}, ..., F_{05}$. U tabeli 5.1 je prikazano vreme izvršavanja koje je u proseku bilo potrebno svakoj od metoda za pronalazak ciljne funkcije. Kolone tabele predstavljaju metode, a redovi funkcije. U preseku i-te vrste i j-te kolone je prikazano vreme izvršavanja (izraženo u sekundama) j-te metode nad instancama i-te funkcije.

Evaluacija i poređenje metaheurističkih metoda

U ovoj sekciji će biti predstavljeni i međusobno upoređeni rezultati metaheurističkih metoda.

Kada je problem simboličke regresije tek počeo da se rešava, efikasnost metoda je predstavljana u terminima koji odgovaraju metaheuristikama [18], [2]. Na primer, jedan način za evaluaciju tačnosti je bio na osnovu srednje vrednosti najboljih vrednosti funkcija prilagođenosti dobijenih pri većem broju nezavisnih pokretanja programa. Drugi način se definisao na osnovu broja uspešnih pokretanja od ukupnog broja pokretanja programa, gde se pod uspešnim pokretanjem smatralo ono u kom postoji barem jedna jedinka koja za svaku instancu iz skupa podataka daje grešku manju od nekog definisanog praga.

Poslednjih godina metode za simboličku regresiju se evaluiraju kao i metode za sve druge tipove regresije [1], [9], [15]. To podrazumeva podelu skupa podataka na trening i test deo i evaluaciju dobijenog modela pomoću MSE, RMSE, R^2 i sličnih metrika. U ovom radu je primenjen ovaj pristup. Za trening je uzeto 70% podataka, za test 30% , a evaluacija je urađena pomoću R^2 koeficijenta određenosti. Veće R^2 vrednosti odgovaraju izrazima koji su bliži optimalnom. Optimalnim izrazom smatraće se i onaj izraz koji originalno ima različit simbolički zapis od ciljnog izraza, ali nakon algebarske simplifikacije dobija njegovu strukturu. Odnosno, ako za ciljni izraz f i nađeni izraz f' važi da algebarska simplifikacija izraza $f' - f$ daje simbol „0", biće smatrano da je pronađeno optimalno rešenje. Za proces algebarskog pojednostavljivanja je korišćen SymPy paket [22].

Svaka metoda je evaluirana tako što je pokrenuta po 30 puta nad svim skupovima podataka. Prilikom svakog od tih nezavisnih pokretanja dobijen je izraz koji u tom izvršavanju najbolje odgovara datom skupu podataka. Za taj izraz je zatim izračunat koeficijent određenosti na trening i test skupu i provereno je da li i simbolički odgovara ciljnom izrazu po postupku koji je naveden iznad. Naravno, provera optimalnosti u simboličkom smislu je rađena samo za skupove podataka kod kojih je poznata ciljna funkcija. Kod „Yacht Hydrodynamics" skupa taj deo je izostavljen. Takođe, pri svakom pokretanju mereno je i vreme koje je bilo potrebno za pronalazak najboljeg rešenja.

U tabeli 5.2 su prikazane vrednosti koje su korišćene za parametre genetskog programiranja. Odabrane su zbog čestog korišćenja u radovima koji se bave rešavanjem ovog problema pomoću genetskog programiranja i na osnovu dobrih rezultata pri inicijalnom testianju metode. Kod „Yacht" skupa je bilo potrebno povećati dozvoljenu veličinu populacije i veličinu jedinki. Parametri koišćeni za ovaj skup se mogu videti u

tabeli 5.3.

Tabela 5.2: Vrednosti parametara genetskog programiranja

Parametar	Vrednost
Veličina populacije	500
Maksimalan broj generacija	50
Broj jedinki za reprodukciju	300
Veličina turnira	3
Verovatnoća mutacie pojedinačnog čvora	0.2
Verovatnoća mutacije celog podstabla	0.2
Minimalna dubina stabla	2
Maksimalna dubina stabla u fazi inicijalizacije	6
Maksimalna veličina stabla	16
Vrsta funkcije prilagođenosti	adjusted
Skup funkcija	+, -, *, /, pow, sin, cos, log

Tabela 5.3: Vrednosti parametara genetskog programiranja za „Yacht Hydrodynamics" skup podataka

Parametar	Vrednost
Veličina populacije	2000
Maksimalan broj geneacija	100
Broj jedinki za reprodukciju	1200
Veličina turnira	10
Verovatnoća mutacie pojedinačnog čvora	0.2
Verovatnoća mutacije celog podstabla	0.2
Minimalna dubina stabla	2
Maksimalna dubina stabla u fazi inicijalizacije	12
Maksimalna veličina stabla	64
Vrsta funkcije prilagođenosti	adjusted
Skup funkcija	+, -, *, /, pow, sin, cos, log

U tabeli 5.4 se mogu videti korišćene vrednosti parametara za metodu promenljivih okolina na svim skupovima podataka, osim za „Yacht". Kod njega je za k_{max} uzeta vrednost 6, a broj iteracija je povećan na 2000.

Tabela 5.4: Vrednosti parametara metode promenljivih okolina

Parametar	Vrednost
k_{max}	4
Minimalna dubina stabla	2
Maksimalna dubina stabla u fazi inicijalizacije	4
Maksimalna dubina stabla kreiranog u fazi prerage	6
Maksimalan broj iteracija	1000
Skup funkcija	+, -, *, /, pow, sin, cos, log

Tabela 5.5: Prosečne vrednosti određenih karakteristika u 30 nezavisnih pokretanja

	Prosečna R^2 vrednost na trening skupu			**Prosečna R^2 vrednost na test skupu**			**Broj pokretanja u kojima je pronađeno rešenje simbolički ekvivalentno ciljnom rešenju**			**Prosečno vreme izvršavanja (s)**		
	GP	**GP sa SSC**	**VNP**	**GP**	**GP sa SSC**	**VNP**	**GP**	**GP sa SSC**	**VNP**	**GP**	**GP sa SSC**	**VNP**
F_{01}	0.879	0.831	**0.949**	0.898	0.861	**0.945**	7	3	**13**	12	19	**4**
F_{02}	0.765	0.802	**0.848**	0.791	0.818	**0.897**	1	3	**11**	7	13	**6**
F_{03}	0.745	0.733	**0.863**	0.810	0.820	**0.926**	5	5	**14**	6	12	**5**
F_{04}	0.733	0.740	**0.914**	0.820	0.775	**0.922**	2	1	**9**	12	18	**5**
F_{05}	0.795	0.771	**0.846**	0.797	0.765	**0.813**	3	1	**9**	**7**	18	**7**

Tabela 5.6: Prosečne vrednosti određenih karakteristika u 30 nezavisnih pokretanja

	Prosečna R^2 vrednost na trening skupu			**Prosečna R^2 vrednost na test skupu**			**Broj pokretanja u kojima je pronađeno rešenje simbolički ekvivalentno ciljnom rešenju**			**Prosečno vreme izvršavanja (s)**		
	GP	**GP sa SSC**	**VNP**	**GP**	**GP sa SSC**	**VNP**	**GP**	**GP sa SSC**	**VNP**	**GP**	**GP sa SSC**	**VNP**
F_1	**0.914**	0.861	0.907	**0.907**	0.827	0.872	0	0	**13**	15	24	**7**
F_2	**0.827**	0.799	0.771	**0.824**	0.798	0.770	0	0	**9**	13	26	**9**
F_3	**0.851**	0.851	0.797	0.695	-1.428	**0.752**	0	0	**2**	20	23	**10**
F_4	0.746	0.691	**0.809**	**0.796**	0.743	0.778	0	0	**2**	14	27	**12**
F_5	0.643	0.606	**0.894**	0.607	0.589	**0.887**	0	0	0	**14**	24	**14**
F_6	0.928	0.917	**0.945**	0.883	0.881	**0.930**	0	0	**1**	13	27	**12**
F_7	0.960	0.968	**0.994**	0.950	0.959	**0.993**	0	0	0	18	51	**13**
F_8	0.857	0.837	**0.968**	0.716	0.657	**0.936**	0	0	**3**	13	35	**7**
F_9	0.950	0.940	**0.963**	0.955	0.938	**0.971**	1	1	**2**	12	28	**8**

Tabela 5.7: Prosečne vrednosti određenih karakteristika u 30 nezavisnih pokretanja za „Yacht Hydrodynamics” skup podataka

Metoda	**Prosečna R^2 vrednost na trening skupu**	**Prosečna R^2 vrednost na test skupu**	**Prosečno vreme izvršavanja (s)**
GP	0.238	0.264	183
GP sa SSC	0.213	0.233	247
VNP	**0.477**	**0.457**	**30**

Tabela 5.8: Informacije o izrazu koji daje maksimalnu R^2 vrednost na test skupu od svih izraza dobijenih pri 30 nezavisnih pokretanja

	Maksimalna R^2 vrednost na test skupu			**Izraz koji ima maksimalnu R^2 vrednost na test skupu**			**Simbolička ekvivalencija sa ciljnim izrazom**		
	GP	**GP sa SSC**	**VNP**	**GP**	**GP sa SSC**	**VNP**	**GP**	**GP sa SSC**	**VNP**
F_{01}	1.0	1.0	1.0	$x_1(x_0+1)$	$x_1(x_0+1)$	$x_1(x_0+1)$	Da	Da	Da
F_{02}	1.0	1.0	1.0	$x_0+x_1^2+x_1$	$x_0+x_1^2+x_1$	$x_0+x_1^2+x_1$	Da	Da	Da
F_{03}	1.0	1.0	1.0	$x_0x_1+cos(x_0)$	$x_0x_1+cos(x_0)$	$x_0x_1+cos(x_0)$	Da	Da	Da
F_{04}	1.0	1.0	1.0	$x_0-x_1^2$	$x_0-x_1^2$	$x_0-x_1^2$	Da	Da	Da
F_{05}	1.0	1.0	1.0	$x_0-x_1^2+x_1$	$x_0-x_1^2+x_1$	$x_0-x_1^2+x_1$	Da	Da	Da

Tabela 5.9: Informacije o izrazu koji daje maksimalnu R^2 vrednost na test skupu od svih izraza dobijenih pri 30 nezavisnih pokretanja

	Maksimalna R^2 vrednost na test skupu			**Izraz koji ima maksimalnu R^2 vrednost na test skupu**			**Simbolička ekvivalencija sa ciljnim izrazom**		
	GP	**GP sa SSC**	**VNP**	**GP**	**GP sa SSC**	**VNP**	**GP**	**GP sa SSC**	**VNP**
F_1	0.992	0.985	**1.0**	$x_0(x_0+\frac{1}{cos(x_0)})$	$\frac{x_0(x_0+1)}{cos(x_0)^{cos(x_0)}}$	$x_0(x_0(x_0+1)+1)$	Ne	Ne	**Da**
F_2	0.994	0.981	**1.0**	$\frac{x_0(x_0+1)}{cos(sin(tan(x_0)))}$	$x_0(x0+1)$ $*(sin(x_0)+1)$	$x_0(x_0+1)(x_0^2+1)$	Ne	Ne	**Da**
F_3	0.981	0.995	**1.0**	$\frac{x_0}{(1.167cos(x_0))^{x_0}(\frac{x_0}{4.25x_0-sin(x_0)})^{x_0}}$	$\frac{x_0^3+x_0^2+sin(x_0)sin(cos(x_0))}{sin(cos(x_0))}$	$x_0(x_0^2(x_0(x_0+1)+1)$ $+x_0+1)$	Ne	Ne	**Da**
F_4	0.986	0.960	**1.0**	$2.74^{\frac{x_0}{cos(x_0))x_0}}$	$2.75^{sin(x_0)}$ $*x_0(x_0+1.08)$	$x_0(x_0^4(x_0+1)$ $+x_0^2(x_0+1)+x_0+1)$	Ne	Ne	**Da**
F_5	0.943	0.964	**0.999**	$\frac{-0.51}{sin(x_0^2+0.482)}$	$log(cos(cos(x_0)))$ -0.331	$-2cos(x_0)+cos(x_0^2)$	Ne	Ne	Ne
F_6	0.971	0.987	**1.0**	$\frac{x_0}{cos(cos(x_0-0.52))}$	$2.499x_0cos(x_0)$ $+ sin(x_0^2cos(x_0))$	$sin(x_0)$ $+sin(x_0(x_0+1))$	Ne	Ne	**Da**
F_7	0.997	0.998	**0.999**	$1.359x_0$	$1.38067x_0$	$x_0+sin(x_0)$ $- sin(sin(0.49x_0^{(1-x_0)}))$	Ne	Ne	Ne
F_8	0.999	0.994	**1.0**	$x_1sin(x_1)+sin(x_0)$	x_0x_1+ $(-x_0+x_1)sin(x_1)$ $+sin(x_0)$	$sin(x_0)+sin(x_1^2)$	Ne	Ne	**Da**
F_9	**1.0**	**1.0**	**1.0**	$2sin(x_0)cos(x_1)$	$2sin(x_0)cos(x_1)$	$2sin(x_0)cos(x_1)$	**Da**	**Da**	**Da**

Tabela 5.10: Informacije o izrazu koji daje maksimalnu R^2 vrednost na test skupu od svih izraza dobijenih pri 30 nezavisnih pokretanja za „Yacht Hydrodynamics" skup podataka

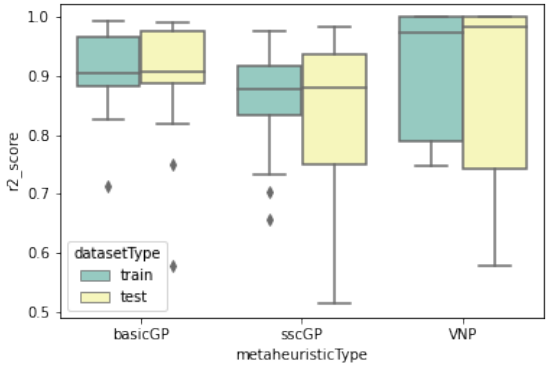
Metoda	**Maksimalna R^2 vrednost na test skupu**	**Izraz koji ima maksimalnu R^2 vrednost na test skupu**
GP	0.929	$1.906^{(3.963x_5)}$
GP sa SSC	0.792	$\frac{0.822}{0.052^{x_5}}$
VNP	**0.956**	$0.000196^{\frac{x_1-x_2x_5}{x_2}*\frac{x_5}{x_1-2x_5^2}}$

U tabelama 5.5 i 5.6 je za svaku metodu prikazana prosečna R^2 vrednost na trening i test skupu svake od funkcija, izračunata na osnovu 30 nezavisnih pokretanja. Takođe, dati su prosečno vreme izvršavanja metode (izraženo u sekundama) i broj pokretanja u kojima je najbolje pronađeno rešenje simbolički ekvivalentno ciljnom rešenju. Slično, tabela 5.7 prikazuje te informacije za „Yacht Hydrodynamics" skup podataka. Može se primetiti da VNP metoda u proseku daje najbolje rezultate, kao i da najveći broj puta dolazi do ciljnog izraza. Takođe, VNP metoda u proseku traje dosta kraće u odnosu na preostale dve metode.

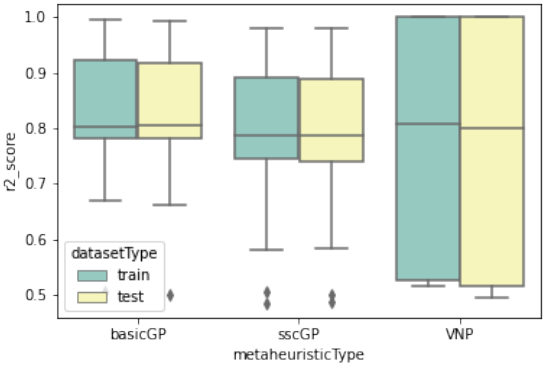
U tabelama 5.8 i 5.9 je za svaku metodu prikazana maksimalna R^2 vrednost na test skupu svake od funkcija, koja je dostignuta u nekom od 30 nezavisnih pokretanja metode. Takođe, prikazana je i funkcija kojom je postignuta ta vrednost, kao i informacija o tome da li je ona simbolički ekvivalentna ciljnom rešenju. Slično, tabela 5.10 prikazuje te informacije za „Yacht Hydrodynamics" skup podataka.

Na osnovu tabele 5.8 se može videti da su sve predložene metode pronašle izraze koji su ekvivalentni sa ciljnim izrazom. Takođe, za istu instancu, sve metode su vratile isti izraz. Na osnovu tabela 5.9 i 5.10 se može videti da metoda promenljivih okolina daje najbolje rezultate na težim problemima. Od 9 problema za koje nam je poznata ciljna promenljiva, VNP metoda je uspešno uspela da reši njih 7, dok su obe varijante genetskog programiranja uspele da pronađu ciljni izraz samo za jedan problem.

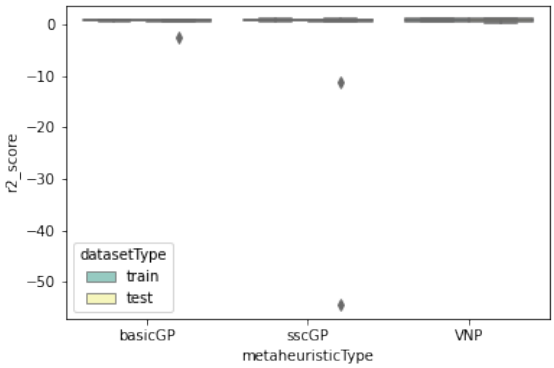
Na slikama 5.1 - 5.15 rezultati su predstavljeni i vizualno, pomoću *box-plot* dijagrama. Svaka slika odgovara rezultatima dobijenim nad jednim skupom podataka. Prika-



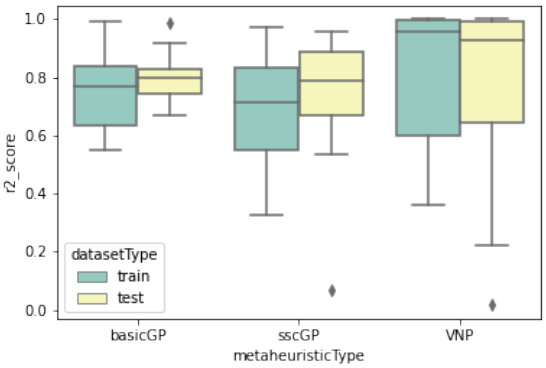
Slika 5.1: F_1



Slika 5.2: F_2



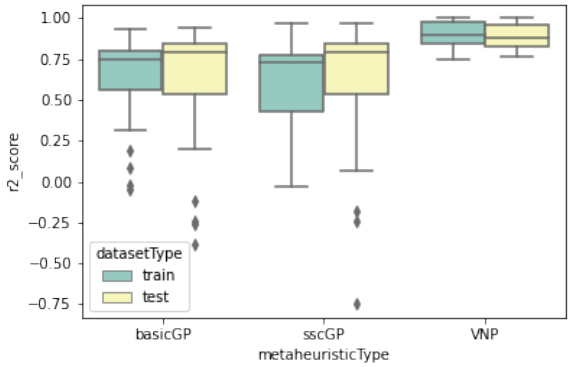
Slika 5.3: F_3



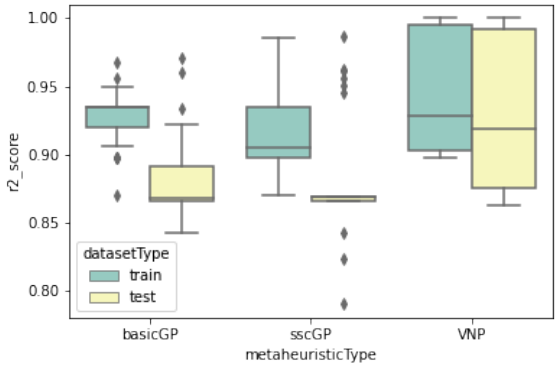
Slika 5.4: F_4

zane su R^2 vrednosti na trening i test skupu za svaku od metoda. I na osnovu ovih slika se može videti da pri većem broju pokretanja VNP metoda češće dolazi do kvalitetnijih rešenja. Obe varijante genetskog programiranja imaju veći broj odstupajućih ocena, u poređenju sa VNP metodom kod koje se ocene van granica *box-plot*-a javljaju samo u primerima $F7$ i $F8$.

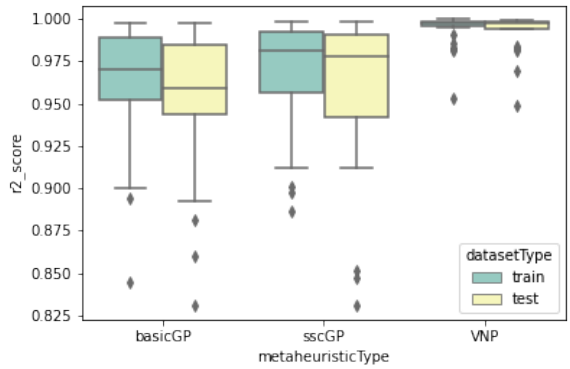
Na osnovu dobijenih rezultata se vidi da metoda promenljivih okolina radi značajno bolje od obe varijante genetskog programiranja, i u smislu preciznosti i u smislu brzine izvršavanja.



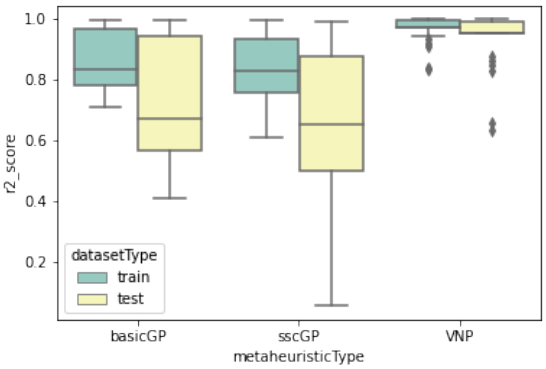
Slika 5.5: F_5



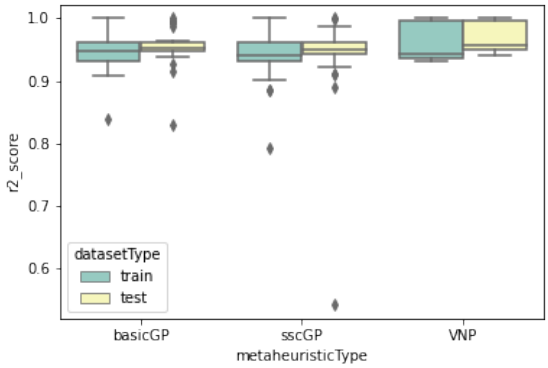
Slika 5.6: F_6



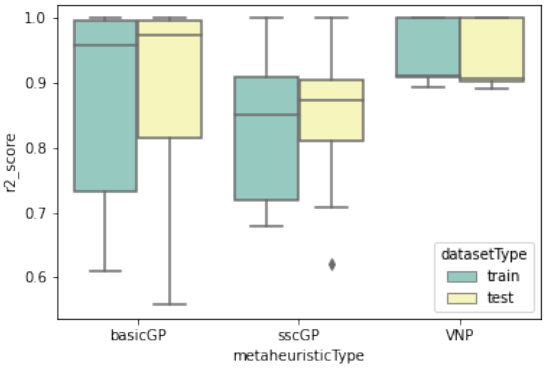
Slika 5.7: F_7



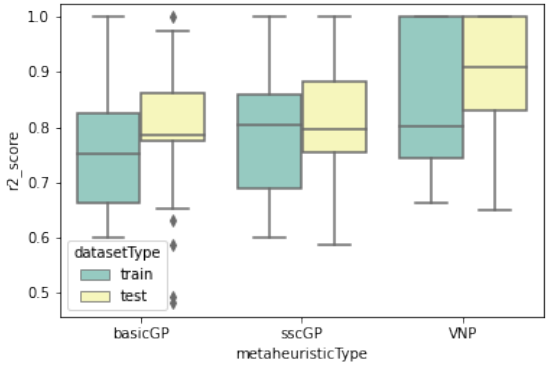
Slika 5.8: F_8



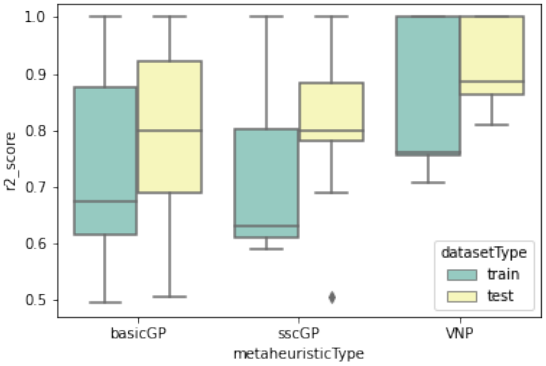
Slika 5.9: F_9



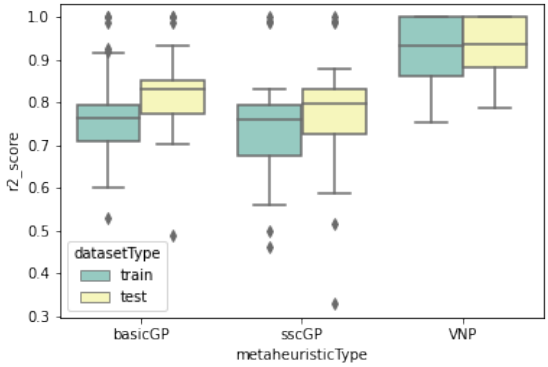
Slika 5.10: F_{01}



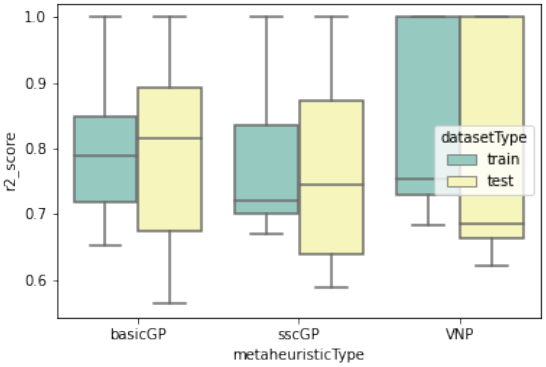
Slika 5.11: F_{02}



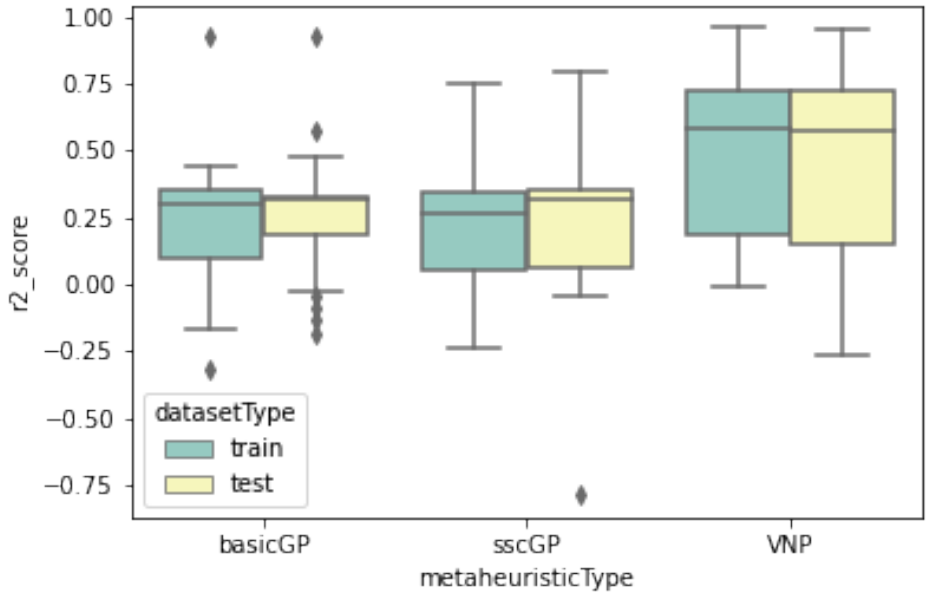
Slika 5.12: F_{03}



Slika 5.13: F_{04}



Slika 5.14: F_{05}



Slika 5.15: *Yacht Hydrodynamics*

Glava 6

Zaključak

Predmet istraživanja ovog rada je problem simboličke regresije. Na početku rada dat je detaljan opis problema i prikazan je princip evaluacije metoda kojima se rešava.

Primenom algoritma grube sile pokazano je da egzaktne metode nisu adekvatne za rešavanje ovog problema nad instancama velikih dimanzija zbog memorijskih i vremenskih ograničenja. U radu su implementirane i dve metaheurističke metode - metoda promenljivih okolina i genetsko programiranje sa dve vaijante ukrštanja jedinki. Analiza performansi heurističkih metoda je pokazala da je metoda promenljivih okolina uspešnija od obe varijante genetskog programiranja, i u smislu preciznosti i u smislu brzine izvršavanja.

Dalji rad vezan za ovu oblast se može odvijati u više pravaca, među kojima su:

- Implementacija drugih metaheurističkih metoda.
- Unapređenje metode promenljivih okolina različitim modifikacijama. Na primer, implementacijom tehnike *najbolje poboljšanje* umesto tehnike *prvo poboljšanje* u koraku lokalne pretrage. Ili bi mogle biti isprobane i varijante redukovane (RVNS, *Reduced VNS*) i iskrivljene (SVNS, *Skewed VNS*) metode promenljivih okolina.
- Hibrdizacija različitih metaheurističkih metoda.
- Hibridizacija neke metaheurističke metode i neke metode dubokog učenja. Kao što je npr. urađeno u radu [4] gde je kombinovano genetsko programiranje sa rekurentnim mrežama za učenje formiranja početne populacije.
- Implementacija metode koja je u potpunosti zasnovana na dubokom učenju.

Bibliografija

[1] Baligh Al-Helali. Genetic programming for symbolic regression on incomplete data. *PhD thesis, Victoria University of Wellington*, 2021. https://openaccess.wgtn.ac.nz/articles/thesis/Genetic_Programming_for_Symbolic_Regression_on_Incomplete_Data/17150609.

[2] Nurhan Karaboga Beyza Gorkemli Dervis Karaboga, Celal Ozturk. Artificial bee colony programming for symbolic regression. *Information Sciences*, 209:1–15, 2012.

[3] T. Nathan Mundhenk et al. Symbolic regression via neural-guided genetic programming population seeding. *Conference on Neural Information Processing Systems*, 35, 2021.

[4] T. Nathan Mundhenk et al. Symbolic regression via neural-guided genetic programming population seeding. *35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia*, 35, 2021.

[5] John Holland. Adaptation in natural and artificial systems. MIT Press, 1975.

[6] Sašo Džeroski Jure Brence, Ljupčo Todorovski. Probabilistic grammars for equation discovery. *Knowledge-Based Systems*, 224:107077, 2021.

[7] Michael Korns. Abstract expression grammar symbolic regression. *Genetic Programming Theory and Practice*, 8:109–128, 2010.

[8] John R. Koza. Genetic programming, on the programming of computers by means of natural selection. MIT Press, 1998.

[9] Peter A.N. Bosman Marco Virgolin, Tanja Alderliesten. Linear scaling with and within semantic backpropagation-based genetic programming for symbolic regression. *The Genetic and Evolutionary Computation Conference (GECCO)*, GECCO'19:1084–1092, 2019.

[10] Enrique Naredo Conor Ryan Muhammad Sarmad Ali, Meghana Kshirsagar. Towards automatic grammatical evolution for real-world symbolic regression. *13th International Conference on Evolutionary Computation Theory and Applications*, 13:68–78, 2021.

[11] N. X. Hoai N. Q. Uy and R. I. McKay. An analysis of semantic aware crossover. *Communications in Computer and Information Science*, 51, 2009.

[12] N. X. Hoai N. Q. Uy and M. O'Neill. Semantic aware crossover for genetic programming: the case for real-valued function regression. *Proceedings of EuroGP09*, 12:292–302, 2009.

[13] Jack Brimberg José A. Moreno Pérez Nenad Mladenovic, Pierre Hansen. Variable neighborhood search. *Handbook of Metaheuristics*, 3:57–97, 2019.

[14] Pierre Hansen Nenad Mladenovic. Variable neighborhood search. *Computers and Operations Research*, 24:1097–1100, 1997.

[15] Jason H. Moore Patrick Orzechowski, William La Cava. Where are we now? a large benchmark study of recent symbolic regression methods. *The Genetic and Evolutionary Computation Conference (GECCO)*, GECCO'18:1183–1190, 2018.

[16] Mladen Nikolič Predrag Janičić. Veštačka inteligencija. 2018. `http://ml.matf.bg.ac.rs/readings/ml.pdf`.

[17] Jun Ren Qiang Lu and Zhiguang Wang. Using genetic programming with prior formula knowledge to solve symbolic regression problem. *Computational Intelligence and Neuroscience*, 2:1–17, 2016.

[18] Robert I. McKay Quang Uy Nguyen, Nguyen Xuan Hoai. Semantically-based crossover in genetic programming: Application to real-valued symbolic regression. *Genetic Programming and Evolvable Machines*, 12:91–119, 2011.

[19] UCI Machine Learning Repository. Yacht Hydrodynamics Data Set. on-line at: `https://archive.ics.uci.edu/ml/datasets/yacht+hydrodynamics`.

[20] Max Tegmark Silviu-Marian Udrescu. Ai feynman: a physics-inspired method for symbolic regression. *Science Advances*, 6:eaay2631, 2020.

[21] Nenad Mladenovic Jun Pei Souhir Elleuch, Bassem Jarboui. Variable neighborhood programming for symbolic regression. *Optimization Letters*, 16:191–210, 2020.

[22] SymPy Development Team. SymPy, 2021. on-line at: `https://www.sympy.org/en/index.html`, `https://pypi.org/project/sympy/`.

[23] Ziprian Zavoianu. Towards solution parsimony in an enhanced genetic programming process. *Thesis for MSc, Robert Gordon University, Aberdeen, UK*, 2010. `https://www.researchgate.net/publication/239917588_Towards_solution_parsimony_in_an_enhanced_genetic_programming_process`.