SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1041

Kratkoročno predviđanje potrošnje električne energije

Marko Deak

Zagreb, lipanj 2015.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 8. ožujka 2015.

Predmet: **Analiza i projektiranje računalom**

DIPLOMSKI ZADATAK br. 1041

Pristupnik: **Marko Deak (0036456921)**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **Kratkoročno predviđanje potrošnje električne energije**

Opis zadatka:

Opisati problem kratkoročnog predviđanja vremenskih slijedova na primjeru predviđanja potrošnje električne energije. Posebnu pažnju posvetiti modelima zasnovanim na genetskom programiranju. Prilagoditi i primijeniti algoritme skupnog učenja kao nadogradnju genetskog programiranja i ocijeniti vremensku učinkovitost i poboljšanje rezultata dobiveno takvim rješenjima. Istražiti mogućnost kombiniranja algoritama za odabir značajki s postojećim modelima. Ispitati inačice i učinkovitost predviđanja korištenjem genetskog programiranja na postojećim ispitnim skupovima. Usporediti učinkovitost ostvarenih postupaka s postojećim metodama strojnog učenja i genetskog programiranja. Radu priložiti izvorne tekstove programa, dobivene rezultate uz dokumentaciju i korištenu literaturu.

Zadatak uručen pristupniku: 13. ožujka 2015.
Rok za predaju rada: 30. lipnja 2015.

Mentor:

Izv. prof. dr. sc. Domagoj Jakobović

Djelovođa:

Doc. dr. sc. Tomislav Hrkać

Predsjednik odbora za diplomski rad profila:

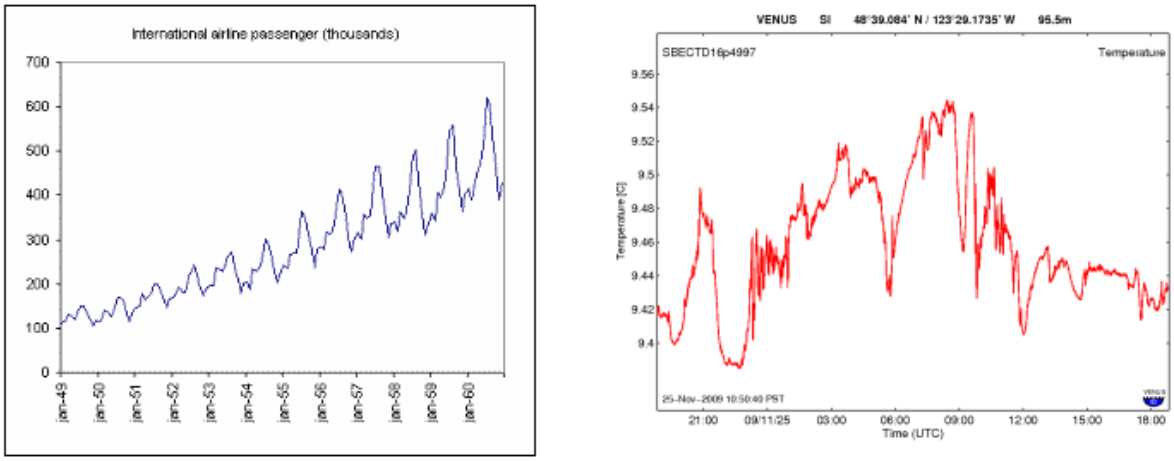
Prof. dr. sc. Siniša Srbljić

SADRŽAJ

POPIS SLIKA

1. Uvod

Vremenski sljedovi (engl. *time series*) su sljedovi podatkovnih točaka, najčešće dobivenih uzastopnim mjerenjima preko određenog vremenskog intervala. Velik postotak podataka iz stvarnog svijeta može se predstaviti vremenskim sljedovima, kao što su temperatura zraka, potrošnja električne energije, kretanje cijena dionica i slično.Primjeri dvaju vremenskih sljedova, točnije kretanja broja putnika na zrakoplovnim linijama te temperature kroz vrijeme, prikazani su na slici 1.1. Kod mnogih vremenskih sljedova stvara se potreba za analizom i posebno predviđanjem njihovih vrijednosti (kao kod npr. temperature zraka ili potrošnje električne energije). Analiza se bavi metodama analize kojima se iz vremenskih sljedova mogu izvući potrebne i smislene statistike te potencijalno uočiti karakteristike podataka, dok se prognoza bavi izradom modela koji je sposoban predvidjeti buduće vrijednosti na osnovu prijašnjih promatranih vrijednosti.



Slika 1.1: Primjeri vremenskih sljedova

Prognoziranje vremenskih sljedova (engl. *time series forecasitng*) u osnovi je blisko problemima regresije dobro poznatih iz strojnog učenja, uz bitnu razliku da se u predviđanju buduće vrijednosti moraju uzimati u obzir prijašnje vrijednosti izlaza

istog vremenskog slijeda. Najčešće se za ovo koriste metode koje se zasnivaju na statističkim modelima za stohastičku simulaciju, no u novije vrijeme koriste se metode s korijenima u strojnom učenju, poput regresije potpornim vektorima (engl. *support vector regression*) ili neuronskim mrežama.

Jedna od ne tako često korištenih metoda u predviđanju vrijednosti vremenskih sljedova je genetsko programiranje (engl. *genetic programming*), metaheuristička metoda optimizacije koja spada u skupinu evolucijskih algoritama (engl. *evolutionary algorithms*), algoritama koji traže rješenja optimizacijskih i sličnih problema koristeći tehnike inspirirane prirodnom evolucijom. Najčešće se ove metode koriste za rješavanje problema simboličke regresije, koji se može smatrati ekvivalentnim problemu predviđanja vrijednosti vremenskih sljedova.

U ovom radu demonstrira se primjena genetskog programiranja na jedan od primjera problema predviđanja vrijednosti vremenskih sljedova, a to je potrošnja električne energije. Predlažu se poboljšanja pristupa korištenjem genetskog programiranja te se rezultati prikazuju na način da budu usporedivi s polaznim radovima [1].

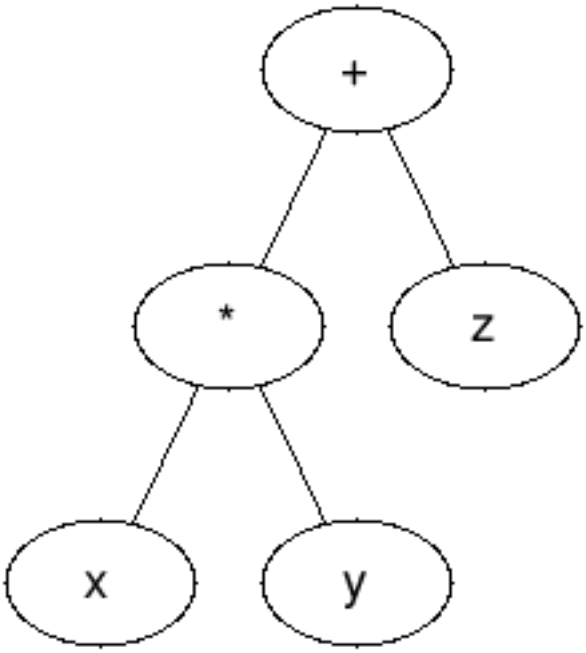
U drugom i trećem poglavlju daje se kratak pregled metoda genetskog programiranja te opis problema predviđanja potrošnje električne energije. U četvrtom poglavlju okvirno je opisana programska implementacija modela koji se koristi, dok se u petom raspravlja o optimizacijama koje su implementirane. U šestom poglavlju dani su rezultati u ovisnosti o korištenim varijantama algoritma, te se kroz raspravu i zaključak daje smisao dobivenim rezultatima i predlažu potencijalna daljnja poboljšanja.

2. Genetsko programiranje

Genetsko programiranje (GP) metaheuristička je metoda optimizacije koja se može smatrati specijalnim slučajem genetskog algoritma (GA) gdje su jedinke koje se evoluiraju programi. Metoda je detaljno opisana u [9], ali ono što je bitno je da je zadržana osnovna ideja GA - imitira se prirodni proces evolucije tako da se od početnog (nasumično ili heuristički generiranog) skupa rješenja kroz određen broj iteracija nastoji dobiti jedno rješenje koje dovoljno dobro odgovara danom problemu. Zahvaljujući tome što kroz genetske operatore imitira prirodnu evoluciju, koja podrazumijeva velik broj jedinki koji evoluira kroz vrijeme uz određenu dozu nasumičnosti, genetsko programiranje pretražuje velik broj rješenja unutar područja mogućih rješenja, te ima ugrađene mehanizme izbjegavanja konvergencije u lokalnim optimumima. Kao i ostale metaheurističke metode, ni GP ne može garantirati da će nađeno rješenje biti optimum, niti blizu globalnog optimuma, no u prosjeku daje prihvatljiva rješenja dovoljno blizu globalnom optimumu, a izbjegava zapinjanje u lokalnim optimumima kakvo se događa algoritmima lokalne pretrage.

2.1. Prikaz rješenja

U genetskom programiranju su po definiciji jedinke programi koji se mogu prikazivati na različite načine, ali najčešće su u memoriji prikazani kao stablaste strukture slične onima koje koriste Lisp ili funkcijski programski jezici. Kod ovog prikaza svaki čvor stabla predstavlja operatorsku funkciju kojom su listovi li podstabla operandi. Najveća prednost ovakvog prikaza jedinki je što ih je lako evoluirati, kao što je opisano dalje u ovom poglavlju, a vrijednost tako evoluirane jedinke efikasno je rekurzivno evaluirati. Ovaj prikaz također čini genetsko programiranje dobrim odabirom za rješavanje problema simboličke regresije budući da se ovakvim stablom može opisati gotovo svaka funkcija. Jedna ovakva jedinka prikazana je na slici 2.1.



Slika 2.1: Stablo koje evaluira funkciju $(x * y) + z$

Različiti drugi prikazi jedinki su mogući, pa tako postoje implementacije linearnog genetskog programiranja koje je bliže današnjim imperativnim programskim jezicima, zatim rješenja koja evoluiraju usmjerene grafove i mnogi drugi. Ipak, u većini radova prikaz jedinki u obliku stabla dovoljno je prikladan za rješavanje problema zadanih u radovima, a najjednostavniji je za implementirati.

2.2. Postupak selekcije

Jedna od glavnih značajki evolucije u prirodi je izumiranje neprilagođenih jedinki i opstanak prilagođenih. Tako je i kod GP za nalaženje dobrog rješenja uvjet da lošije jedinke, odnosno rješenja, nestaju iz populacije, dok bolja ostaju kako bi se iz njih mogla stvarati nova (dobra) rješenja. Kako bi se imitirao prirodni proces evolucije gdje preživljavaju one jedinke koje su prilagođene okolišu, definira se za svaku jedinku vrijednost dobrote (engl. *fintess*) koja opisuje koliko je rješenje koje ta jedinka predstavlja kvalitetno. Dobrota se može izračunavati na razne načine ovisno o problemu i algoritmu, no generalno veća vrijednost dobrote je pridijeljena boljoj jedinci te se na vrijednosti dobrote temelji postupak selekcije. Za odabir jedinki koje se uklanjaju iz populacije i/ili jedinki koje će sudjelovati u stvaranju novih jedinki koriste se postupci selekcije. Najčešći selekcijski mehanizmi su generacijski jednostavni odabir (engl. *roulette wheel*), eliminacijski jednostavni odabir i turnirski odabir.

Generacijski i eliminacijski jednostavni odabir funkcioniraju slično u smislu da se skalirane vrijednosti dobrota kod generacijskog i kazne (ili bilo koje mjere inverzno proporcionalne s dobrom jedinke) kod eliminacijskog jednostavnog odabira skaliraju i poslažu na brojevni pravac. Nakon ovoga se slučajnim odabirom biraju točke na pravcu te kod generacijskog jednostavnog odabira jedinke u čijem segmentu pravca se nalaze odabrane točke ulaze u iduću generaciju, a kod eliminacijskog jednostavnog odabira se te jedinke eliminiraju iz populacije. Kod turnirskog odabira stvaraju se turniri (podgrupe) od k jedinki koje se uspoređuju po dobroti, te se određen broj najgorih jedinki unutar podgrupe eliminira iz populacije te se mijenjaju novim jedinkama dobivenim primjenom genetskih operatora, najčešće na najboljim jedinkama unutar turnira. Najčešća i najjednostavnija turnirska selekcija je 3-turnirska selekcija ($k = 3$), no k može biti bilo koji pozitivan cijeli broj, iako prevelik k dovodi do previsokog selekcijskog pritiska, gdje će se za stvaranje novih jedinki uglavnom razmatrati samo najbolje jedinke unutar populacije čime se riskira konvergencija algoritma u lokalnom optimumu.

Važno je ovdje napomenuti elitizam, odnosno svojstvo genetskog algoritma (te implicitno i genetskog programiranja) da se određen dio najboljih jedinki u populaciji uvijek prenosi u novu populaciju prilikom izmjene generacija. Spomenuta turnirska selekcija implicitno ima ovo svojstvo (budući da je nemoguće da najbolje jedinke budu eliminirane turnirom), no generacijski jednostavni odabir te eliminacijski jednostavni odabir to nemaju, te se mora eksplicitno implementirati pri implementaciji selekcijskog mehanizma. Elitizam nije nužan u evolucijskim algoritmima, no uobičajena je praksa barem najbolju jedinku uvijek prenositi u iduću generaciju kako bi se sačuvalo najbolje rješenje algoritma do trenutne iteracije.

2.3. Genetski operatori

Najbitniji dio genetskog programiranja genetski su operatori križanja i mutacije. Oni omogućavaju stvaranje novih jedinki iz postojećih te nasumičnu promjenu postojećih jedinki i time omogućavaju pretraživanje šireg prostora rješenja, ali i intenzifikaciju pretraživanja oko dobrih rješenja. Za svaki prikaz rješenja zasebno se mogu definirati raznovrsni genetski operatori, no općenito se oni uvijek svode na operatore mutacije i operatore selekcije.

2.3.1. Operatori mutacije

Operatori mutacije unarni su operatori koji kao i kod genetskog algoritma služe za diverzifikaciju rješenja, odnosno omogućavaju da algoritam izađe iz potencijalnog lokalnog optimuma i pretraži što veći prostor rješenja. Sama implementacija operatora mutacije može biti raznolika, no u osnovi se svodi na mijenjanje jednog dijela kromosoma nasumično generiranom vrijednošću s određenom vjerojatnošću. Kao i točan princip rada operatora, vjerojatnost mutacije također ovisi o problemu, algoritmu i operatoru mutacije korištenom pa se najčešće može podesiti na različitu vrijednost pri svakom pokretanju algoritma. Generalno vrijedi da preniska vjerojatnost mutacije najčešće vodi do konvergencije u lokalni optimum (gdje populacija postane zasićena gotovo jednakim jedinkama), dok previsoka vjerojatnost pretvara genetski algoritam u slijepo (nasumično) pretraživanje. Zato je to jedan od parametara koji se podešava nakon više pokretanja, koristeći unakrsnu provjeru (engl. *cross validation*) ili slične metode.

Ključna razlika implementacija operatora mutacije u genetskom programiranju u odnosu na općenite genetske algoritme je što operatori mutacije moraju očuvati integritet stabla. Budući da operator mutacije može zamijeniti cijeli čvor jedinke ili podatke unutar tog čvora, mora znati rukovati sa različitim tipovima vrijednosti i razlikovati između čvorova koji predstavljaju npr. binarne i unarne operacije kako bi se nakon mutacije program koji predstavlja jedinka mogao i dalje izvoditi.

2.3.2. Operatori križanja

Operatori križanja (engl. *crossover*) binarni su operatori i za razliku od operatora mutacije služe za intenzifikaciju rješenja, tj. detaljnije pretraživanje prostora rješenja oko najboljih jedinki. Na osnovu 2 jedinke iz populacije, koje se u kontekstu križanja nazivaju "roditeljima", stvaraju jednu ili više novih jedinki (u ovom kontekstu se jedna takva jedinka naziva "dijete") rekombinacijom elemenata obaju roditelja. Budući da opisani operator selekcije iz populacije izbacuje loše jedinke, najčešće se operatori križanja koriste nakon selekcije kako bi se stvorile jedinke koje će zauzeti mjesto izbačenih jedinki u populaciji. Budući da rade rekombinaciju elemenata jedinki, implementacijski su najčešće uže vezani uz prikaz rješenja koji se koristi nego operatori mutacije. Važno je uočiti da nije nužno slučaj da dijete dvaju dobrih jedinki bude dobra jedinka, već je poanta da je u prostoru rješenja nova jedinka bliska roditeljima što povećava šansu da će biti dovoljno dobra, a u isto vrijeme unositi raznolikost u po-

pulaciju. Prevelika učestalost rekombinacije gena i previsok selekcijski pritisak mogu uzrokovati preranu konvergenciju algoritma, no ovaj problem se u implementaciji rješava prilikom definiranja operatora selekcije.

2.4. Općeniti pseudokod genetskog algoritma

Iako se genetski algoritmi mogu modificirati korištenjem raznolikih operatora selekcije, mutacije i križanja, općenito se pseudokod može svesti na onaj prikazan u algoritmu 1.

Uvjet zaustavljanja algoritma može biti raznolik, a neki od najčešće korištenih su broj izmjena generacija, broj evaluacija funkcije dobrote, stagnacija najboljeg rješenja te mnogi drugi, ovisno o ograničenjima problema. Česti odmak od općenitog pseudokoda radi optimizacije vremena izvođenja je evaluiranje funkcije dobrote samo jednom za svaku funkciju, odnosno samo kad je potrebno, kako bi se uštedjelo vrijeme potrošeno na nepotrebne evaluacije kad se dobrota jedinke ne mijenja.

Algoritam 1 Općeniti pseudokod genetskog algoritma

```
brojGeneracija ← 0
populacija ← generirajPocetnuPopulaciju()
while uvjetZaustavljanjaNijeIspunjen() do
    for i = 1 → brojJedinki do
        jedinka_i.dobrota ← izracunajDobrotu()
    end for
    populacija ← provediSelekciju(populacija)
    provediKrizanje(populacija)
    provediMutaciju(populacija)
    brojGeneracija = brojGeneracija + 1
end while
```

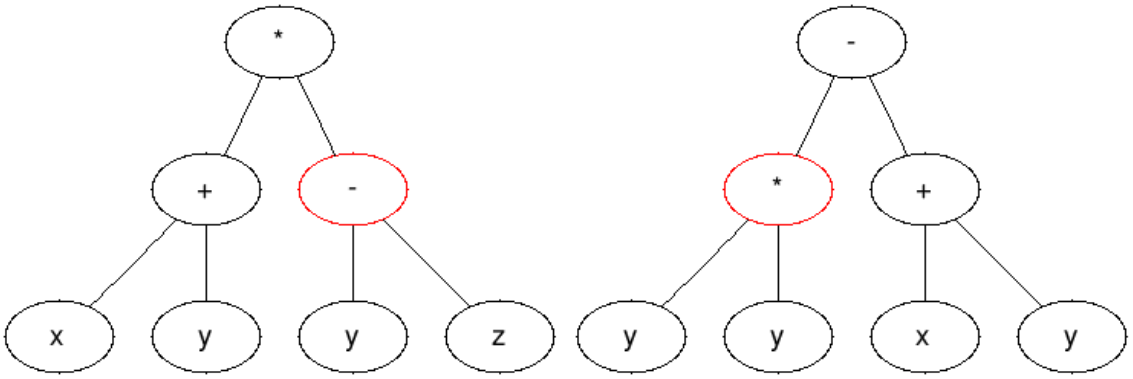
2.5. Skup genetskih operatora korišten u radu

Kao što je razmatrano u [8], u genetskom programiranju nisu svi operatori križanja jednako učinkoviti na svim problemima. U sklopu ovog rada zato je implementirano više operatora kako bi bila ispitane i uspoređenje njihove učinkovitosti u evoluciji mo-

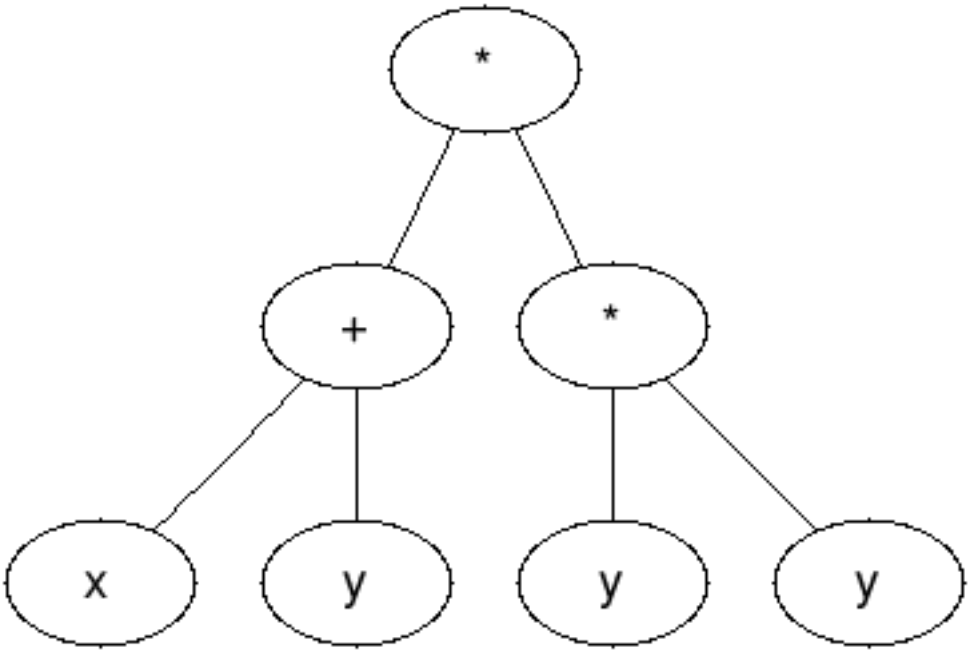
dela za ovaj problem. Svaki od njih kratko je ovdje opisan i ilustriran kako bi se shvatio njihov princip rada, bolje opisan u [8].

Jednostavno križanje

Genetski operator jednostavnog križanja bira na svakom od dva stabla roditelja potpuno nasumično kromosome, odnosno čvorove stabla, na kojima se događa križanje. Odabrani čvorovi su korijeni podstabala koja se zamijene unutar dva stabla čime nastaju dva moguća nova stabla. Ovaj princip zamjene prikazan je na slici 2.2, a većinom ga slijede i ostali operatori križanja, osim što oni za odabir kromosoma na kojima se događa križanje uvode dodatne uvjete. Pri svakom ovakvom križanju moguće je nastajanje dva stabla djeteta, a budući da je najčešće potrebno samo jedno, nasumično se bira ono koje će biti dodano u populaciju.



(a) Stabla roditelji, s crveno označenim čvorovima odabranim za točke križanja



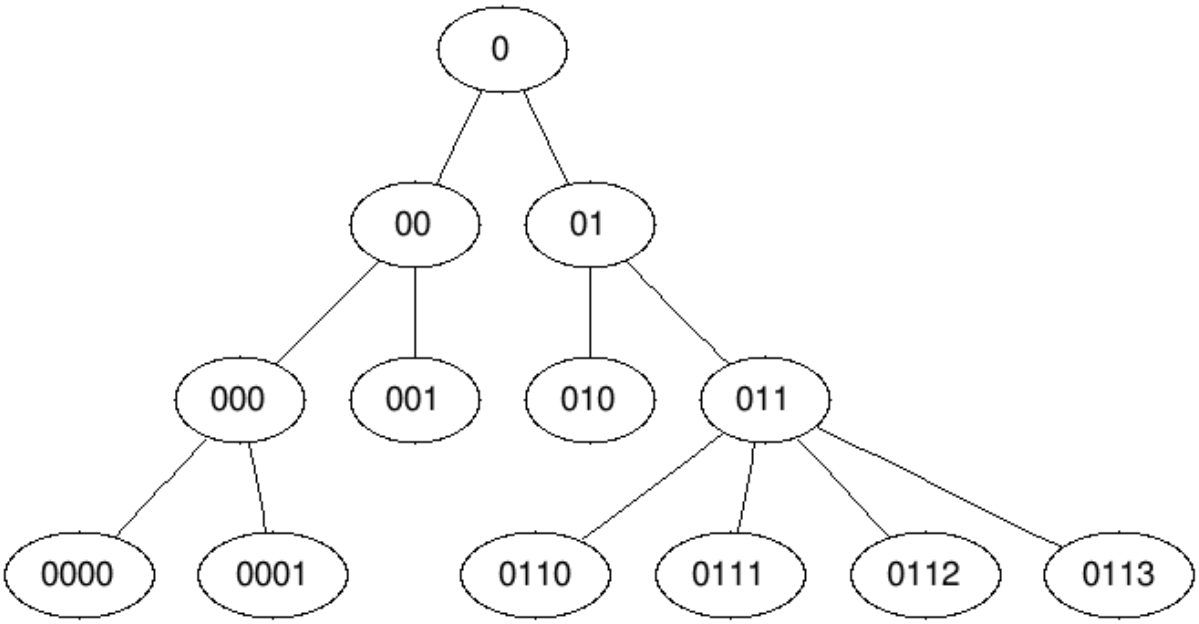
(b) Moguće stablo dijete

Slika 2.2: Jednostavno križanje, gore su dva stabla roditelja, a dolje je prikazano jedno od dva moguća stabla djeteta

Križanje s očuvanjem konteksta

Ovaj operator u stablo uvodi dodatnu informaciju koja predstavlja koordinatu svakog čvora. Koordinata čvora jedinstven je identifikator puta kojim se od korijena dolazi do određenog čvora. Na slici 2.3 prikazane su koordinate čvorova u stablu uzetom za primjer. Po konvenciji uzetoj u ovom radu, korijen stabla uvijek ima koordinatu 0 (iako bi mogao imati jednostavno praznu koordinatu kao u nekim radovima), a koordinata svakog idućeg čvora računa se konkatenacijom koordinate čvora roditelja sa indeksom čvora u listi djece čvora roditelja. Tako će prvi čvor dijete korijena imati koordinatu

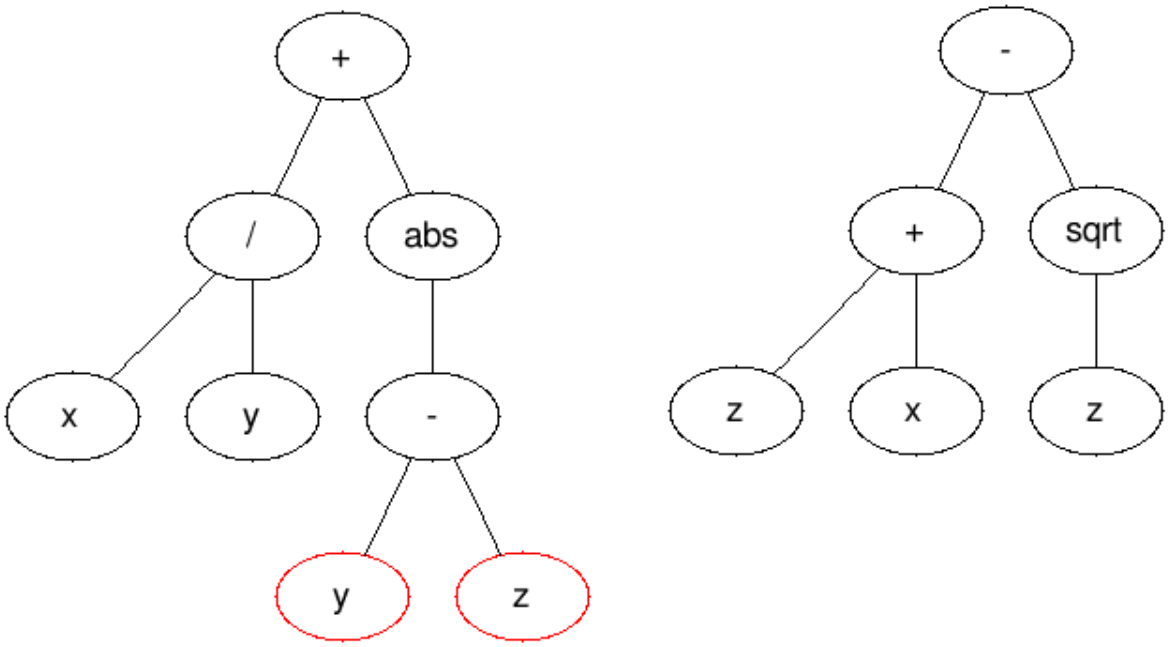
00, a drugo dijete 01.



Slika 2.3: Koordinate čvorova za križanje s očuvanjem konteksta

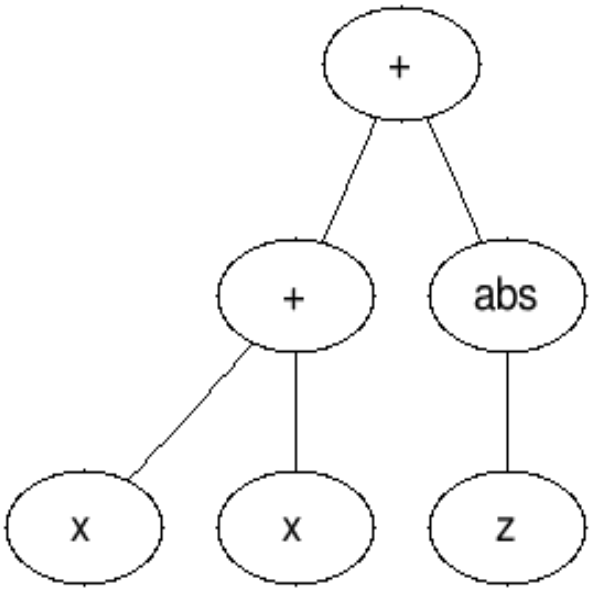
Uniformno križanje

Zadnji implementirani operator križanja operator je uniformnog križanja. Operator imitira uniformno križanje iz linearnih struktura genetskog algoritma, u kojem se na svakoj poziciji nasumično odabire gen jednog od roditelja koji će biti prisutan u djetetu na toj poziciji. Ipak, kod genetskog programiranja prisutan je problem različitog oblika jedinki pa je potrebno pronaći zajedničko područje dva roditelja, odnosno sve čvorove koji dijele pozicije u oba stabla. Zajedničko područje prikazano je na slici 2.4, a implementacijski se ovdje koriste informacije o koordinatama čvora koje su već implementirane kod križanja s očuvanjem konteksta.



Slika 2.4: Dva stabla roditelja za uniformno križanje. Čvorovi izvan zajedničkog područja označeni su crvenom bojom.

Na slici 2.5 prikazano je moguće stablo dijete nastalo križanjem stabala prikazanih na slici 2.4. Svaki od čvorova unutar zajedničkog područja nasumično je odabran iz jednog roditelja, dok se za čvorove izvan zajedničkog područja i njihove izravne roditelje pri križanju mora paziti da broj djece čvorova odgovara definiciji operacije koju čvor opisuje.



Slika 2.5: Moguće dijete nastalo uniformnim križanjem roditelja prikazanih na prijašnjoj slici.

Operator mutacije

Za razliku od operatora križanja, za mutaciju je korišten samo najjednostavniji operator mutacije koji sa vjerojatnošću p mijenja jedan nasumičan čvor u stablu čvorom koji prima jednaki broj operanada (što znači da će binarni biti zamijenjeni binarnim, unarni unarnima, a čvorovi koji predstavljaju simbol i mogu biti samo listovi će zamijeniti druge simbole). Iznimka principu rada ovog operatora su čvorovi koji predstavljaju operatore grananja, za koje je operator mutacije zasebno definiran i opisan prije u ovom radu. Ovakav jednostavni operator mutacije ipak unosi dovoljnu razinu nasumičnosti i diverzifikacije rješenja, a ne postoji potreba za provjeravanjem konteksta ili slično.

3. Predviđanje potrošnje električne energije

Predviđanje potrošnje električne energije jedan je od osnovnih primjera problema predviđanja vrijednosti vremenskih sljedova. Podaci za ovaj problem lako se mogu nabaviti u velikim količinama uz suradnju operatera distribucijskih sustava, a interes za što preciznijim predviđanjem postoji kod svih sudionika na tržištu električne energije (krajnjih korisnika, distributera i proizvođača) pa je svaki napredak u ovom području bitan. Kako je skladištenje električne energije prilično skupo u usporedbi sa cijenom proizvodnje, od najranijih dana proizvodnje električne energije pazilo se na balansiranje opterećenja električnih postrojenja, odnosno proizvodnje i potrošnje. U slučaju nedovoljne proizvodnje riskiraju se nestanci struje koji mogu oštetiti postrojenja i uzrokovati značajnu financijsku štetu, dok se kod prevelike proizvodnje nepotrebno troši novac budući da se tolike količine električne energije ne mogu skladištiti.

Zbog različitih potreba preciznosti i brzine predviđanja, korišteni pristupi razlikuju se jedan od drugog ovisno o vremenskom horizontu predviđanja. Tako se razlikuju vrlo kratkoročno predviđanje (horizont manji od jednog sata, uglavnom za fine optimizacije opterećenja u realnom vremenu), kratkoročno previđanje (manje od 30 dana unaprijed, najčešće korišten dnevno sa granulacijom predviđanja potrošnje po satu), srednjoročno predviđanje (do godinu dana, koriste ga kupci na tržištu električne energije, često se koristi za određivanje cijene energije na tržištu) te dugoročno planiranje opterećenja (osnova za planiranje energetskog sustava na razini cijelog društva). [10]

3.1. Ulazni podaci

Na potrošnju električne energije utječu razni faktori vezani uz krajnje korisnike i njihovo okruženje. Kroz povijest predviđanja električne energije identificirani su ključni faktori koji imaju najviše utjecaja na uzorak opterećenja električne mreže. Ovi

faktori mogu se podijeliti u tri ključne skupine, a to su vremenski, ekonomski i društveni faktori. Vremenski faktori najčešće su korišteni, a odnose se na vrijednosti poput temperature i vlažnosti zraka, vremenskih uvjeta, brzine vjetra i slično. Najbitniji od ovih najčešće je temperatura zraka budući da ima najveći utjecaj na ljude i njihove navike korištenja električne energije. Ekonomski faktori najbolje prikazuju tržišno okruženje i gospodarsku aktivnost, a primjeri su cijena nafte, BDP države, cijena energenata i slično. Zadnja skupina su društveni faktor, koji mogu biti kalendarski (praznici i slično), društvene strukture i razne druge mjere društvenih događanja. [11]

Unatoč raznim spomenutim faktorima koji mogu utjecati na potrošnju, ulazni podaci za predviđanje potrošnje električne energije moraju biti mjerljive veličine, koje također ovise o tipu, odnosno vremenskom horizontu, predviđanja koje se koristi. U podatcima korištenim u ovom radu, vremenski faktori zastupljeni su u obliku temperature i vlažnosti zraka, koji se smatraju dovoljnima da predstavljaju trenutne vremenske uvjete i klimu područja, a društveni u obliku cjelobrojnih vrijednosti kao što su dan u mjesecu, mjesec u godini te oznaka da li su trenutni, sljedeći i/ili prethodni dan praznici. Nedostaju podatci koji bi predstavljali ekonomske faktore poput cijene energenata i slično. Naravno, dodatno se kao ulazni podatci koriste prijašnje izmjerene vrijednosti potrošnje električne energije, i to aritmetička sredina zadnjih N mjerenja kao i zadnja izmjerena vrijednost za sat u danu za koji se predviđa vrijednost. Ipak, kako se ovaj rad fokusira na kratkoročno predviđanje potrošnje, smatra se da je ovaj podatkovni skup dovoljno dobar za dobivanje zadovoljavajućih rezultata (ili barem usporedivih sa postojećim rezultatima u području).

3.2. Metode predviđanja potrošnje

Kao što je spomenuto, predviđanje električne energije jedan je od problema koji je poznat gotovo jednako dugo kao što je električna energija korištena kao glavni energent u kućanstvima diljem svijeta. Tako su se na početku koristile statističke metode kojima se pokušavalo opisati stohastičko kretanje potrošnje raznim distribucijama. U novije vrijeme sve više radova koristi metode proizašle iz sfere strojnog učenja, kao što su skupovi neuronski mreža [3], regresija potpornim vektorima [1] i slične. Popularnost ovih metoda proizlazi iz rastuće popularnosti strojnog učenja i konstantnog napretka u području koje rezultira nalaženjem novih metoda i poboljšanja za postojeće metode. Samim poboljšanjem korištenih metoda, preciznost modela za predviđanje potrošnje konstantno se poboljšava, iako je mjesta za poboljšanje sve manje, a potreba

za boljim rezultatima je stalno prisutna na tržištu. Kako poboljšanja postaju sve manja, otvara se pitanje hoće li metode preuzete iz strojnog učenja doći do točke gdje će daljnji napredak biti teško moguć. Iz ovoga proizlazi mogućnost razvoja novih metoda temeljenih na evolucijskim algoritmima ili korištenja hibridnih metoda gdje bi se klasični modeli strojnog učenja učili ili optimizirali korištenjem evolucijskih ili sličnih metaheurističkih metoda umjesto klasičnim metodama nadziranog učenja (engl. *supervised learning*).

4. Predviđanje vrijednosti vremenskih sljedova uporabom genetskog programiranja

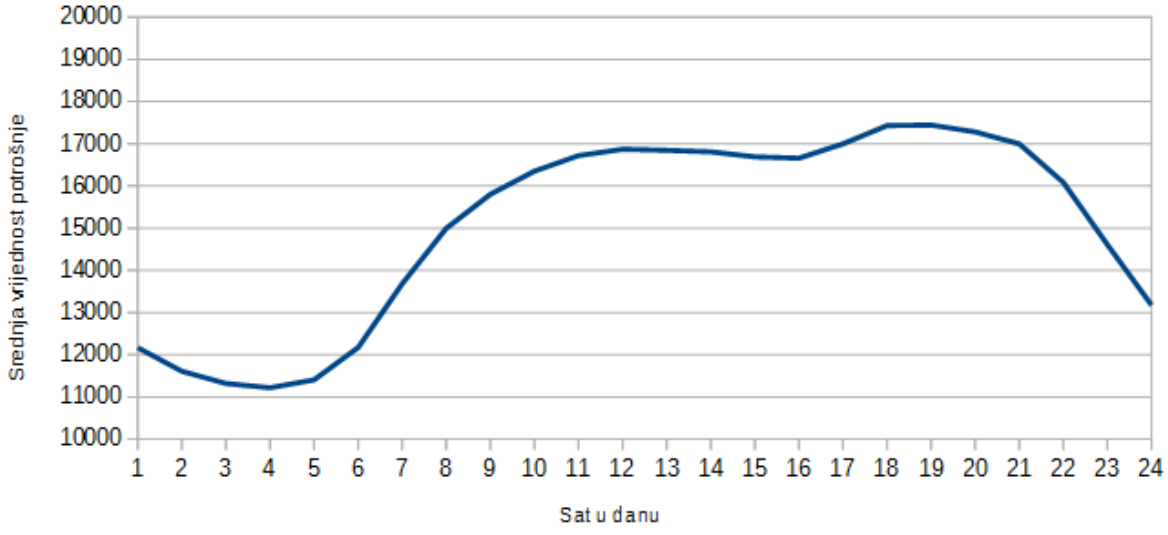
U okviru ovog rada implementirano je kratkoročno predviđanje potrošnje električne energije uporabom genetskog programiranja, odnosno izgradnjom stabla koje na osnovu definiranih ulaznih podataka izračunava iduću vrijednost vremenskog slijeda. Programska implementacija rješenja pisana je u programskom jeziku *Java*, a sama struktura je inspirirana onom koju koristi *Evolutionary Computation Framework (ECF)* od kojeg je preuzeta ideja razvoja okruženja koje omogućava osnovne GP algoritme te koje je lako proširivo dodatnim operatorima i funkcijskim čvorovima po potrebi, uz mogućnost lake konfiguracije kroz jedinstvenu konfiguracijsku datoteku. Sam *ECF* ipak nije korišten zbog kasnije potrebe za dodatnim podatkovnim strukturama koje nisu potrebne kod uobičajenih GP implementacija. Detalji osnovne implementacije opisani su u ovom poglavlju, a kasnije optimizacije i poboljšanja rješenja kasnije u radu.

Za provjeru rezultata implementiranog algoritma korišteni su javno dostupni skupovi podataka spomenuti u [12]. Svaki od skupova podataka podijeljen je na skup za učenje na kojem se genetskim programiranjem evoluira stablo te skup za testiranje nad kojim se evoluirani model evaluira. Skup za učenje činio je 70% podataka, a skup za ispitivanje 30% podataka, koji nisu bili nasumično odabrani već se održavala distribucija uzoraka po satima u danu (ako je originalni podatkovni skup imao 4% očitanja skupljenih u 23h, otprilike toliki postotak očitanja skupljenih u 23h trebao je biti u skupu za učenje i skupu za ispitivanje). Iako je ovakva raspodjela automatska u slučaju da su podaci podijeljeni po satima, budući da se svi podatci učitavaju odjednom kako bi se kasnije po potrebi dijelili po satima za različite modele, potrebno je eksplicitno zadati raspodjelu programu kako bi bili pokriveni svi slučajevi.

4.1. Jednosatni i višesatni modeli

U početnoj implementaciji model je izgrađen tako da bude usporediv sa onim opisanim u [1], odnosno ne evoluira se jedinstven model koji predviđa vrijednost funkcije za sve sate u danu, već se evoluira 24 zasebna modela od kojih onda svaki radi predviđanja za jedan sat u danu. Osnovna prednost ovog pristupa leži u trivijalnoj paralelizaciji učenja i predviđanja te mogućnosti prilagođavanja (ponovnog učenja) modela nakon svakog očitavanja stvarne vrijednosti. Glavna mana pristupa leži u činjenici da je ipak memorijski zahtjevnije pohranjivati i interpretirati 24 stabla nego jedno koje bi predviđalo potrošnju za cijeli dan.

Dodatno je implementirana varijanta koja slijedi princip granulacije modela, ali ne tako ekstremno. Podatci o potrošnji električne energije vizualizirani su na slici 4.1 te je iz grafa zaključeno da je dovoljno podijeliti dan na 5 modela umjesto 24, gdje modeli redom predviđaju potrošnju električne energije za sate u danu redom: 2h - 6h, 7h - 9h, 10h - 17h, 18h - 22h, 23h - 1h. Unutar ovakvih intervala varijacija potrošnje je dovoljno malena da se može očekivati da će model biti gotovo jednako precizan kao onaj gdje svako stablo predviđa jedan sat u danu, a memorijski zahtjevi će biti znatno smanjeni.



Slika 4.1: Kretanje srednje potrošnje električne energije kroz dan

4.2. Ulazni podatci

Kao ulazni podatci u ovom radu korišteni su ISO New England podatci o potrošnji električne energije. Podatci su razvrstani po satima kako bi se mogli dati različitim modelima, te se za učenje svakom modelu predaje par datoteka (X, Y), svaki sa po n zapisa. Svaki i-ti element skupa Y y_i je jedna realna vrijednost koja predstavlja promatranu vrijednost vremenskog slijeda u trenutku i. Elementi skupa X x_i su vektori sa realnim i cjelobrojnim varijablama. Realne vrijednosti su vanjska temperatura, vlažnost zraka (ako je dostupna na skupu), prosjek posljednjih m očitanja trenutnog modela (za isti sat), te prosjek vrijednosti vremenskog slijeda u posljednjih k sati. Cjelobrojne varijable su dan u tjednu, dan u mjesecu, mjesec u godini, te cjelobrojno kodirana vrijednost koja predstavlja je li trenutni, sljedeći ili prethodni dan praznik (0 predstavlja niti jedno, 1 da je trenutni dan praznik, 2 da je sljedeći dan praznik, 3 da je prijašnji dan praznik, a 4 da je dva dana prije bio praznik). Preglednije su sve ove vrijednosti dane u tablici 4.1.

Tablica 4.1: Ulazne varijable sa opisom i tipom varijable

Varijabla	Tip varijable	Kratki opis
Temperatura	Realna	Izmjerena vanjska temperatura
Vlažnost	Realna	Izmjerena vlažnost zraka (ako je dostupna)
Prosjek m	Realna	Prosjek posljednjih m očitanja vrijednosti vremenskog slijeda u istom satu (vrijednosti y_{i-24}, y_{i-48}, ... , y_{i-m24})
Prosjek k	Realna	Prosjek posljednjih k očitanja vrijednosti vremenskog slijeda
Dan u tjednu	Cjelobrojna	1 ako je ponedjeljak, 2 za utorak itd.
Dan u mjesecu	Cjelobrojna	Izravni zapis datuma iz kalendara
Mjesec u godini	Cjelobrojna	1 ako je siječanj, 2 ako je veljača itd.
Praznik	Cjelobrojna	1 ako je trenutni dan praznik, 2 ako je sljedeći dan praznik, 3 ako je prijašnji dan bio praznik, 4 ako je pretprošli dan bio praznik, 0 ako ništa od ovog nije zadovoljeno

4.3. Funkcije pogreške

Na skupovima podataka korištenim mogu se koristiti različite funkcije za mjeru pogreške modela. Kod svake mjere pogreške koriste se standardne oznake, gdje je $(\mathbf{x_i}, y_i)$ par ulaznog vektora podataka i očekivanog izlaza funkcije, a $f(\mathbf{x_i})$ je izlaz modela za dani ulazni vektor. Dodatno, m označava broj uzoraka. Najjednostavnija mjera

pogreške je srednja apsolutna pogreška, definirana kao:

$$err = \frac{\sum_{i=1}^{m} |f(\mathbf{x_i}) - y_i|}{m}$$

Nešto češća mjera pogreške, slična srednjoj apsolutnoj pogrešci, je srednja kvadratna pogreška, definirana kao:

$$err = \frac{\sum_{i=1}^{m} (f(\mathbf{x_i}) - y_i)^2}{m}$$

Zadnja mjera pogreške, koja uzima u obzir moguće dosta velike razlike unutar očekivanih izlaza uzoraka kako pogreške na izlazima većeg reda veličine ne utječu na ukupnu pogrešku ništa više od onih na manjima izlazima.

$$err = \frac{\sum_{i=1}^{m} |\frac{f(\mathbf{x_i}) - y_i}{y_i}|}{m}$$

4.4. Funkcija dobrote

Kao funkcija dobrote modela, koristi se srednja relativna pogreška (opisana u prijašnjem odjeljku) na skupu za učenje. Preciznije, Ista mjera se na skupu za ispitivanje koristi za ocjenu dobivenog modela.Kako tipično vrijednosti potrošnje mogu dosta varirati kroz sat u danu te je najčešće traženi izlaz broj reda veličine 10^4 do 10^5, korištenje srednje apsolutne pogreške ili srednje kvadratne pogreške može uzrokovati prilične praktične probleme u izračunu (pogreška u satima s očekivanim velikim brojem kao izlazom koja je gledajući relativne vrijednosti jednaka onoj u satima s manjim izlazom uzrokovat će značajno veću kaznu na modelu) te se zbog toga bira relativna pogreška koju se u evoluciji modela minimizira.

4.5. Skup čvorova stabla

Za izgradnju stabla u genetskom programiranju potrebno je unutar osnovnog algoritma i genetskih operatora koje koristi definirati skup čvorova koji potencijalno mogu izgrađivati jedinku, odnosno stablo. Kako ne postoji skup čvorova koji bi bio univerzalno primjenjiv na svaki problem optimizacije, klasifikacije ili regresije, potrebno je definirati potencijalne čvorove stabla koji su dovoljni za izgradnju prihvatljivog mo-

dela. Definirani su zasebno terminalni čvorovi (listovi) te unutarnji čvorovi stabla, koji se pak mogu jasno podijeliti na aritmetičke operatore (preciznije funkcijske čvorove) i čvorove koji predstavljaju operatore grananja.

4.5.1. Funkcijski čvorovi

Kako se kod predviđanja vremenskih sljedova ustvari radi o simboličkoj regresiji, polazi se od pretpostavke da je skup aritmetičkih operatora, odnosno binarnih i unarnih funkcija dovoljan da se s prihvatljivom preciznošću evoluira funkcija koja odgovara danom skupu podataka. Implementirani su svi osnovni aritmetički operatori (zbrajanje, oduzimanje, množenje i dijeljenje) kao čvorovi s dva djeteta te eksponencijalna funkcija, prirodni logaritam, korijen funkcija te apsolutna vrijednost kao čvorovi s jednim djetetom.

Iako su implementirani svi navedeni operatori, sama programska implementacija ih ne koristi nužno sve, već se u konfiguracijskoj datoteci pri svakom pokretanju može odabrati podskup funkcija koje će biti kandidati za unutarnje čvorove stabla. Veći skup operatora u teoriji omogućava pretraživanje većeg prostora stanja, odnosno evoluiranje većeg broja funkcija, no također nije potrebno nepotrebno povećavati prostor pretraživanja ako postojeći skup operatora već može dovoljno dobro aproksimirati funkciju.

4.5.2. Terminalni čvorovi stabla

Kao terminalni čvorovi, odnosno listovi stabla korištene su ulazne varijable podatkovnog skupa na kojima se provodi učenje i predviđanje potrošnje. Ako nije drugačije definirano, kandidati za listove su sve ulazne varijable instance problema, no ovo se može nadjačati definiranjem ulaznih varijabli u konfiguracijskoj datoteci.

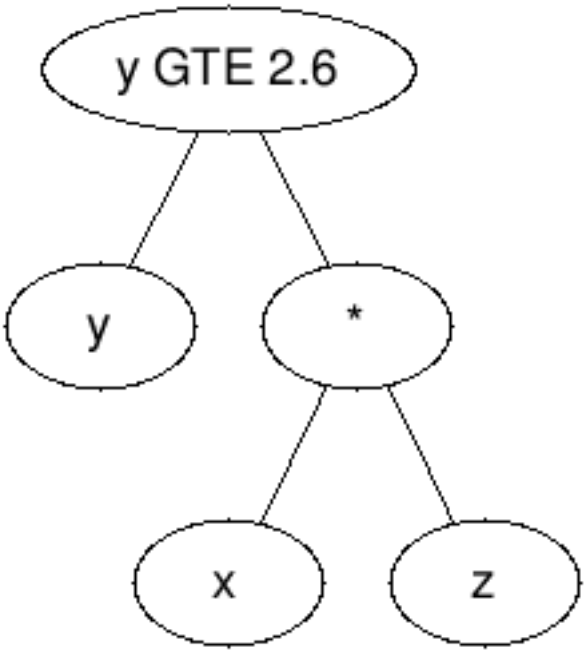
Dodatno su isprobani i terminalni čvorovi koji predstavljaju nasumičnu realnu konstantu s namjerom da postoje čvorovi koji bi mogli onda izvršiti posmak funkcije po njenim osima u n-dimenzionalnom prostoru. Ipak, kod njih se posebno mora paziti na mogućnost evolucije pretjerano prilagođenih (engl. *overfitted*) stabala jer u slučaju da je varijacija podataka dovoljno mala, moguće je da se korištenjem dovoljno konstanti može dobiti mala pogreška modela.

4.5.3. Operatori grananja

Iako su osnovni aritmetički operatori dovoljno dobri za aproksimirati velik broj kompleksnih funkcija, pri logičkom razmatranju ulaznih podataka uočene su mogućnosti modeliranja izlaza na načine koje bi bilo lako programski implementirati, ali ih nije moguće opisati aritmetičkim operatorima. Naime, na osnovu varijabli koje predstavljaju mjesec u godini, dan u tjednu i posebno činjenicu je li trenutni datum praznik ili ne mogla bi se izlazna vrijednost stabla modificirati (logički se ovo zasniva na činjenici da je potrošnja električne energije ljeti različita od one zimi, te da je potrošnja električne energije u kućanstvima oko praznika drugačija nego tijekom ostatka godine). Ovakvo ponašanje ne može se opisati prije navedenim aritmetičkim operatorima, te se stvara potreba za definiranjem ponašanja sličnog operatoru grananja. U tu svrhu implementirana su dva različita operatora koji definiraju ovakvo ponašanje te su detaljnije opisani ovdje.

Operator grananja sa dva operanda

Osnovni operator grananja koji je implementiran u stablo koje se evoluira se ugrađuje tako da mu se postaje dva čvora djeteta te se nasumično inicijaliziraju jedna varijabla (terminalna vrijednost stabla), operator ispitivanja (GTE ili LTE) te granična vrijednost. Pri izračunu vrijednosti čvora tada se provjerava je li ovisno o operatoru ispitivanja vrijednost varijable veća ili jednaka (GTE) ili manja ili jednaka (LTE) od granične vrijednosti - vrijednosti desne strane (engl. *right hand side*). U slučaju da provjera vrati istinitu vrijednost, vrijednost čvora odgovara vrijednosti prvog čvora djeteta, dok se u slučaju da nejednakost definirana u čvoru nije zadovoljena vraća vrijednost drugog čvora djeteta. Primjer ovakvog čvora je na slici 4.2.



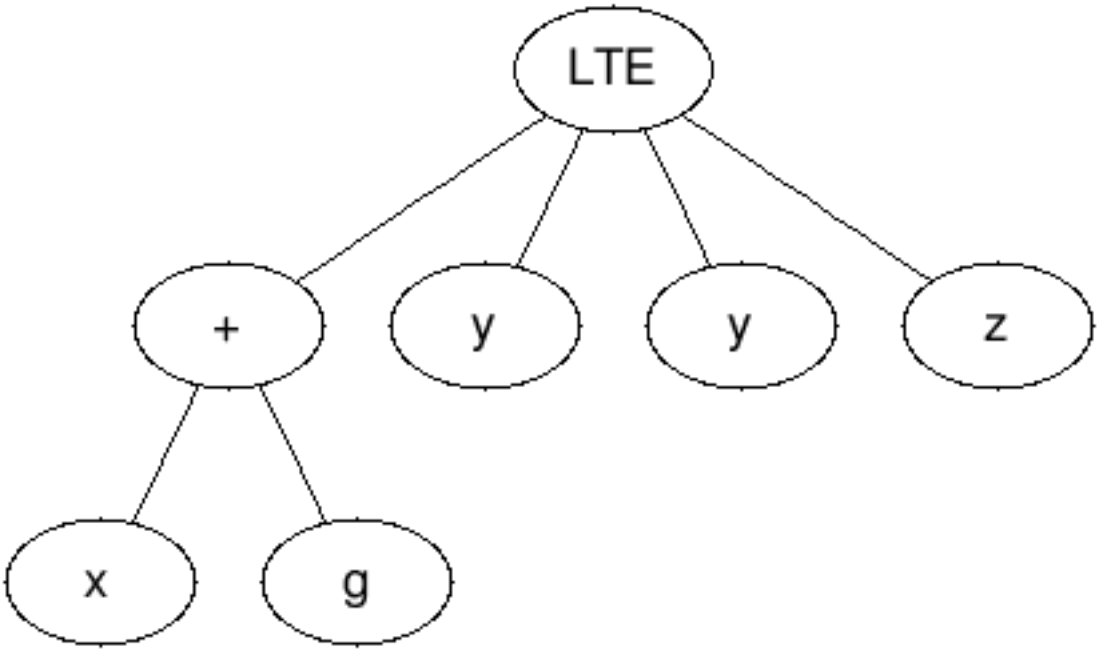
Slika 4.2: Operator grananja sa dva operanda. Ako je ispunjen uvjet $y \geq 2.6$ vrijednost čvora se evaluira kao y, a inače kao $x * z$

Kako se ne bi riskirao gubitak ovakvih operatora u stablu i povećao broj kombinacija varijabli koje se ispituje stvara se potreba za posebnim operatorom mutacije za ovakve čvorove. Tako je implementiran specifičan postupak za mutaciju ovih čvorova gdje ako se mutira ovaj čvor, ne mijenja se nijednim drugim čvorom već se s određenom vjerojatnošću mijenjaju varijabla koju se ispituje, operator ispitivanja i/ili vrijednost s desne strane.

Operator grananja sa četiri operanda

Prvi opisani operator grananja teoretski pokriva dovoljno velik prostor mogućih nejednakosti, očiti su neki nedostatci koji se ne mogu ispraviti u takvoj implementaciji. Prvi je činjenica da se provjeravaju samo pojedinačne varijable (terminalne vrijednosti stabla) i određene granične vrijednosti, što ne samo da ograničava prostor pretraživanja već ne daje mogućnost opisivanja operatora grananja u kojem bi se provjeravala vrijednost jedne varijable ili podstabla u ovisnosti o drugoj varijabli. Za potrebe boljeg opisivanja operatora grananja implementiran je čvor grananja koji prima četiri čvora djeteta. Iako se ovakav čvor po strukturi i funkciji dosta razlikuje od ostalih čvorova stabla, po principu rada je izravniji od prvog opisanog operatora grananja. Pri inicijalizaciji čvora definira se samo operator ispitivanja (ponovno GTE ili LTE, sa istom funkcijom). Izračun vrijednosti čvora tada se radi tako da se ovisno o operatoru us-

poredbe usporede vrijednosti prva dva čvora djeteta. Ako je nejednakost istinita, kao vrijednost čvora vraća se vrijednost trećeg čvora djeteta, a inače ona četvrtog čvora djeteta. Ovo daje mogućnost usporedbe proizvoljno mnogo kombinacija varijabli i podstabala, za razliku od prvog opisanog operatora koji uspoređuje varijable s konstantama. Primjer čvora dan je na slici 4.3



Slika 4.3: Operator grananja sa četiri operanda. Ako je ispunjen uvjet $x + z \leq y$ vrijednost čvora se evaluira kao y, a inače kao z

Kako je struktura čvora puno jednostavnija, operator mutacije implementiran za ovaj čvor s definiranom vrijednošću jednostavno mijenja operator ispitivanja unutar čvora na onaj suprotni (točnije, GTE postaje LTE i obratno). Uz definirane operatore križanja ovime se postiže isti efekt kao s kompliciranijim operatorom mutacije za prvi tip operatora grananja, uz puno jednostavniju implementaciju. Naravno, izgubljena je mogućnost usporedbe varijabli ili podstabla s konstantnim vrijednostima, no potrebe za tim se mogu zadovoljavati dopuštanjem oba tipa operatora grananja u stablu.

U tablici 4.2 je za kraj ovog poglavlja dan pregled svih operatora implementiranih i korištenih za potrebe evolucije stabala korištenih kao modela u ovom radu.

Tablica 4.2: Popis operatora korištenih u genetskom programiranju

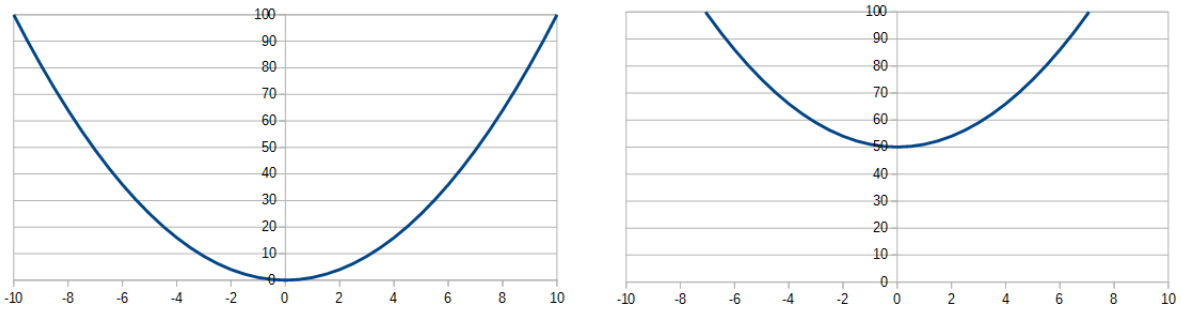
Operator (simbol)	Broj operanada	Vrijednost funkcije
Zbrajanje (+)	2	$f(\mathbf{x}) = x_1 + x_2$
Oduzimanje (-)	2	$f(\mathbf{x}) = x_1 - x_2$
Množenje (*)	2	$f(\mathbf{x}) = x_1 * x_2$
Dijeljenje (/)	2	$f(\mathbf{x}) = x_1/x_2$
Apsolutna vrijednost (abs)	1	$f(\mathbf{x}) = \|x_1\|$
Eksponencijalna funkcija e^x (exp)	1	$f(\mathbf{x}) = e^{x_1}$
Prirodni logaritam (log)	1	$f(\mathbf{x}) = ln(x)$
Korijen funkcija (sqrt)	1	$f(\mathbf{x}) = \sqrt{x_1}$
Operator grananja s dva čvora djeteta (if2)	2	x_1 ako je ispunjen uvjet grananja, x_2 inače
Operator grananja s četiri čvora djeteta (if4)	4	x_3 ako je ispunjen uvjet grananja između x_1 i x_2, x_4 inače

5. Optimizacija postupka rješavanja

Iako GP već u osnovnoj varijanti algoritma može dati dosta prihvatljive rezultate za ovaj problem, ipak ostaju otvorene mnoge mogućnosti za poboljšanje dobivenih rezultata. Kako većina drugih radova koristi metode strojnog učenja za predviđanje vrijednosti vremenskih sljedova, korištene optimizacije zasnivaju se baš na postojećim metodama iz sfere strojnog učenja.

5.1. Linearno skaliranje

Kod osnovne implementacije simboličke regresije genetskim programiranjem uočen je problem učenja funkcija koje sadrže posmak po nekoj od koordinantnih osi. Uzme li se za primjer funkcija poput $f(x_1, x_2, ..., x_n) = x_1^2$, moguće je korištenjem ulaznih varijabli x_i kao terminalnih čvorova i aritmetičkih operatora kao unutarnjih čvorova evoluirati stablo koje potpuno točno modelira ovakvu funkciju. Uzme li se u obzir funkcija g koja odgovara posmaku osnovne funkcije, tako da $g(x_1, x_2, ..., x_n) = x_1^2 + C$, gdje je C realna konstanta, nemoguće je potpuno točno modelirati takvu funkciju. Primjer dvije ovakve funkcije dan je na slici 5.1. Ako se dozvoli dovoljna dubina stabla, moguće je dobiti model koji će ovakvu funkciju pokušati modelirati korištenjem mnogo više varijabli, čime će se na skupu za učenje dobiti relativno mala pogreška jer je u višim dimenzijama moguće aproksimirati vrijednosti u m točaka korištenjem dovoljno varijabli, no takva funkcija će biti pretjerano prilagođena i sigurno oblikom neće odgovarati originalnoj funkciji, što će vrlo vjerojatno dovoditi do vrlo loših rezultata na ispitnom skupu. Alternativa ovom pristupu je uvođenje terminalnog čvora koji predstavlja nasumičnu realnu konstantu, no kako je ranije opisano u radu, dopuštanje ovih čvorova u stablu također često dovodi do pretjerano prilagođenih modela, što opet nije prihvatljivo.



Slika 5.1: Usporedba funkcija $f(x) = x^2$ i $g(x) = x^2 + 50$. Funkcije su istog oblika, no desna funkcija je po x osi posmaknuta za 100

Rješenje ovakvog problema predloženo je u [6] i naziva se linearnim skaliranjem (engl. *linear scaling*).

Linearno skaliranje provodi se tada tako da se za dani izlaz modela y i stvarnu vrijednost t te njihove srednje vrijednosti na m uzoraka $\bar{y}$ i $\bar{t}$ izračunavaju koeficijenti a i b kao:

$$b = \frac{\sum[(t-\bar{t})(y-\bar{y})]}{\sum[(y-\bar{y})^2]}$$

$$a = \bar{t} - b\bar{y}$$

Tada se umjesto y kao izlaz iz modela za izračun pogreške daje skalirana vrijednost $y' = a + by$. Ovim postupkom postiže se minimizacija srednje kvadratne pogreške na podatkovnom skupu u $O(m)$ vremenu (što implicitno smanjuje i srednju relativnu pogrešku), a efekt odgovara provođenju jednostavne linearne regresije. Zahvaljujući linearnom skaliranju, evoluirana funkcija mora aproksimirati samo oblik funkcije, a ne i potencijalni posmak, čime se smanjuje pogreška modela bez rizika preprilagođenosti modela.

Važno je uočiti da se u osnovnoj varijanti linearno skaliranje koristi na skupovima za učenje, te po potrebi i ispitivanje, gdje su poznati odgovarajući parovi ulaznih i izlaznih podatka. Ako nije poznat željeni izlaz, već samo izlaz modela, ne može se tako precizno modificirati izlaz modela. Ipak, pod pretpostavkom da će stvarni podatci barem djelomično odgovarati onima iz skupa za učenje i ispitnog skupa, konstante a i b mogu se izračunati pri učenju, odnosno izračuna se njihova srednja vrijednost na cijelom skupu podataka, te se ta vrijednost koristi pri radu samog modela. Ovakav pris-

tup može ili nadmašiti ili zamijeniti evoluiranje stabla sa čvorovima koji predstavljaju nasumične konstante.

5.2. Semantičko genetsko programiranje

Pri pokretanju osnovnih varijanti genetskog programiranja u sklopu ovog rada, uočen je zabrinjavajuće velik broj situacija gdje je konačni evoluirani model pribrajao vrijednostima predviđene potrošnje cjelobrojne vrijednosti poput dana u mjesecu ili mjeseca u godini kako bi smanjio relativnu pogrešku. Iako je pretpostavka bila da će linearno skaliranje opisano u ovom radu uspjeti otkloniti potrebnu za ovakvom prejakom prilagodbom (engl. *overfitting*), ostala je činjenica da je potrebno ukloniti mogućnost da se u operacijama poput zbrajanja i oduzimanja miješaju varijable koje imaju različite semantičko značenje.

Semantiku varijabli koja je ovdje opisana najlakše je interpretirati kao mjernu jedinicu svake od varijabli. Tako jedna od semantika može biti temperatura zraka, koja bi se inače mjerila u stupnjevima, te također varijable poput dana u tjednu, mjeseca u godini i sličnih varijabli koje predstavljaju vrijeme dobivaju svaka svoju semantiku. Ovaj problem već je opisan u [7] gdje je dana ideja pridruživanja semantičkih vrijednosti svakoj od ulaznih varijabli (odnosno terminalnih vrijednosti stabla), te definiranje izračuna semantičke vrijednosti čvora na osnovu semantičkih vrijednosti njegovih podstabala.

Pri implementaciji semantičkog GP nametnulo se pitanje hoće li se kao u originalnom radu kršenje semantičkih pravila kažnjavati u izračunu vrijednosti funkcije dobrote ili će se kao u [13] pri izgradnji stabla i primjeni genetskih operatora onemogućiti stvaranje stabala koja u sebi sadržavaju semantički neispravne čvorove. Unatoč većoj točnosti (odnosno strogoj primjeni semantičkih pravila) drugog pristupa, zbog razlika u tipu i opsegu problema koji se opisuje u ovom radu odabrana je ipak prva metoda, odnosno samo kažnjavanje semantičkih pogrešaka. Razlog za ovaj odabir je činjenica da se radi sa većim brojem različitih semantika u ulaznim varijablama (dani, temperatura, potrošnja energije, vrijednosti bez semantike itd.) te je mogućnost stvaranja semantičkih pogrešnih podstabala dovoljno visoka da je procijenjeno da bi utrošak vremena potrebnog za ponovno generiranje podstabala kod svakog kršenja semantičkih pravila bio nepotrebno visok te da je bolje jednostavno kažnjavati jedinke i pustiti evoluciji da se pobrine za minimiziranje učestalosti semantičkih pogrešaka.

Poželjno, najveći napredak trebao bi se pokazati u činjenici da bi evoluirana stabla bila smislenija i prosječnom korisniku kao modeli, no naravno poželjno je da ovaj pristup minimizira i pogrešku samog modela.

5.2.1. Implementacija semantičkog genetskog programiranja

Uzevši u obzir spomenuti cilj kažnjavanja jedinki koje krše semantička pravila, u konfiguraciji programa za svaku varijablu definirana je njena semantika koja je predstavljena cijelim brojem. Pri izračunu vrijednosti svakog od čvorova tada se također izračunava semantika tog čvora na osnovu njegovih podstabala. Pravila izračuna semantika čvorova na osnovu podstabala dana su u tablici , gdje x predstavlja semantiku čvora a $y_1, y_2, ..., y_n$ predstavljaju semantike svakih od n čvorova djece. Određeni čvorovi poput množenja i dijeljenja dopuštaju da im djeca imaju različite semantike, dok neki poput zbrajanja i oduzimanja to gledaju kao semantičku pogrešku. U tom slučaju jedinci se dodaje vrijednost funkcije kazne za semantičku pogrešku, a za čvor se provodi oporavak, odnosno njegova semantička vrijednost se izračunava kao da nije bilo pogreške.

U tablici 5.1 pri računanju semantike sa s_i označena je semantika i-tog čvora djeteta.

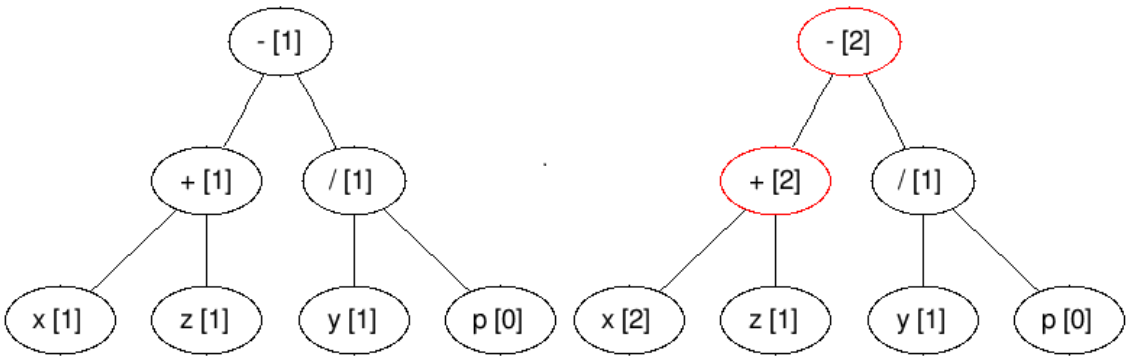
Tablica 5.1: Izračunavanje semantičkih vrijednosti čvorova

Operator (simbol)	Uvjet semantike	Semantika rezultantnog čvora
Zbrajanje (+)	$s_1 = s_2$	s_1
Oduzimanje (-)	$s_1 = s_2$	s_1
Množenje (*)		$s_1 + s_2$
Dijeljenje (/)		$s_1 - s_2$
Apsolutna vrijednost (abs)		s_1
Eksponencijalna funkcija e^x (exp)	$s_1 = 0$	0
Prirodni logaritam (log)	$s_1 = 0$	0
Korijen funkcija (sqrt)	$s_1 = 0$	0
Operator grananja s dva čvora djeteta (if2)		s_1 ako je ispunjen uvjet grananja, s_2 inače
Operator grananja s četiri čvora djeteta (if4)	$s_1 = s_2$	s_3 ako je ispunjen uvjet grananja, s_4 inače

Pri ovoj implementaciji nametnuo se problem propagacije semantičkih pogrešaka u stablu. Naime, dani model oporavka od pogreške će prikriti činjenicu da se na većim dubinama stabla dogodila semantička pogreška, iako bi ispravno bilo dodatno penalizirati jedinku jer će se pogreška propagirati dalje kroz stablo. Zato se za definiranu konstantu SEM_KAZNA koja predstavlja funkciju kazne u slučaju semantičke pogreške kazna izračunava kao $(d + 1) * SEM\_KAZNA$, gdje d predstavlja dubinu čvora u kojem se dogodila semantička pogreška. Tako će se strože kažnjavati semantičke pogreške bliže listovima stabla, koje bi se inače propagirale kroz stablo.

Primjer dva različita stabla sa semantičkim oznakama prikazan je na slici 5.2, gdje su prikazana dva semantička stabla, na lijevoj strani jedno semantički ispravno, a desno stablo sa dvije semantičke pogreške. U svakom čvoru u uglatoj zagradi je naznačena semantika čvora, predstavljena cijelim brojem. U početku je semantika definirana samo za terminalne čvorove, odnosno one na dubini 2, a za čvorove bliže korijenu stabla semantika se izračunava po definiranim pravilima. U desnom stablu greška se prvo događa u čvoru na dubini 1, no oporavak od pogreške izračunava semantiku čvora tako da bude jednaka semantici prvog operanda. Ipak, ovo dovodi

di pogreške u korijenu stabla, gdje oporavak od pogreške opet na isti način izračunava semantiku čvora. Tako je ukupna kazna za semantičku pogrešku stabla jednaka $(1+1) * SEM\_KAZNA + (0+1) * SEM\_KAZNA = 3 * SEM\_KAZNA$.



Slika 5.2: Primjer semantički ispravnog i semantički neispravnog stabla sa crveno označenim čvorovima pri čijoj se evaluaciji događa semantička pogreška.

Dodatna prednost ovog pristupa u odnosu na onaj opisan u [13] je činjenica da nije potrebno modificirati postojeće operatore križanja i mutacije u odnosu na već postojeći pristup uporabom GP, budući da se dozvoljava nastajanje semantički neispravnih stabala.

5.3. Algoritmi skupnog učenja

U strojnom učenju se za probleme regresije na kojima klasični algoritmi imaju problema sa učenjem težih uzoraka primjenjuju se metode skupnog učenja (engl. *ensemble learning*), u kojima se učenjem skupa "slabih" modela (ili algoritama) koji imaju problema sa učenjem skupa podataka pokušava proizvesti "jaki", odnosno precizniji model. Jedna od njih je *boosting*, gdje se skup inkrementalno gradi tako da se u svakoj iteraciji novi model fokusira više na instance podataka na kojima je prijašnji model imao veću pogrešku. Najpopularniji od ovih algoritama je *AdaBoost* [5], koji na težim podatkovnim skupovima generalno daje bolje rezultate i manju varijancu od klasičnih modela klasifikacije i regresije uz uočeni veći rizik prejake prilagodbe. Drugi tip algoritama skupnog učenja je *bagging* (od engl. *bootstrap aggregating*), kojima se uzorci iz skupa za učenje mijenjaju duplikatima drugih uzoraka, čime se stvaraju novi skupovi s manjom varijancom. Uz ovo, rezultati korištenja ovog principa često pokazuju manju vjerojatnost pretjeranog prilagođavanja skupu za učenje.

Kako genetsko programiranje pri evoluciji već koristi velik broj "slabih" modela u populaciji, dolazi se do pitanja jesu li ove metode, ili barem ideja koja stoji iza njih, primjenjive i u kombinaciji sa genetskim programiranjem. Dvije primjene *boosting* principa na genetsko programiranje opisane su u [4], te su isprobane u sklopu ovog rada kako bi se poboljšali rezultati dobiveni korištenjem osnovnog algoritma. U teoriji ove metode ne bi nikako smjele pogoršati rezultate dobivene bez njih, osim u slučaju prejake prilagodbe modela skupu za učenje.

5.3.1. GPBoost

Prvi od implementiranih algoritama je *GPBoost*, koji je izravna prilagodba algoritma *AdaBoost* u domeni genetskog programiranja. Pseudokod algoritma dan je u algoritmu 2, a općenito je najbitniji dio algoritma korištenje vektora koji predstavlja težinu svakog od uzoraka. U prvoj iteraciji težine uzoraka uniformno su raspodijeljene, no kasnije se težine ažuriraju na osnovu pogreške modela na svakom od uzoraka. Vrijednosti funkcije dobrote svake od jedinke također se ažuriraju ovisno o težinama uzoraka, tako da modeli koje manje griješe na težim uzorcima dobivaju veću vrijednost funkcije dobrote. Budući da algoritam za svaku od jedinki izračunava i stopu pouzdanosti, kao konačan izlaz gleda se srednja vrijednost izlaza svih dovoljno pouzdanih jedinki. Važno je napomenuti da ovaj algoritam originalno nije dizajniran da rješava probleme regresije, već klasifikacije, te je implementacija u ovom radu jednostavno prilagodba postojećeg algoritma na drugi tip problema. Tako bi se zapravo za povratnu vrijednost ovog algoritma moglo drugačije implementirati koje stablo/skup stabala se vraća (samo najbolje, ili staviti drugačiju granicu pouzdanosti, ovisno o srednjoj pouzdanosti populacije), no generalno se ideja dovoljno dobro translatira iz domene klasifikacije.

Problem koji se može uočiti već iz pseudokoda algoritma je povećanje vremenske složenosti učenja ovakvog modela u odnosu na osnovnu implementaciju genetskog programiranja koja proizlazi iz češćeg izračuna vrijednosti funkcije dobrote. U osnovnoj implementaciji za svaku jedinku funkcija dobrote se računa točno jednom i ostaje konstantna kroz cijeli životni vijek jedinke. Ovo je moguće zbog toga što vrijednost funkcije dobrote ovisi samo o pogrešci na skupu za učenje, a parovi ulaz - izlaz su konzistentni kroz proces učenja, dok se jedinka ne mijenja (budući da se mutacija radi samo na novonastalim jedinkama, odnosno onima na kojima još nije ni proveden izračun funkcije dobrote). *GPBoost* uvodi potrebu za ažuriranjem vektora težina na osnovu

Algoritam 2 GPBoost

m je veličina skupa za učenje, t broj iteracija a (x_i, y_i) ulaz-izlaz par skupa za učenje

f predstavlja jedinku, a f_t najbolju jedinku u iteraciji t
$D_1(i)$:= 1/m
for t := 1; t <= T; t++ **do**
fit := $\sum_{i=1}^{m} |f(x_i) - y_i| * D_t(i) * m$
$L_i := \frac{|f_t(x_i)-y_i|}{max_{i=1...m}|f_t(x_i)-y_i|}$
$\bar{L} := \sum_{i=i}^{m} L_i D_i$
$\beta_t := \frac{\bar{L}}{1-\bar{L}}$
$D_{t+1}(i) := \frac{D_t(i)^{1-L_i}}{Z_t}$, gdje je Z_t normalizacijski faktor
end for
return $F(x_i)$ = $avg(f(x_i)$ takav da $\beta >= 0.5$

kojeg se izračunava vrijednost funkcije dobrote svake jedinke. Kako je u svakoj iteraciji vektor modificiran u odnosu na prošlu iteraciju, moraju se osvježiti i vrijednosti funkcije dobrote (iz kojih se opet računaju nove težine). Zbog ovoga, broj evaluacija funkcije dobrote značajno je povećan, a kako se za izračun mora izračunati pogreška na svim uzorcima unutar skupa za učenje, ovo uzrokuje značajno povećanje vremena potrebnog za učenje modela.

5.3.2. BCC

Drugi algoritam opisan u [4] je *Boosting Correlation Coefficients*, koji je adaptacija *boosting* principa s namjerom da budu korišteni za rješavanje problema regresije. Algoritam se također temelji na korištenju težinskog vektora pri računanju vrijednosti funkcije dobrote, s razlikom da se težine osvježavaju na temelju korelacijskih koeficijenata. Korelacijski koeficijenti statistička su mjera međuzavisnosti dvije nasumične varijable, odnosno opisuju koliko promjena jedne varijable utječe na drugu. Dodatno, algoritam je napravljen s namjerom da bude korišten za regresiju i predviđanje, pa bi teoretski trebao davati veća poboljšanja rezultata nego *GPBoost*. Pseudokod algoritma dan je u algoritmu 3.

Važan dio algoritma je korištenje vrijednosti korelacije dvaju varijabli, koji je za varijable X i Y sa m uzoraka definiran kao:

Algoritam 3 Boosting Correlation Coefficients

$D_1(i)$:= 1/m
for t := 1; t <= T; t++ **do**
  fit := $\sum_{i=1}^{m} |f(x_i) - y_i| * D_t(i) * m$
  $L_t(x_i) := 1 - exp(-\frac{|f_t(x_i)-y(x_i)|}{max_{i=1..m}|f_t(x_i)-y(x_i)|})$
  $D_{t+1}(x_i) := \rho_t(f_t(x_i), y(x_i)) * D_t * L_t(x_i)$
end for
if $1 \leq i \leq m-1$ **then**
  return $F(x_{i-1}) = \frac{\sum_{i=1}^{T} \rho_t(f_t(x_i),y_i)*f_t(x_i)}{\sum_{i=1}^{T} \rho_t(f_t(x_i),y_i)}$
else if $i = m$ **then**
  return $F(x_m) = \frac{\sum_{i=1}^{T} f_t(x_i)}{T}$
end if

$$\rho(x,y) = \frac{\sum_{i=1}^{m}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{m}(x_i-\bar{x})^2 \sum_{i=1}^{m}(y_i-\bar{y})^2}}$$

Ponovno je važno uočiti da i ovaj algoritam ima isti implementacijski problem kao i *GPBoost* koji proizlazi iz potrebe za stalnim ažuriranjem vrijednosti funkcije dobrote. Ipak, kako je algoritam prilagođen za regresijske probleme, veća je mogućnost da će ovdje veća vremenska složenost biti opravdana većom preciznošću konačnog modela dobivenog evolucijom.

6. Rezultati

U sklopu ovog rada implementirana su mnoga potencijalna poboljšanja i modifikacije originalnog algoritma genetskog programiranja te je prije konačnog prikaza rezultata potrebno ispitati utjecaj svih mogućih modifikacija algoritma kako bi se najbolja kombinacija mogla primijeniti za dobivanje konačnog algoritma te najboljeg modela za predviđanje potrošnje električne energije.

Konačni rezultati dani su u odjeljku 6.5 gdje je prikazana srednja relativna pogreška modela po mjesecima, za razliku od ostalih odjeljaka gdje se prikazuje ili ukupna greška modela ili greška po satima (kako bi se provjerio utjecaj granulacije modela po satima). Ovo je napravljeno kako bi se rezultati ovog rada mogli usporediti sa postojećim radovima koji pretežito koriste takav prikaz pogreške modela.

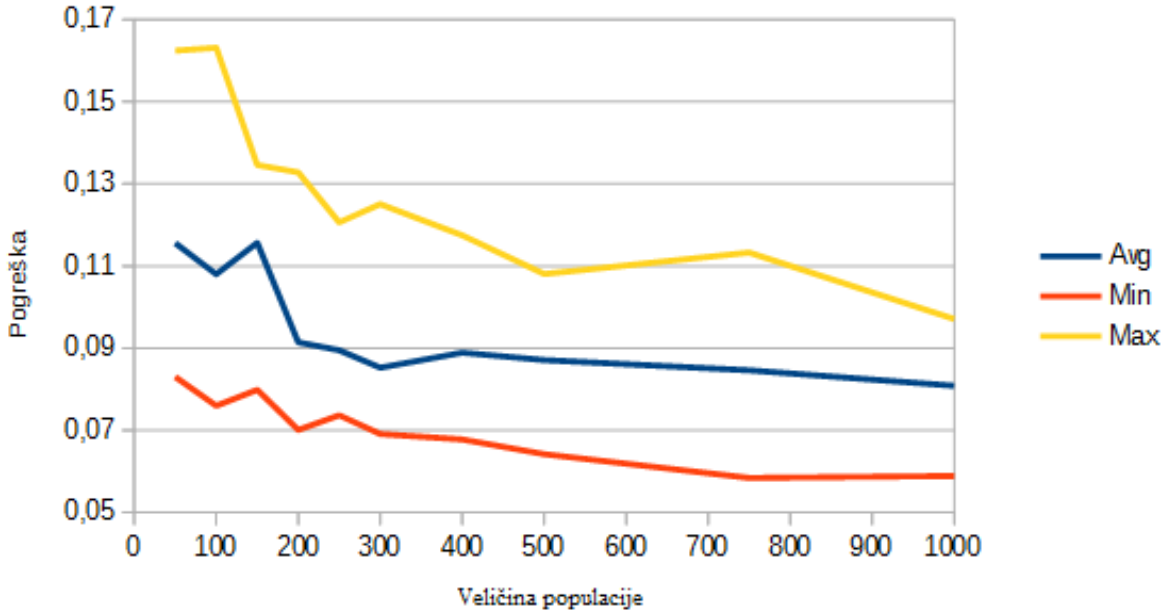
Osnovna podjela podatkovnog skupa, gdje je 70% podataka postavljeno kao skup za učenje, a 30% kao skup za ispitivanje modela, za potrebe provjere utjecaja varijacije parametara algoritama na kvalitetu evoluiranog modela korišten je skup za validaciju. Skup za validaciju stvoren je iz skupa za učenje tako da je skup za učenje podijeljen u 5 preklopa te je uzimanjem po 4 preklopa stvoreno 5 različitih skupova koji zajedno tvore skup za validaciju. Ako se pri validaciji algoritam pokreće više od 5 puta te nije moguće svaki put koristiti drugačiji preklop, ponovno se koristi prvi korišteni skup itd. Zbog ovoga je pri validaciji najčešće potrebno pokretati algoritam 5, 10 ili općenito $5n$ puta, kako bi se algoritam izveo jednako puta na svim podskupovima. Višestruko pokretanje na istom skupu je poželjno kako bi se neutralizirao stohastički utjecaj evolucijskog algoritma. Iako su bila dostupna dva podatkovna skupa, korišten je onaj dostupan iz ISO New England testnog slučaja, budući da on sadrži sve opisane ulazne varijable (uključuje i vlažnost zraka).

6.1. Utjecaj varijacije osnovnih parametara algoritma

Osnovni parametri genetskog programiranja koji se mogu lako varirati su oni koji se odnose na populaciju koja se evoluira. To su redom veličina populacije (odnosno broj jedinki koje se evoluira), broj izmjena generacija (kod turnirske selekcije jednom izmjenom generacija može se smatrati broj iteracija potreban da se zamijeni toliko jedinki kolika je veličina populacije) te vjerojatnost mutacije jedinke. Dodatno je ispitan utjecaj odabranog skupa čvorova stabla na način da se probao reducirati skup korištenih operatora dio po dio te je razmatrana srednja relativna pogreška modela.

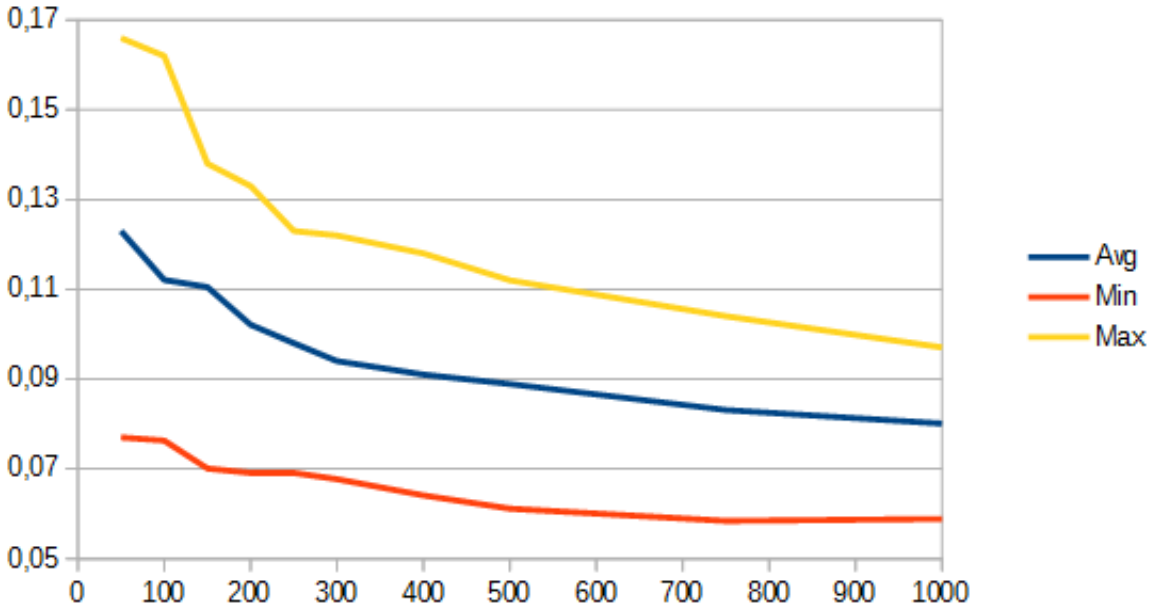
Za razliku od algoritama strojnog učenja, gdje bi se podešavanje parametara moglo raditi tehnikama poput k-struke unakrsne provjere (engl. *k-folded cross validation*), genetsko programiranje ipak je stohastički algoritam, što u kombinaciji sa jako velikim brojem mogućih kombinacija parametara čini većinu metoda unakrsne validacije neupotrebljivom. Zato je napravljen slični postupak, gdje je svaki od parametara variran zasebno te se gleda pogreška modela u ovisnosti o parametru. Pretpostavka ovdje je da je međusobni utjecaj ovih parametara zanemariv, te da nije potrebno testirati sve kombinacije već da je najbolja kombinacija parametara ustvari kombinacija najboljih vrijednosti svakog od ovih parametara.

Dodatna poteškoća ovdje je činjenica da model koji se evoluira ustvari spoj više modela (točan broj ovisi o granulaciji modela, no za potrebe prikaza rezultata korišten je 24-satni model) te je varijacija pogreške unutar onoga što bi se moglo nazvati podmodelima moguće veoma visoka. Zato je za prikaz greške odabran onaj prikazan u prezentaciji [2] gdje se prikazuju zasebno srednja, minimalna i maksimalna pogreška svih satnih modela. Naravno, ni ovo nije potpuno idealan pristup, no daje bolju informaciju nego samo srednja pogreška modela budući da model koji ima previsoku maksimalnu pogrešku nije poželjan, čak iako srednja pogreška može biti prihvatljiva. Za svaku od vrijednosti prikazanih na idućim grafovima algoritam je pokrenut 10 puta te su gledane srednje vrijednosti svih prikazanih linija na ovim pokretanjima.



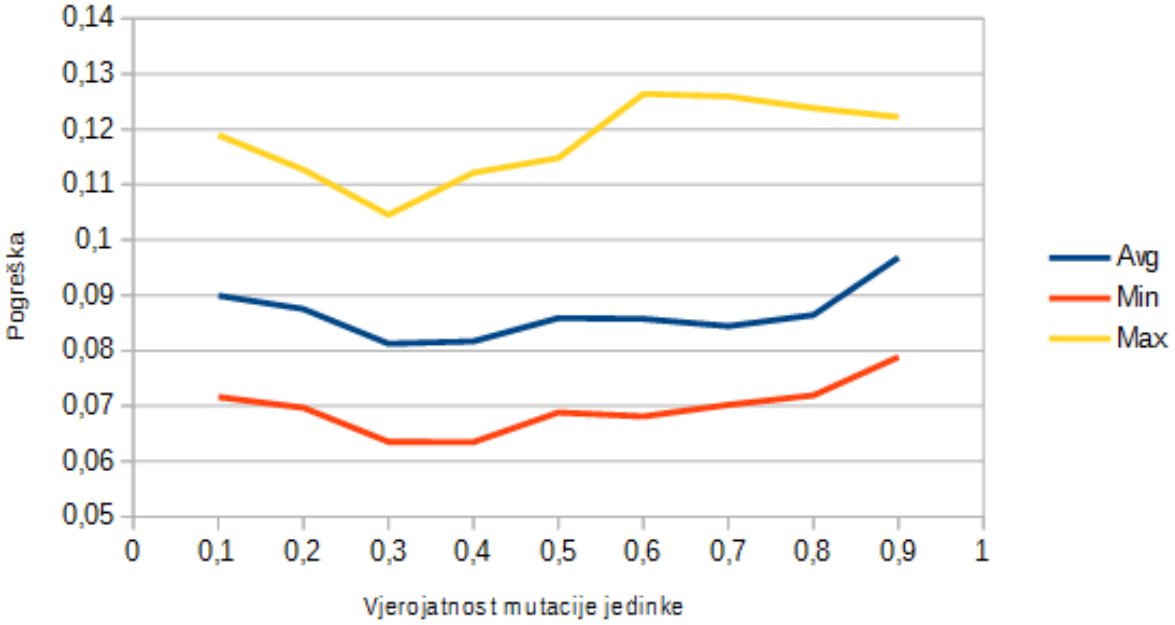
Slika 6.1: Utjecaj veličine populacije na grešku dobivenog modela na skupu za validaciju

Iz slike 6.1 vidljivo je kako povećanje broja jedinki u populaciji smanjuje grešku modela, no nakon što je broj jedinki dostatno velik, smanjivanje srednje pogreške postaje sve manje, iako se smanjuje varijacija greške (minimalna i maksimalna pogreška postaju sve bliže jedna drugoj, a time i srednjoj pogrešci). Za potrebe testiranja utjecaja populacije, uvjet zaustavljanja bio je dostatno velik broj iteracija algoritma (fiksiran na 10^6 zamjena jedinki, odnosno toliko provedenih turnirskih selekcija), te je u sljedećem koraku taj parametar variran kako bi se vidjelo može li se smanjiti.



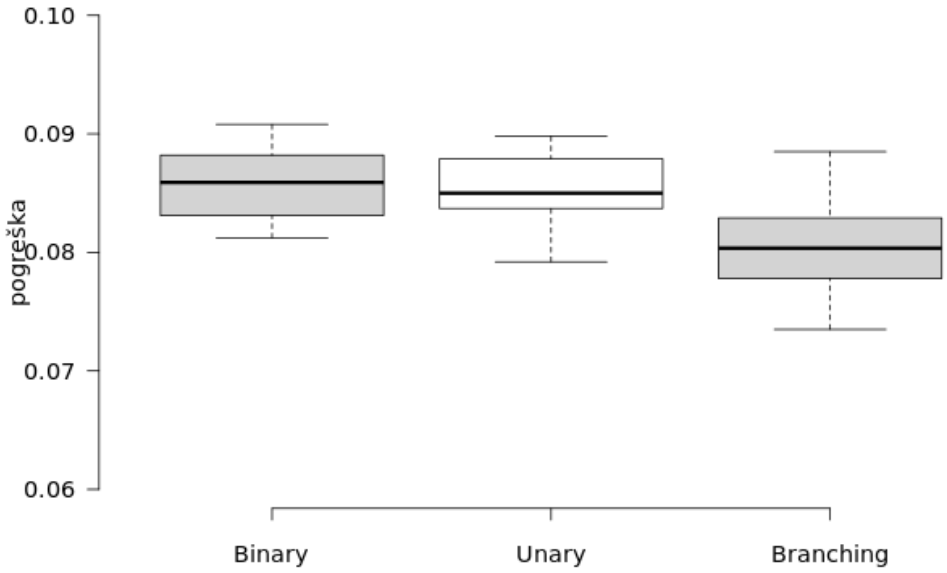
Slika 6.2: Utjecaj broja generacija (iteracija algoritma) na grešku dobivenog modela na skupu za validaciju

Na slici 6.2 prikazana je ovisnost pogreške o broju generacija, odnosno iteracija algoritma. Za razliku od veličine populacije, ovaj parametar je izravno povezan uz vrijeme izvođenja algoritma, a može se uočiti isti trend kao kod populacije - povećanje broja generacija smanjuje pogrešku modela, iako nakon određenog vremena sve manje, dok se povećava vrijeme izvođenja te je potrebno naći točku kompromisa. Također se može uočiti da povećanje broja generacija uz fiksnu veličinu populacije ne smanjuje toliko varijaciju srednje pogreške koliko povećanje veličine populacije to radi. Očito je da je potreban dostatan broj jedinki kako bi se konzistentno pretražio dovoljno velik dio područja rješenja, dok s manjim brojem jedinki velik broj iteracija algoritma ne može to garantirati.



Slika 6.3: Utjecaj vjerojatnosti mutacije na grešku dobivenog modela na skupu za validaciju

Zadnji preostali parametar samog algoritma je vjerojatnost mutacije jedinke te su rezultati promjene ovog parametra prikazani na slici 6.3. Iako se utjecaj samog parametra ne čini prevelikim, iz slike je vidljivo kako se uključivanjem mutacije u algoritam (odnosno dizanjem vjerojatnosti iznad 0.0) ipak smanjuje greška modela. Također, prevelikom vjerojatnošću mutacije ovaj utjecaj se negira, budući da mutacija tada ne donosi samo diverzifikaciju rješenja, već mijenja većinu novonastalih jedinki čime se negira utjecaj križanja kojom nastaju nove jedinke bliske boljim jedinkama. Ovakvo ponašanje algoritma u skladu je s očekivanjem, budući da se inače slično ponašaju operatori mutacije u genetskim algoritmima - uvijek postoji točka gdje je vjerojatnost mutacije dovoljno visoka da se pretražuju različita rješenja, no ne previsoka toliko da se upropasti utjecaj operatora križanja.



Slika 6.4: Utjecaj korištenog skupa funkcijskih čvorova na grešku dobivenog modela na skupu za validaciju

Na slici 6.4 prikazano je ponašanje pogreške u odnosu na to koji se skup funkcijskih čvorova koristi. Sa *Binary* označen je skup čvorova u kojem su samo binarni operatori, odnosno osnovne operacije zbrajanja, oduzimanja, množenja i dijeljenja, dok je sa *Unary* označen skup koji uključuje sve te čvorove, ali i unarne operatore koji su implementirani. Posljednji skup označen je sa *Branching* i on uključuje sve čvorove prijašnja dva skupa, ali i one koji predstavljaju operatore grananja, bilo sa dva ili četiri čvora djeteta. Konačne kombinacije parametara korištenje prikazane su u tablici **??**

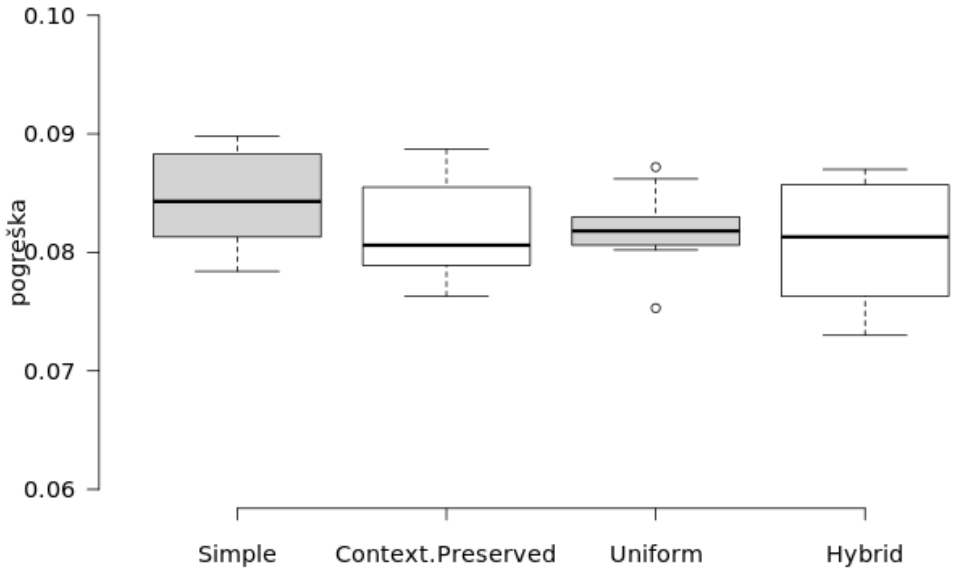
Tablica 6.1: Početne i konačne vrijednosti parametara genetskog programiranja

Parametar	Početna vrijednost	Konačna vrijednost
Veličina populacije	400	800
Broj iteracija algoritma	400	900
Vjerojatnost mutacije	0.4	0.3
Korišteni skup funkcijskih čvorova	Binary (samo binarni operatori)	Branching (svi operatori)

??

6.2. Utjecaj operatora križanja

Iako je kod mutacije testirana samo vjerojatnost kojom se ona događa, za potrebe ovog rada implementirano je više operatora križanja te je bilo potrebno prikazati i njihov utjecaj kako bi se mogao odabrati najbolji za korištenje. Dodana je i opcija korištenja hibridnog operatora križanja, odnosno nasumičnog izbora jednog od različitih operatora. Vjerojatnost odabira može se podesiti za svaki od operatora, a ako se ne podesi, pretpostavlja se uniformna razdioba vjerojatnosti.



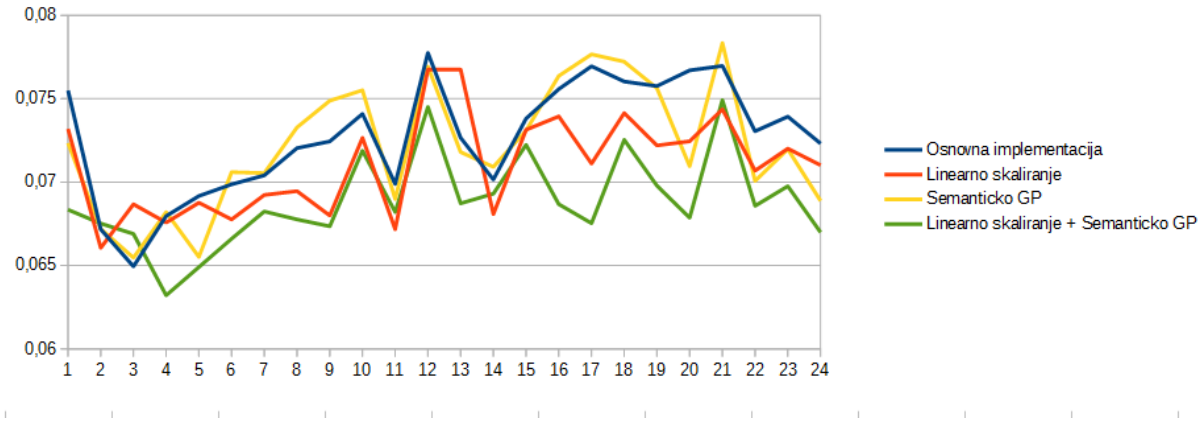
Slika 6.5: Utjecaj operatora križanja na grešku dobivenog modela

Iz slike 6.5 je uočljivo da korišteni operatori križanja nemaju prevelik utjecaj na dobrotu, odnosno srednju pogrešku, dobivenog modela. Ipak, postoji razlika između operatora križanja koji se koriste. Najbolje rezultate algoritam dobiva korištenjem operatora križanja s očuvanjem konteksta, ali i hibridnog operatora križanja, što također i daje veću raznolikost samom algoritmu, a umjerena diverzifikacija rješenja u sklopu operatora križanja također je uvijek poželjna u genetskim algoritmima. Najgore rezultate daje jednostavno križanje, dok je prednost korištenja uniformnog križanja to što pogreška dobivena korištenjem samo uniformnog križanja najmanje varira u usporedbi sa svim drugim operatorima križanja.

Nakon razmatranja ovih rezultata, u svim daljnjim testovima i implementacijama korišten je hibridni operator križanja sa uniformnom razdiobom vjerojatnosti uporabe pojedinog operatora križanja. Takav operator vremenski je u praksi jednako efikasan kao i svi drugi operatori, a daje najbolje rezultate i teoretski omogućava korištenje bilo kojeg od implementiranih operatora (iako ne nužno uvijek).

6.3. Utjecaj primjene semantičkog genetskog programiranja i linearnog skaliranja

Semantičko GP i linearno skaliranje implementirano u sklopu rada testirani su zajedno jer služe sličnoj funkciji - smanjenju pogreške modela, ali također i većoj smislenosti evoluiranih modela, odnosno lakšoj interpretaciji funkcija koje predviđaju vrijednost potrošnje. Rezultati primjene svakog od ovih postupaka te njihove kombinacije prikazani su na slici 6.6. Ipak, nemoguće je grafički prikazati utjecaj parametara na smislenost ili interpretabilnost modela te se može samo iz nekoliko primjera zaključiti da je stvarno slučaj da dodavanje čvorova grananja u kombinaciji s ovim postupcima čini dobivene modele lakšima za ljudsku interpretaciju (što može biti dodatna prednost u usporedbi sa postupcima poput neuronskih mreža ili regresije potpornim vektorima).



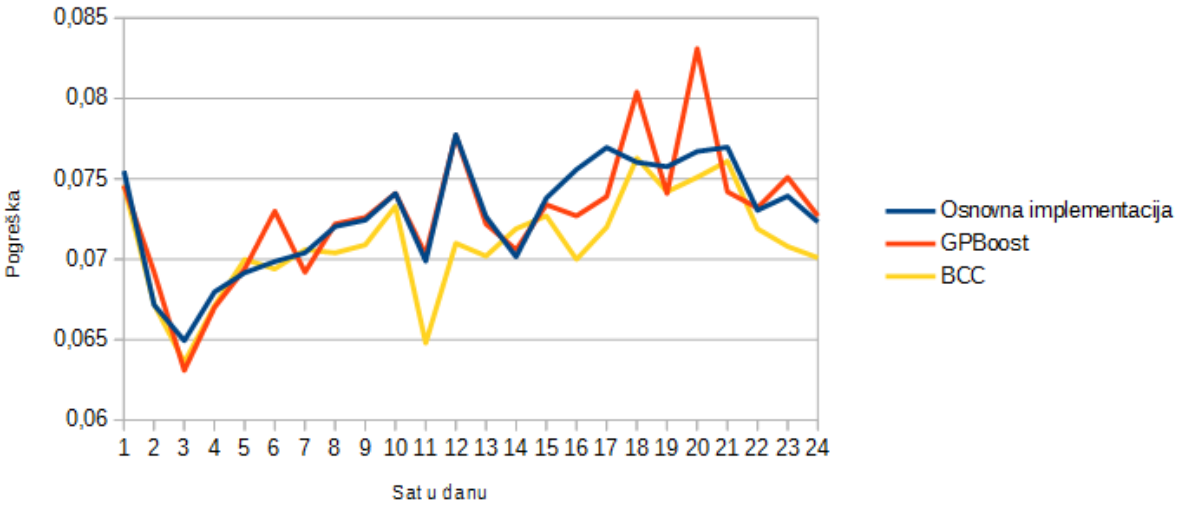
Slika 6.6: Utjecaj implementiranih postupaka linearnog skaliranja i semantičkog genetskog programiranja

Gledajući rezultate može se zaključiti da linearno skaliranje i semantičko GP daju poboljšanja rezultata, odnosno smanjenje pogreške, ali njihov efekt i nije toliko značajan koliko je očekivano. Na nekim od podmodela po satima, osnovna implementacija zapravo daje manju pogrešku nego modeli koji koriste linearno skaliranje ili semantičko GP (iako kombinacija linearnog skaliranja i semantičkog GP u gotovo svakom slučaju dalje bolje rezultate). Gledajući samo srednju grešku modela, linearno skaliranje daje značajnija poboljšanja rezultata nego semantičko genetsko programiranje, čiji je utjecaj u nekim slučajevima zanemariv. Ipak, semantičko genetsko programiranje daje veću smislenost modelima, te je potrebno u algoritmu imati određenu informaciju o smislu svake varijable i dobivenog rješenja kako bi se dobiveni modeli mogli lakše interpretirati i analizirati.

Ovo nije moguće dobiti uporabom samo linearnog skaliranja, koje jednostavno dobivenu funkciju pokušava bolje prilagoditi podatcima pomakom po rezultantnoj osi. U prosjeku se najbolji rezultati ipak dobivaju kombinacijom i linearnog skaliranja i semantičkog genetskog programiranja, a za razliku od prije isprobanih parametara, ovdje je situacija slična kao kod testiranja različitih operatora križanja - dodavanje ovih metoda ne povećava značajno vremensku složenost algoritma, a napredak je dovoljno značajan da se isplati uvijek ih imati u algoritmu.

6.4. Utjecaj algoritama skupnog učenja

Implementirani algoritmi *GPBoost* i *BCC* ispitani su u okviru koliko utječu na srednju pogrešku modela kroz sate te su na slici 6.7 uspoređeni sa osnovnom implementacijom algoritma. Može se vidjeti da iako smanjenje pogreške nije značajno, svaki od algoritama ipak može smanjiti pogrešku modela, što u slučaju kad je relativna pogreška već dostatno malena može biti dosta značajno.



Slika 6.7: Utjecaj primjene algoritama skupnog učenja

Iz slike 6.7 vidljivo je više trendova. Kao prvo, *GPBoost* u određenim modelima, odnosno satima, daje bolje rezultate nego osnovna implementacija GP, no postoji previše situacija gdje ova varijanta zapravo daje lošije rezultate. Razlog ovoga možda se može naći u činjenici da algoritam originalno nije prilagođen za probleme regresije već klasifikacije. S druge strane, *BCC* daje vidljivo bolje rezultate te je njegova primjena

opravdana boljim rezultatima (s druge strane, ovaj algoritam je originalno prilagođen regresijskim problemima pa se nešto ovakvo moglo i očekivati).

Ipak, uz prikazane rezultate algoritama skupnog učenja potrebno je u obzir uzeti i vrijeme izvođenja svake od varijacije algoritama. Kao što je već rečeno u opisu algoritma, vremenska složenost ovih algoritama znatno je veća od složenosti osnovnog GP algoritma zbog potrebe za stalnom reevaluacijom funkcije dobrote. Kod računanja pogreške ovih modela uočeno je znatno dulje vrijeme izvođenja algoritma te je u tablici 6.2 prikazana usporedba srednjeg vremena izvođenja (odnosno evolucije jednog satnog modela) osnovne implementacije te *boosting* varijanti u ovisnosti o različitim maksimalnim dubinama stabla. Veća dubina stabla povećava vrijeme evaluacije funkcije dobrote budući da je evaluacija stabla potencijalno dulja.

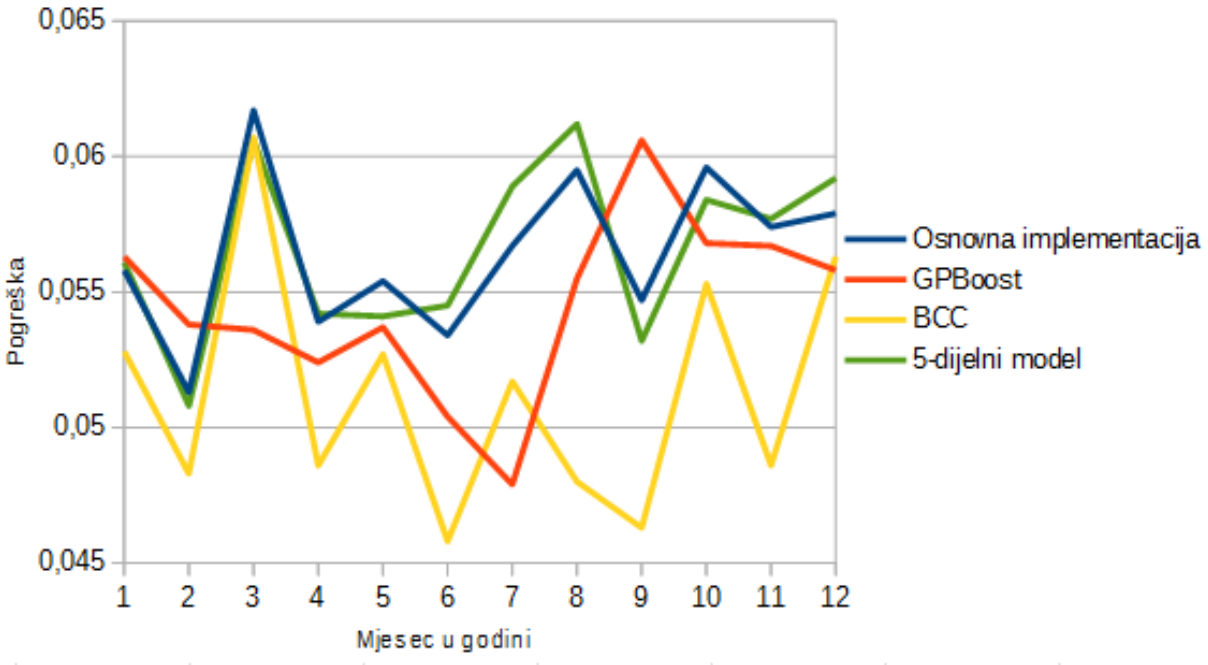
Tablica 6.2: Vrijeme izvođenja varijanti algoritma u ovisnosti o dubini stabla

Maksimalna dubina stabla	Osnovna implementacija	GPBoost	BCC
4	1.21 min	5.18 min	5.37 min
6	4.46 min	17.91 min	16.42 min
8	8.13 min	47.83 min	48.12 min

Iz tablice je očito da je složenost ovih varijanti za red veličine veća od osnovne implementacije, što stvara problem budući da evolucija dubljih stabala postaje preteška da bi se model mogao ponovno naučiti unutar vremena između dva predviđanja, koje je jedan sat (budući da su uzastopne vrijednosti vremenskog slijeda međusobno udaljene jedan sat). Zbog toga su ovi algoritmi upotrebljivi samo u slučaju da model nije potrebno ponovno trenirati, čime se gubi mogućnost prilagođavanja modela po potrebi. Ipak, kako je pogreška tako dobivenih modela nešto manja nego ona osnovne implementacije, otvara se mogućnost za uporabom ovih algoritama ako je glavni prioritet preciznost modela.

6.5. Greška modela po mjesecima

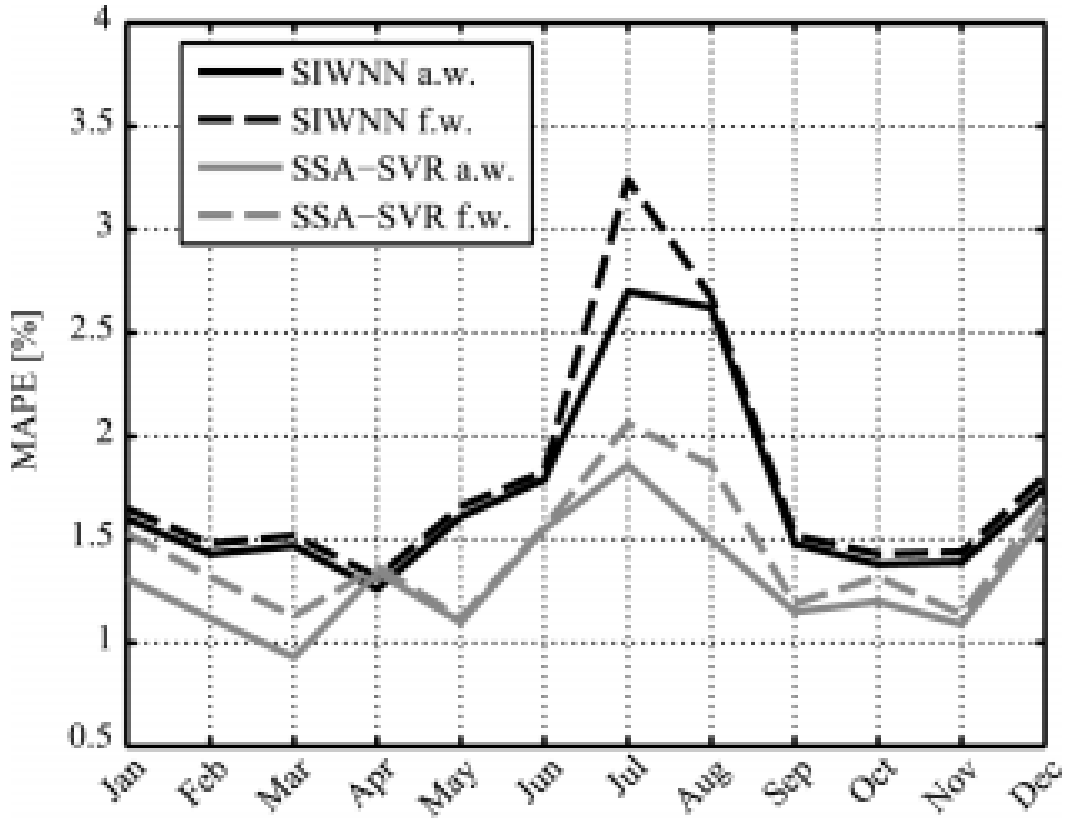
Nakon što su isprobani svi parametri, iz prethodnih grafova se otprilike može dobiti najbolja (ili barem dovoljno dobra) kombinacija svih parametara. Ova kombinacija konačno je primijenjena na problemu kako bi se izračunala srednja relativna pogreška modela, ali ovaj put umjesto po satima pogreška je prikazana po mjesecima. U osnovi razlika između ova dva prikaza nije toliko velika, ali budući da većina postojećih radova grešku modela prikazuje baš na ovaj način, potrebno je rezultate prikazati na isti način kako bi dobiveni model bio usporediv sa modelima opisanim u drugim radovima. Kao što je već rečeno, model koji se koristi ovdje, isto kao svi korišteni u prikazu rezultata, koriste granulaciju modela po satima, odnosno svaki model se sastoji od 24 različita modela. Posebno je na grafu pokazan i rezultat korištenja osnovne implementacije, ali 5-dijelnog modela opisanog ranije, gdje se ne koristi 24 različita modela, već samo njih 5. Na grafu se može vidjeti da su rezultati ovog modela usporedivi sa onim koji koristi granulaciju po satima, no ipak nešto lošiji. Kao što je rečeno unaprijed, ova konačna provjera rezultata modela je na skupu za ispitivanje, koji čini 30% ukupnog skupa podataka.



Slika 6.8: Srednja relativna pogreška modela po mjesecima u godini na skupu za ispitivanje

Konačni rezultati dobiveni kombinacijom najboljih mogućih parametara, čak i bez metoda skupnog učenja, definitivno su usporedivi sa rezultatima dobivenim u ostalim

radovima (gdje su rezultati iz [1] prikazani na slici 6.9), ali ipak još nisu na razini najboljih rezultata. U kombinaciji sa metodama skupnog učenja, preciznije sa *BCC*, koji je pokazao bolje rezultate u usporedbi sa originalnim algoritmom i onim koji koristi *GPBoost*, postižu se još bolji rezultati. Ipak, vrijeme izvođenja takvog algoritma značajno je veće od vremena izvođenja algoritma koji daje ovakve rezultate, a uzme li se u obzir i namijenjena uporaba algoritma, predugo vrijeme izvođenja može biti neprihvatljivo.



Slika 6.9: Srednja relativna pogreška modela opisanog i implementiranog u [1] po mjesecima

7. Rasprava

Rezultati su potvrdili neke od pretpostavki danih u radu, no i predstavili dodatne probleme poput upitne isplativosti *boosting* algoritama. Naime, rezultati dobiveni pri variranju osnovnih parametara modela slijede uobičajeno ponašanje genetskih algoritama - povećanje broja generacija i broja jedinki poboljšavaju dobrotu konačnog rješenja, do određene granice gdje poboljšanja postaju zanemariva u odnosu na povećanje vremenske složenosti samog algoritma. Također, preniska vrijednost vjerojatnosti mutacije ne pomaže algoritmu, dok previsoka ustvari odmaže nalaženju rezultata. Također, uvođenje više funkcijskih čvorova smanjuje grešku konačnog modela, iako se prihvatljivi rezultati postižu već i sad prilično reduciranim skupom funkcija.

Kod operatora križanja najgore rezultate daje korištenje samo jednostavnog križanja budući da se populacija često dovede u situaciju da sastoji mnogo duplikata jedne jedinke. Iako generalno operatori križanja ne daju jako različite rezultate, najbolji rezultati dobiveni su korištenjem hibridnog modela, odnosno svih implementiranih operatora križanja, gdje se pri svakom križanju nasumično bira jedan od operatora. Ovo je u skladu sa većinom radova u kojima se implementiraju različiti operatori križanja za genetske algoritme.

Semantičko GP i linearno skaliranje ne daju rezultate kakvi su se očekivali u smislu smanjivanja greške modela, no budući da nije uočeno nikakvo povećanje vremena izvođenja algoritma, njihovo uključivanje u proces učenja definitivno je poželjno kako bi evoluirani modeli bili semantički smisleniji i kako bi se fokus posvetio evoluiranju prikladne funkcije, uz što manju mogućnost prejake prilagodbe modela. Većinom će ovo koristiti u slučaju da je potrebno napraviti analizu modela (budući da model nije crna kutija poput neuronskih mreža i sličnih modela), no linearno skaliranje je ustvari metoda jednostavne linearne regresije, što znači da bi se možda dodavanjem kompleksnijih regresijskih metoda u kombinaciji sa genetskim programiranjem mogli dobiti slični ili bolji rezultati.

Algoritmi skupnog učenja daju poboljšanja u smislu smanjivanja greške modela, no prikazano je da je vrijeme učenja modela kada se oni koriste za red veličine veće što može otežati primjenjivost tog pristupa. Ako vrijeme izvođenja nije bitno ili se može dozvoliti dulje vrijeme za učenje, oni se mogu koristiti, no za brzo kratkoročno predviđanje nisu primjenjivi. Ipak, predstavljaju dosta velik pomak u smjeru kombiniranja metoda strojnog učenja i genetskog programiranja kako bi se poboljšali rezultati dobiveni genetskim programiranjem, te se otvara mogućnost primjene prvo preostalih metoda skupnog učenja (*bagging* metode), a i ostalih kompleksnijih metoda u kombinaciji s genetskim programiranjem.

8. Zaključak

U sklopu rada opisane su i implementirane metode genetskog programiranja na problem predviđanja vrijednosti vremenskih sljedova, na primjeru predviđanja potrošnje električne energije. Osnovni algoritam koji koristi samo genetsko programiranje pokazao je zadovoljavajuće rezultate u usporedbi sa postojećim modelima, no isprobane su i različite optimizacije dobivenih modela.

Rezultati dobiveni primjenom genetskog programiranja zadovoljavajući su i usporedivi sa postojećim rezultatima dobivenim metodama strojnog učenja, no važan fokus ovog rada predstavlja i dio gdje se ova dva pristupa kombiniraju za izradu boljeg modela. Pokazano je kako algoritmi skupnog učenja te linearno skaliranje (koje je u osnovi jednostavna linearna regresija) poboljšavaju modele dobivene genetskim programiranjem. U kombinaciji sa različitim varijacijama osnovnog modela genetskog programiranja, od kojih je najzanimljivije vjerojatno semantičko genetsko programiranje, dobiva se velik broj kombinacija parametara čijim se mijenjanjem mogu dobiti modeli koji se međusobno dosta razlikuju u smislu složenosti, preciznosti i/ili mogućnosti ljudske interpretacije.

Dodatno, programska implementacija napravljena za potrebe rada iznimno je modularna, te su izmjene i daljnje nadogradnje lako moguće. Isto tako, primjena postojećih algoritama na druge probleme predviđanja vremenskih sljedova moguća je, no vjerojatno uz modifikacije koje se tiču granulacije modela i slično, budući da je vjerojatno da za neke druge probleme ovo nije potrebno, ili pak da je potrebna još finija granulacija.

Opisane metode poboljšale su osnovne modele, no mjesta za napredak i dalje ima. Jedna od osnovnih metoda u strojnom učenju koja se koristi za probleme kod kojih je teže naučiti modele je odabir značajki (engl. *feature selection*), gdje se odabirom, uklanjanjem i/ili modifikacijom ulaznih varijabli pokušava naći najbolji skup ulaznih

varijabli za učenje modela. Iako genetsko programiranje implicitno u sebi sadrži odabir značajki (budući da se za razliku od metoda strojnog učenja ne koristi cijeli ulazni vektor, već samo one varijable koje se nasumično odaberu kao terminalne vrijednosti stabla), moguće je daljnje poboljšanje uporabom nekog postojećeg algoritma odabira značajki. Također u obzir dolazi i primjena drugih metoda koje se koriste u strojnom učenju, dok god su primjenjive na genetsko programiranje bez većih poteškoća (budući da su *boosting* algoritmi pokazali probleme što se tiče vremenske složenosti).

LITERATURA

[1] Ervin Ceperic, Vladimir Ceperic, i Adrijan Baric. A strategy for short-term load forecasting by support vector regression machines. *Power Systems, IEEE Transactions on*, 28(4):4356–4364, 2013.

[2] Mirela Ćosić. Kratkoročna prognoza potrošnje električne energije. 2014.

[3] Matteo De Felice i Xin Yao. Short-term load forecasting with neural network ensembles: A comparative study [application notes]. *Computational Intelligence Magazine, IEEE*, 6(3):47–56, 2011.

[4] Luzia Vidal de Souza, Aurora TR Pozo, Anselmo C Neto, i Joel MC da Rosa. Genetic programming and boosting technique to improve time series forecasting. 2009.

[5] Yoav Freund, Robert Schapire, i N Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.

[6] Maarten Keijzer. Improving symbolic regression with interval arithmetic and linear scaling. U *Genetic programming*, stranice 70–82. Springer, 2003.

[7] Maarten Keijzer i Vladan Babovic. Dimensionally aware genetic programming. U *Proceedings of the Genetic and Evolutionary computation Conference*, svezak 2, stranice 1069–1076, 1999.

[8] Maja Kontrec. Ocjena učinkovitosti operatora križanja u genetskom programiranju. 2014.

[9] John R Koza. *Genetic programming: on the programming of computers by means of natural selection*, svezak 1. MIT press, 1992.

[10] Marin Matijaš. Predvićanje potrošnje električne energije regresijom zasnovanom na potpornim vektorima.

[11] Marin Matijaš. Electric load forecasting using multivariate meta-learning. 2013.

[12] Payman Shamsollahi, KW Cheung, Quan Chen, i Edward H Germain. A neural network based very short term load forecaster for the interim iso new england electricity market system. U *Power Industry Computer Applications, 2001. PICA 2001. Innovative Computing for Power-Electric Energy Meets the Market. 22nd IEEE Power Engineering Society International Conference on*, stranice 217–222. IEEE, 2001.

[13] Marko Đurasević. Optimizacija raspoređivanja u okruženju nesrodnih strojeva. 2014.

Kratkoročno predviđanje potrošnje električne energije

Sažetak

Problem kratkoročnog predviđanja potrošnje električne energije jedan je od osnovnih primjera prognoze vrijednosti vremenskih sljedova te ima značajnu ulogu u električnoj opskrbi. Svrha ovog rada je demonstrirati primjenu metoda genetskog programiranja na problemu kratkoročnog predviđanja potrošnje električne energije te usporedba rezultata sa postojećim dobivenim primjenom drugih metoda. Uz osnovnu implementaciju genetskog programiranja dodano je i linearno skaliranje te semantičko genetsko programiranje za poboljšanje dobivenog rješenja te strože poštivanje semantike prisutne u ulaznim podatcima. Najveći dio optimizacije dobivenih modela implementacija je metoda skupnog učenja primijenjenih na genetsko programiranje, odnosno algoritama *GPBoost* i *BCC*. Prikazani su rezultati implementiranog algoritma te njegovih različitih varijacija uz utjecaj promjene parametara algoritma na kvalitetu najboljeg dobivenog rješenja. Konačno su dana razmatranja rezultata te kraće ideje za daljnja poboljšanja rezultata.

Ključne riječi: genetsko programiranje, kratkoročno predviđanje potrošnje, semantičko GP, skupno učenje, linearno skaliranje

Short-Term Load Forecasting

Abstract

The short-term electric energy load forecasting problem is one of the basic examples of time series forecasting and has an important role in electric power distribution. The goal of this paper is to demonstrate the application of genetic programming methods on the problem of short-term load forecasting and to compare results with the existing ones gained when using other methods. Together with the basic implementation of genetic programming linear scaling and semantic genetic programming was added to improve the current solution and enforce the stricter following of the semantics which exist in the input data. The largest part of optimization of current models was the implementation of ensemble learning methods applied to genetic programming, namely algorithms *GPBoost* and *BCC*. Results of the implemented algorithm and its different variations are presented together with the effect that changing of algorithm parameters has on the quality of the best solution. Finally, further consideration of results is given as well as short ideas for further improvement of the results.

Keywords: genetic programming, short-term load forecasting, semantic GP, ensemble learning, linear scaling