TEHNIČKO REŠENJE

IP jezgra za hardversko generisanje ansambala stabala odluka

M-85: Prototip, nova metoda, softver**,** standardizovan ili atestiran instrument, nova genetska proba, mikroorganizmi

Autori:

Rastislav Struharik, Fakultet tehničkih nauka (FTN), Novi Sad,
Ladislav Novak, Fakultet tehničkih nauka (FTN), Novi Sad.

1. Kratak opis

Hardverske implementacije stabala odluka mogu predstavljati jedino rešenje u slučajevima kada je potrebno izvršiti klasifikaciju instance u vrlo kratkom vremenu, ili kada je potrebno adaptivno formiranje prediktivnog modela, u toku rada sistema. Ovo su tipični zahtevi koji se sreću prilikom projektovanja savremenih a pogotovo budućih *embedded* sistema. Imajući u vidu ove činjenice, razvijena su IP jezga za hardversko generisanje ansambala stabala odluka koja u mnogome olakšavaju integraciju i rada sa stablima odluka prilikom projektovanja *embedded* sistema.

Tehničke karakteristike:

IP jezgra za hardversko generisanje ansambala stabala odluka modelovana su pomoću standardnog jezika za specifikaciju hardvera, VHDL. Razvijeni modeli su parametrizovani što omogućava jednostavno prilagođavanje arhitekture trenutnim potrebama korisnika.

Tehničke mogućnosti:

Korišćenjem konfiguracionih parametara definisanih unutar VHDL modela arhitektura za hardversko generisanje ansambala stabala odluka (broja stabala u ansamblu, broj bitova koji se koristi za predstavu vednosti atributa problema, koeficijenata razdvajajućih hiperpovrši, ciljnih klasa; broja atributa preko kojih je opisan klasifikacioni problem; maksimalnog broja čvorova u realizovanom stablu odluke, itd.) moguće je prilagoditi VHDL model potrebama tekuće aplikacije. Pilikom sinteze hardvera ovi parametri se koriste kako bi se automatski generisala optimalna hardverska implementacija za tekuću aplikaciju.

Realizator:

Fakultet tehničkih nauka – FTN.

Korisnik:

Fakultet tehničkih nauka – FTN,

Podtip rešenja:

Softver -M85

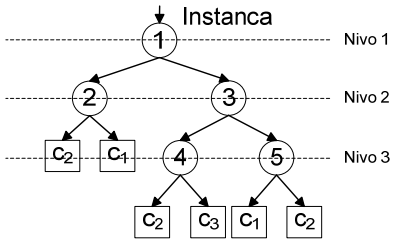
Projekat u okviru koga je realizovano tehničko rešenje:

Program istraživanja u oblasti tehnološkog razvoja za period 2011.-2014.

Tehnološka oblast:	Elektronika, telekomunikacije i informacione tehnologije
Rukovodilac projekta:	dr Ljiljana Živanov, redovni profesor
Naziv projekta:	Nove generacije ugrađenih elektronskih komponenti i sistema u neorganskim i organskim tehnologijama za uređaje široke potrošnje
Broj projekta:	TR 32016

2. Stanje u svetu

Posmatrajmo binarno stablo odluke prikazano na slici 2.1. Ovo stablo bi moglo biti rezultat izvršavanja nekog od algoritama za formiranje stabala odluke [1], [2].



Slika 2.1 Binarno stablo odluke

Stablo odluke klasifikuje instance prolazeći kroz stablo počevši od korena do nekog od listova, kome je asocirana jedna od mogućih klasa, i na taj način obezbeđuje klasifikaciju tekuće instance. Svaki čvor u stablu specificira test nekog od atributa date instance (mada se test može sastojati i od više od jednog atributa), a svaka od grana koja polazi od tog čvora predstavlja jedan od mogućih ishoda testa. Instanca se klasifikuje počevši od korena stabla, izvodeći test koji je pridružen korenu, a zatim se kreće duž grane koja odgovara rezultatu izvedenog testa. Ovaj proces se zatim rekurzivno ponavlja uzimajući podstablo čiji koren predstavlja čvor do kojeg se stiže krećući se duž odabrane grane.

Kao što je rečeno algoritmi za formiranje stabala odluka mogu da rade sa problemima koji su opisani kategoričkim i numeričkim atributima. Pošto su praktični problemi najčešće predstavljeni pomoću numeričkih atributa, detaljno će biti razmotreni mogući tipovi testova koji se mogu koristiti u ovom slučaju.

Ako se u svakom čvoru koristi test samo jednog atributa onda taj test u opštem slučaju ima oblik

$$A_i > a_i \tag{2.1}$$

Ovaj test ekvivalentan je hiperravni koja je ortogonalna u odnosu na osu koja predstavlja atribut A_i. Stabla koja koriste ovakve testove zovu se *stabla odluka sa ortogonalnim hiperravnima*. Ovo je najčešći tip stabala odluka koja se koriste u praksi. Kada se formira stablo odluke sa ovakvim testovima, ono će izvršiti particiju hiperprostora koristeći samo ovakve, ortogonalne hiperravni.

Međutim, moguće je formirati i znatno složenije testove. Testovi mogu da predstavljaju linearnu kombinaciju atributa problema,

$$\sum_{i=1}^{n} a_i A_i + a_{n+1} > 0 \tag{2.2}$$

Kao što se može videti iz prethodne jednačine ovakvi testovi zapravo definišu hiperravni potpuno proizvoljnog položaja u hiperprostoru definisanom atributima problema. Stabla koja koriste ovakve testove zovu se *stabla odluka sa neortogonalnim hiperravnima*. Ovakvi testovi mogu biti vrlo pogodni ako je priroda problema takva da se particija instanci može znatno lakše izvesti pomoću neortogonalnih hiperravni, jer će tada formirana stabla odluka biti znatno manja nego u slučaju da se koriste testovi samo pojedinačnih atributa. Sa druge strane, problem nalaženja optimalnog položaja neortogonalne hiperravni je znatno teži jer on zahteva pretragu u $n+1$ dimenzionalnom prostoru koeficijenata za razliku od slučaja kada se koriste ortogonalne hiperravni kada je potrebno vršiti pretragu u 2 dimenzionalnom prostoru pretrage. U literaturi je pokazano da je problem pronalaženja optimalnog položaja neortogonalne hiperravni NP težak problem, [3].

Na kraju, mogu se koristiti još opštiji testovi. Najopštiji oblik testa u nekom od čvorova stabla ima sledeći oblik.

$$f(A_1, A_2, ..., A_n) > 0 \tag{2.3}$$

Funkcija f može biti bilo koja funkcija n atributa. Sada se prilikom particije hiperprostora atributa koristi hiperpovrš definisana funkcijom f. Ako je funkcija f nelinearna funkcija, stabla koja koriste takve testove se zovu *stabla odluka sa nelinearnim hiperpovršima*.

Za razliku od ortogonalnih hiperpovrši koje se mogu opisati jednoznačno korišćenjem izraza (2.2), nelinearne hiperpovrši se ne mogu jednoznačno opisati. U ovom radu razmatrane su nelinearne hiperpovrši bazirane na korišćenju polinoma drugog i trećeg reda.

$$\sum_{i=1}^{n} a_i A_i^2 + \sum_{i=1}^{n} a_i A_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} a_{i,j} A_i A_j + c > 0$$
$$\sum_{i=1}^{n} a_i A_i^3 + \sum_{i=1}^{n} a_i A_i^2 + \sum_{i=1}^{n} a_i A_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} a_{i,j} A_i A_j + \tag{2.4}$$
$$+ \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{i,j} A_i^2 A_j + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \sum_{k=j+1}^{n} a_{i,j,k} A_i A_j A_k + c > 0$$

Stablo prikazano na slici 2.1 sadrži ukupno pet čvorova i šest listova raspoređenih u četiri nivoa. U svakom od čvorova definisan je po jedan test atributa klasifikacionog problema. Ovi testovi mogu biti ortogonalni, (2.1), neortogonalni, (2.2), ili nelinearni, (2.4). Instanca problema koju je potrebno klasifikovati (u slučaju stabla sa slike 2.1 u jednu od tri moguće klase) dovodi se u koren stabla. U zavisnosti od ishoda testa specificiranog u korenu stabla, instanca se prosleđuje jednom od dva naslednika (čvorovi 2 i 3). Čitav postupak se zatim ponavlja sve dok se ne dođe do nekog od listova stabla. U tom trenutku instanca se klasifikuje u klasu koja je pridružena posmatranom listu. Tipično se čitav ovaj postupak realizuje u vidu programa koji se zatim izvršava na nekom od procesora opšte namene.

Većina algoritama za formiranje stabala odluka predstavljaju varijacije osnovnog algoritma koji koristi „*greedy*“ pretragu kroz prostor svih mogućih stabala. Glavni predstavnici ovog načina formiranja stabala odluka su dva algoritma predložena od strane Quinlan-a, ID3 algoritam [1] i njegov naslednik C4.5 [2]. U nastavku će biti prikazan ID3 algoritam formiranja stabla odluke.

koren = ID3 (*trening_skup, atributi*)

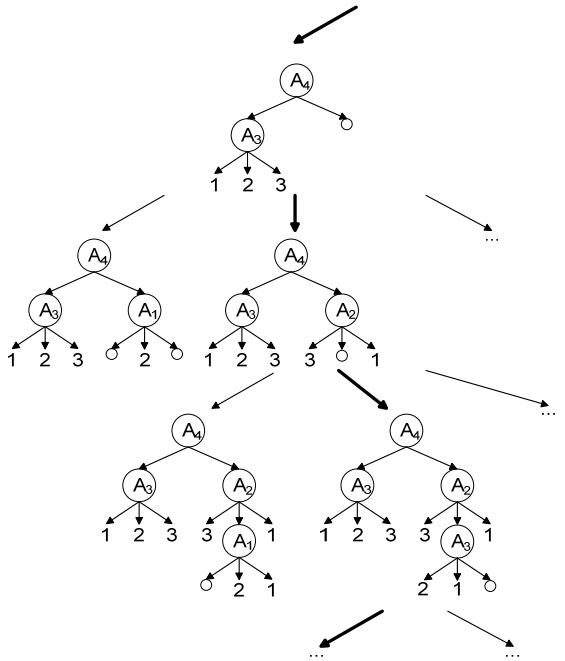
Ulazi:	
trening_skup	*Skup trening instanci. Svaka od instanci je predstavljena pomoću niza vrednosti atributa,* ***d****, praćenog brojem klase kojoj ta instanca pripada*
atributi	*Lista atributa pomoću kojih je opisan problem. Lista ima ukupno* ***d*** atributa.
Izlazi:	
koren	*Koren formiranog neortogonalnog stabla odluke*

- Kreiraj čvor *koren*
- Ako sve instance iz skupa *trening_skup* pripadaju istoj klasi, vrati stablo sa jednim čvorom, *koren*, kome je pridružen broj klase kojoj pripadaju trening instance.
- Ako je skup *atributi* prazan, vrati stablo sa jednim čvorom, *koren*, kome je pridružen broj klase kojoj pripada najviše instanci iz skupa *trening_skup*
- Inače
  - $A \leftarrow$ atribut iz skupa *atributi* za koji *funkcija za merenje kvaliteta testova* ima maksimalnu vrednost
  - Test u čvoru *koren* $\leftarrow A$
  - Za svaku od mogućih vrednosti v_i atributa A,
    - Dodaj novu granu koja polazi iz čvora *koren*, a kojoj odgovara test $A = v_i$
    - Formiraj podskup $trening\_skup_{v_i}$ koji čine sve instance kod kojih je vrednost atributa A jednaka v_i
    - Ako je skup $trening\_skup_{v_i}$ prazan
      - Dodaj list kome je pridružen broj klase kojoj pripada najviše instanci iz skupa *trening_skup*
      - Inače dodaj podstablo ID3($trening\_skup_{v_i}$, *atributi*-$\{A\}$)
- Vrati *koren*

Algoritam 2.1 ID3 algoritam za formiranje stabla odluke

Kao što se može videti analizom algoritma, ID3 algoritam predstavlja „*greedy*“ algoritam koji formira stablo počevši od korena, pri čemu u svakom čvoru bira atribut koji najbolje klasifikuje trening instance pridružene tom čvoru. Ovaj proces se nastavlja sve dok se ne formira stablo koje ispravno klasifikuje sve instance iz trening skupa, ili dok se ne iskoriste svi raspoloživi atributi.

Razmotrimo sada prostor i strategiju pretrage ID3 algoritma. Prostor pretrage ID3 algoritma čini skup svih mogućih stabala odluka. ID3 pretražuje ovaj prostor polazeći od praznog stabla i razmatrajući sve složenija stabla u potrazi za stablom koje će ispravno klasifikovati sve trening primere. Funkcija koja rukovodi ovom pretragom jeste gore navedena *funkcija za merenje kvaliteta testova*. Ova pretraga je prikazana na slici 2.2.



Slika 2.1 Način pretrage prostora hipoteza pomoću ID3 algoritma

Na osnovu analize prostora i strategije pretrage mogu se uočiti sledeće mogućnosti i ograničenja ID3 algoritma.

Prostor pretrage ID3 algoritma čine sva moguća stabla odluka i on čini kompletan prostor svih funkcija sa diskretnim vrednostima i konačnim brojem atributa. S obzirom da se svaka od ovih funkcija može predstaviti nekim stablom odluke, ID3 izbegava jedan od glavnih rizika koji se javlja u radu sa postupcima koji vrše pretragu nekompletnih prostora, da prostor pretrage ne sadrži ciljnu funkciju.

ID3 čuva samo jedno, tekuće stablo dok vrši pretragu prostora rešenja. Ovo je ozbiljno ograničenje, jer na primer, ID3 nema načina da utvrdi koliko alternativnih stabala odluka je konzistentno sa trening podacima.

ID3 ne vrši nikakav „*backtracking*“ tokom svoje pretrage. Kada se u nekom čvoru odabere atribut koji će biti testiran, ID3 se nikada više ne može vratiti na taj čvor da bi preispitao svoju odluku. Zbog toga ID3 može konvergirati ka lokalno optimalnim rešenjima. Lokalno optimalno rešenje odgovara stablu odluke koje će biti odabrano tokom puta pretrage koji se ispituje. Ali ovo lokalno optimalno stablo može biti lošije od stabala na koja bi ID3 algoritam naišao tokom nekog drugačijeg pravca pretrage.

ID3 koristi sve raspoložive instance iz tekućeg trening skupa u svakom koraku pretrage da bi doneo statistički zasnovane odluke u cilju poboljšanja tekućeg stabla. Ova činjenica omogućava da formirano stablo bude mnogo manje osetljivo na greške koje mogu biti prisutne u pojedinim trening instancama.

Na kraju, interesantno je razmotriti način na koji ID3 bira stabla tokom svog rada, odnosno koja je strategija pretrage ID3 algoritma. ID3 tokom svog rada preferira kraća u odnosu na duža stabla i stabla koja atribute sa najvećim vrednostima *funkcije mere kvaliteta testova* postavljaju bliže korenu stabla.

Kao što se moglo videti iz predstavljenog ID3 algoritma za formiranje stabala odluka, centralno mesto u algoritmu zauzima izbor testova za svaki od čvorova stabla. Kao što je već rečeno kompletno stablo vrši particiju hiperprostora na regione koji sadrže samo instance iz jedne klase. Prilikom formiranja stabla, svrha dodavanja novih čvorova jeste profinjenje trenutne particije hiperprostora sa ciljem minimizacije mere „nečistoće“ trening skupa. Alternativno, mogu se definisati i mere „dobrote“ trenutne podele, u kom slučaju se algoritam formiranja stabla odluke „trudi“ da maksimizuje datu meru. U teoriji stabala odluka proučavane su mnoge mere kvaliteta. Generalno, sve mere kvaliteta testova mogu se podeliti u dve grupe: mere bazirane na teoriji informacija i mere bazirane na razdaljini.

Mere bazirane na teoriji informacija: Kod ovih mera zajedničko je da se one u osnovi baziraju na korišćenju pojma entropije, [3]. Tako postoje algoritmi za formiranje stabala odluka koji maksimizuju globalnu zajedničku informaciju, [4], [5]. Drugi algoritmi vrše lokalnu optimizaciju takozvanog informacionog dobitka, redukcije u entropiji usled podele u svakom od čvorova stabla, [1], [6], [7].

Mere bazirane na razdaljini: Pod pojmom razdaljine ovde se misli na razdaljinu između raspodela verovatnoća pojedinih klasa. Funkcije zasnovane na razdaljini mere separabilnost, odnosno divergenciju među klasama. Popularna funkcija je *Gini* indeks, [8-9]. Takođe je često korišćena *twoing rule* funkcija, [8]. *Bhattacharya* razdaljina, [10], *Kolmogorov-Smirnof* razdaljina, [11-12] i χ^2 statistika, [13-15], predstavljaju neke od drugih funkcija baziranih na razdaljini koje su bile korišćene prilikom formiranja stabala odluka.

U nastavku će najpoznatije mere kvaliteta testova biti detaljno razmotrene.

Information Gain – Ovu meru je u kontekstu stabala odluka popularizovao Quinlan, [1]. Ova mera predstavlja očekivanu redukciju entropije skupa instanci *T*, nakon testiranja atributa *A*.

$$Gain(T, A) = Entropy(T) - \sum_{i \in Vrednosti(A)} \frac{|T_i|}{|T|} \cdot Entropy(T_i) \tag{2.5}$$

U prethodnoj formuli, *T* označava skup svih instanci u tekućem čvoru, T_i podskup instanci skupa *T* koji čine instance kod kojih atribut *A* ima vrednost *i*, $T_i = \{s \in T | A(s) = i\}$. *Vrednosti(A)* jeste skup svih mogućih vrednosti atributa *A*. Entropijska funkcija *Entropy*, definisana je na sledeći način

$$Entropy(S) = \sum_{i=1}^{k} - p_i \log_2 p_i \tag{2.6}$$

U gornjem izrazu, p_i predstavlja procenat instanci iz skupa *S* koje pripadaju klasi *i* od ukupno *k* klasa.

U svakoj od narednih formula, skup instanci *T* u tekućem čvoru koji je potrebno podeliti sadrži *n>0* instanci koje pripadaju jednoj od *k* klasa. Inicijalno ovaj skup sadrži ceo trening skup. Hiperpovrš *H* vrši podelu skupa *T* u dva disjunktna podskupa T_{iznad} i T_{ispod}, koji sadrže instance koje se nalaze iznad, odnosno ispod hiperpovrši. U_j i B_j predstavljaju broj instanci koje pripadaju klasi *j* u skupu T_{iznad} i T_{ispod} respektivno. Sve mere inicijalno proveravaju da li su T_{iznad} i T_{ispod} homogeni, odnosno da li sve instance pripadaju istoj klasi, i ako je to slučaj vraćaju minimalnu vrednost, odnosno nulu.

Gini Index – Gini index predložio je Breiman [3]. Gini index meri verovatnoću pogrešne klasifikacije skupa instanci, umesto nečistoće podele. U literaturi postoje razne varijante ove mere, a vrlo često se koristi sledeća varijanta

$$\begin{aligned} Gini_{iznad} &= 1 - \sum_{i=1}^{k} \left(\frac{U_i}{|T_{iznad}|} \right)^2 \\ Gini_{ispod} &= 1 - \sum_{i=1}^{k} \left(\frac{B_i}{|T_{ispod}|} \right)^2 \\ Gini &= \frac{|T_{iznad}| \cdot Gini_{iznad} + |T_{ispod}| \cdot Gini_{ispod}}{n} \end{aligned} \tag{2.7}$$

U gornjim izrazima, $Gini_{iznad}$ predstavlja Gini index instanci koje se nalaze iznad hiperpovrši, a $Gini_{ispod}$ predstavlja Gini index instanci koje se nalaze ispod hiperpovrši *H*.

Twoing Rule – Ovu meru je takođe predložio Breiman, [3]. Definicija ove mere je:

$$TwoingValue = \left(\frac{|T_{iznad}|}{n} \right) \cdot \left(\frac{|T_{ispod}|}{n} \right) \cdot \left(\sum_{i=1}^{k} \left| \frac{U_i}{|T_{iznad}|} - \frac{B_i}{|T_{ispod}|} \right| \right)^2 \tag{2.8}$$

U gornjem izrazu $|T_{iznad}|, |T_{ispod}|$ označavaju broj instanci koje se nalaze iznad, odnosno ispod hiperravni *H*.

Max Minority – Originalno definisan u [4]. Prednost u korišćenju ove mere jeste da je teorijski pokazano da će formirano stablo imati dubinu od maksimalno *log n*. Međutim, razni eksperimenti [5] upućuju na to da i ostale mere retko kada rezultuju u stablima čija je dubina značajno veća od gore navedene.

$$\begin{aligned} Minority_{iznad} &= \sum_{i=1, i \neq \max U_i}^{k} U_i \\ Minority_{ispod} &= \sum_{i=1, i \neq \max B_i}^{k} B_i \\ MaxMinority &= \max\left(Minority_{iznad}, Minority_{ispod} \right) \end{aligned} \tag{2.9}$$

Sum Minority – Ova mera je slična *Max Minority* meri. Ako su $Minority_{iznad}$ i $Minority_{ispod}$ definisani kao kod *Max Minority* mere, tada je *Sum Minority* prosto zbir ove dve vrednosti. Ova mera predstavlja najjednostavniji način merenja kvaliteta podele, jer prosto predstavlja broj pogrešno klasifikovanih instanci u tekućem čvoru.

Sum of Variances – Definicija ove mere je:

$$Variance_{iznad} = \sum_{i=1}^{|T_{iznad}|}\left(Cat\left(T_{iznad_i}\right) - \sum_{j=1}^{|T_{iznad}|}\frac{Cat\left(T_{iznad_j}\right)}{|T_{iznad}|}\right)^2$$

$$Variance_{ispod} = \sum_{i=1}^{|T_{ispod}|}\left(Cat\left(T_{ispod_i}\right) - \sum_{j=1}^{|T_{ispod}|}\frac{Cat\left(T_{ispod_j}\right)}{|T_{ispod}|}\right)^2 \quad (2.10)$$

$$Sum\ of\ Variances = Variance_{iznad} + Variance_{ispod}$$

U gornjim izrazima $Cat(T_i)$ označava klasu kojoj pripada instanca i.

U dostupnoj literaturi postoji mnoštvo različitih algoritama za formiranje stabala odluka. Svi ovi algoritmi mogu se grubo podeliti, prema načinu kako se formiraju testovi u čvorovima na:

- algoritme koji formiraju stabla sa testovima samo jednog atributa u svakom čvoru i
- algoritme koji formiraju stabla sa testovima više atributa u svakom od čvorova.

Takođe su razvijeni mnogi hibridni algoritmi u smislu da se stabla odluka kombinuju sa drugim paradigmama iz mašinskog učenja i matematike, kakvi su, na primer, algoritmi koji kombinuju stabla odluka sa neuronskim mrežama, fazi logikom, Fišerovim linearnim diskriminatorima, linearnim programiranjem, evolutivnim algoritmima, itd. U nastavku će biti dat kratak pregled najznačajnijih radova iz svake od gore pomenutih oblasti.

Algoritmi koji u svakom od čvorova stabla testiraju samo po jedan atribut predstavljaju najstarije algoritme za formiranje stabala odluka. Među najpoznatije algoritme ovog tipa spadaju već spomenuti ID3 [1] i C4.5 [2] algoritmi Quinlan-a kao i Breimanov CART algoritam [8], predloženi početkom 80-tih godina prošlog veka.

Algoritmi koji u svakom od čvorova stabla testiraju više od jednog atributa su daleko manje proučavani i poznati. Henrichon i Fu [17] su među prvima predložili algoritam koji formira ortogonalna stabla ali po zahtevu korisnika može pridodati i testove koji predstavljaju linearnu kombinaciju atributa. Iyengar [18], je predložio iterativnu proceduru formiranja obeležja koja identifikuju potencijalno korisne položaje hiperravni pomoću kojih se vrši particija instanci. Breiman je predložio proširenje svog originalnog CART algoritma, CART-LC [8] koji koristi linearne kombinacije atributa.

You i Fu [19] su koristili linearne diskriminante u svakom od čvorova stabla. Loh i Vnichsetakul [20] su predložili algoritam koji koristi modifikovanu linearnu diskriminantnu funkciju i particiju u polarnom koordinatnom sistemu ako je sferična simetrija detektovana unutar trening skupa.

Obzirom da pronalaženje optimalnog položaja hiperravni predstavlja optimizacioni problem, istraživači su predlagali korišćenje raznih optimizacionih algoritama.

Heath je koristio algoritam simuliranog očvršćavanja za pronalaženje pozicije hiperravni, [16]. Brown [21], je koristio linearno programiranje u svakom od čvorova stabla i pokazao da ovaj pristup dovodi do formiranja manjih i tačnijih stabala u odnosu na CART-LC algoritam.

Murthy [22], je predložio OC1 algoritam koji koristi postupak randomizacije u vidu višestrukih slučajnih restartovanja i slučajnog perturbovanja koeficijenata hiperravni ako se prilikom determinističkog algoritma naiđe na lokani minimum.

Evolutivni algoritmi su takođe bili korišćeni u procesu formiranja stabala odluka iako su uglavnom razmatrana ortogonalna stabla. Koza [23] je koristio modifikovanu verziju genetskog programiranja [24], za formiranje stabala odluka. Folino [25] je takođe koristio genetsko programiranje za formiranje ortogonalnih stabala, ali je on koristio model koji omogućava laku paralelizaciju. Bot i Langdon [26] koristili su genetsko programiranje za formiranje neortogonalnih stabala odluke. Cantú-Paz [27] je predložio niz varijacija OC1 algoritma u kome se kao optimizacioni algoritam koristi simulirano očvršćavanje, evolutivne strategije ili genetski algoritam.

Korišćenje perceptrona [28], lociranih u čvorovima neposredno iznad listova stabla [29] ili u svakom od čvorova stabla [30-31] je takođe predloženo.

Kombinacija stabala odluka i neuronskih mreža takođe je proučavana. Korišćenje stabala odluka kod kojih su za testove u čvorovima korišćene male višeslojne neuronske mreže predloženo je u [32]. Takođe pokušano je i sa obrnutim smerom, Sethi [33] je predložio algoritam za formiranje neuronskih mreža na osnovu ortogonalnog stabla odluke.

Na kraju, postoji i mogućnost korišćenja nelinearnih testova u čvorovima stabla. Ovaj pristup je slabo proučavan i ako se izuzmu hibridni algoritmi koji kombinuju neuronske mreže i stabla odluka jedino je Ittner [34] predložio algoritam za formiranje stabala sa nelineranim testovima na taj način što se originalni skup atributa proširuje sa novim atributima formiranim od svih mogućih parova proizvoda originalnih atributa.

Ansambli prediktivnih modela bazirani su na ideji kombinovanja predikcija većeg broja individualno obučenih modela (tipično stabala odluke ili veštačkih neuronskih mreža) prilikom klasifikovanja novih instanci. Ova ideju koristimo u svakodnevnom životu prilikom donošenja važnih odluka koje mogu imate značajne finansijske, medicinske ili socijalne konsekvence. Konsultovanje većeg broja eksperata prilikom donošenja neke važne odluke, izvodi se u cilju učvršćivanja ubeđenja da donosimo ispravnu odluku. Stoga je bilo prirodno da se sličan koncept primeni i u oblasti mašinskog učenja.

Najraniji rad iz oblasti ansambala prediktivnih modela jeste [35], u kome se predlaže particija prostora atributa korišćenjem dva ili više klasifikatora. U radu [36] autori su pokazali da se generalizacija izolovane veštačke neuronske mreže može poboljšati korišćenjem ansambla veštačkih neuronskih mreža sa sličnom topologijom. Prekretnicu u razvoju teorije ansambala prediktivnih modela napravio je Shapire sa svojim radom [37] u kojem je dokazao da se jak klasifikator u PAC smislu (Probably Approximately Correct, PAC) može dobiti kombinovanjem slabih klasifikatora. Nakon ovog ključnog rada, istraživanja u oblasti ansambala prediktivnih modela su se intenzivirala, iako su se rezultati u literaturi pojavljivali pod različitim imenima. Dugačka lista uključuje: smeše eksperata [38-39], sjedinjavanje konsenzusa [40], kombinaciju višestrukih klasifikatora [41-45], fuziju klasifikatora [46-48], komitete neuronskih mreža [49], glasajući skup klasifikatora [50], ansamble klasifikatora [51], itd.

Svi ovi pristupi se međusobno razlikuju u načinu na koji formiraju individualne prediktivne modele i kombinuju njihove predikcije. Generalno, postoje dva načina na koji se mogu kombinovati prediktivni modeli: selekcija klasifikatora i fuzija klasifikatora. U slučaju selekcije klasifikatora, svaki klasifikator je obučen tako da postane ekspert u jednom, lokalnom regionu čitavog prostora atributa. Kombinovanje klasifikatora je zatim bazirano na trenutnom ulaznom vektoru: klasifikator obučavan sa podacima koji su najbliži trenutnom ulaznom vektoru, mereno pomoću odgovarajuće metrike, dobija najveći značaj. Jedan ili više lokalnih eksperata mogu biti nominovani da donesu odluku [38], [45], [52-54]. U slučaju fuzije klasifikatora, svaki od članova ansambla obučava se sa primerima iz čitavog prostora atributa. U ovom slučaju, kombinovanje klasifikatora uključuje sjedinjavanje individualnih (slabih) prediktivnih modela u cilju formiranja jednog (jakog) eksperta sa superiornim performansama. U ovu grupu spadaju poznati Bagging [55], Boosting [37], [56] algoritmi, kao i njihove brojne varijacije.

Kombinacija prediktivnih modela može se odnositi na diskretne, ili pak na kontinualne vrednosti izlaza individualnih prediktivnih modela [48], [57-58]. U drugom slučaju, predikcije pojedinih klasifikatora su najčešće normalizovane na interval [0, 1], i interpretiraju se kao podrška klasifikatora svakoj od mogućih klasa, ili čak kao uslovne verovatnoće [48], [59]. Ovakva interpretacija omogućava formiranje ansambla korišćenjem algebarskih pravila kombinovanja (većinsko glasanje, maksimum/minimum/suma/proizvod ili neka druga kombinacija uslovnih verovatnoća), [32], [58], [60-61], različite kombinacije bazirane na fuzzy logici, [47], [62-63], Dempster-Shafer fuziju klasifikatora, [43], [64], itd.

Prediktivni model sa savršenom generalizacijom nije moguće realizovati, usled šuma koji je prisutan u podacima, neadekvatno odabranih atributa, preklapajućih raspodela podataka iz različitih klasa, itd. U najboljem slučaju moguće je napraviti model koji će ispravno klasifikovati podatke u najvećem broju slučajeva. Osnovna ideja prediktivnih modela baziranih na korišćenju ansambala sastoji se u formiranju velikog broja individualnih klasifikatora, i kombinovanju njihovih predikcija na način koji će obezbediti da tačnost ansambla bude veća od tačnost individualnog modela. Da bi to bilo moguće, neophodno je da individualni klasifikatori prave greške na različitim instancama problema. U slučaju da svaki klasifikator pravi drugačije greške, njihovim pravilnim kombinovanjem biće moguće umanjiti ukupnu grešku klasifikacije. Stoga je osnovni uslov koji je potrebno zadovoljiti prilikom formiranja ansambla prediktivnih modela, da svaki od članova ansambla bude što je moguće više unikatan, pogotovo kada se posmatra njegovo ponašanje u slučaju pogrešno klasifikovanih instanci. Drugačije rečeno, potrebni su nam klasifikatori čije se razdvajajuće površi u značajnoj meri razlikuju od razdvajajućih površi ostalih klasifikatora iz ansambla. Za takav skup klasifikatora kažemo da je raznolik.

Raznolikost klasifikatora se može postići na nekoliko načina. Najpopularniji metod sastoji se u korišćenju različitih trening skupova za formiranje individualnih klasifikatora. Ovakvi trening skupovi se često dobijaju kao rezultat korišćenja tehnika odabiranja, kao što su Bootstraping ili Bagging, kod kojih se trening skupovi formiraju slučajnim odabirom instanci iz celokupnog trening skupa. Drugi pristup za ostvarivanje raznolikosti koristi drugačije parametre prilikom obučavanja individualnih klasifikatora. Na primer, čitav niz veštačkih neuronskih mreža može biti formiran obučavanjem koje koristi različite inicijalne vrednosti težina veza, različit broj slojeva odnosno neurona u mreži, različite ciljne funkcije, itd. Menjanjem ovih parametara može se uticati na raznolikost formiranih modela. Mogućnost da se relativno lako može kontrolisati struktura i performanse formiranog modela u slučaju veštačkih neuronskih mreža i stabala odluke, čini ih pogodnim kandidatima za korišćenje u ansamblima. Raznolikost se može postići i korišćenjem različitih tipova klasifikatora unutar jednog ansambla, na primer neuronskih mreža, stabala odluke, support vector mašina, itd. Na kraju, raznolikost se može postići i korišćenjem različitih podskupova atributa prilikom obučavanja individualnih klasifikatora [65].

Da bi se raznolikost mogla numerički izraziti, u praksi se koristi veliki broj različitih mera raznolikosti, od kojih su najpoznatije: mera bazirana na korelaciji, Q-statistika, entropijska mera, Kohavi-Wolpert varijansa, mera težine, itd.

Svaki ansambl prediktivnih modela sastoji se od dve ključne komponente. Prvo, potrebno je definisati postupak za formiranje ansambla koji će biti što je moguće više raznolik. Nakon što je ansambl formiran, potrebno je definisati način na koji će se kombinovati predikcije individualnih prediktivnih modela koji čine ansambl na takav način da se tačne odluke pojačavaju, a netačne smanjuju.

U dostupnoj literaturi postoji veliki broj predloženih postupaka za formiranje ansambla, od kojih su najpoznatiji: Bagging [55], Boosting [37], AdaBoost [56], Stacked generalization [66] i Mixture of experts [67]. Prilikom formiranja ansambla mora se odgovoriti na dva pitanja. Kako će se formirati individualni prediktivni modeli i kako će se oni međusobno razlikovati. Odgovori na ova pitanja direktno utiču na raznolikost ansambla, a time i na njegove performanse kao prediktivnog modela. Zbog toga bi svaki postupak za formiranje ansambla trebao da teži da poveća raznolikost ansambla. Međutim, najveći broj postojećih postupaka za formiranje ansambla ne pokušava da maksimizuje neku od mera raznolikosti, iako ima i suprotnih primera, [68-69]. Raznolikost ansambla se uglavnom postiže indirektno, korišćenjem odgovarajućih postupaka odabiranja trening podskupova ili odabirom drugačijih parametara prilikom obučavanja individualnih prediktivnih modela.

Bagging

Bagging algoritam [55] je jedan od prvih predloženih postupaka za formiranje ansambala prediktivnih modela. Predloženi algoritam je vrlo intuitivan i jednostavan za implementaciju, ali i pored toga odlikuje se izuzetno dobrim performansama. Raznolikost u *Bagging* algoritmu se postiže korišćenjem različitih trening podskupova za obučavanje individualnih prediktivnih modela. Svaki trening podskup je formiran slučajnim izvlačenjem podataka, sa vraćanjem, iz celokupnog trening skupa. Svaki od trening podskupova se zatim koristi za formiranje jednog prediktivnog modela istog tipa. Predikcije individualnih prediktivnih modela se zatim kombinuju korišćenjem većinskog odlučivanja. Za svaki test primer, bira se ona klasa koju je odabrao najveći broj prediktivnih modela iz ansambla.

E = Bagging (T_S, Slab_prediktor, T, P)

Ulazi:

T_S	*Skup trening instanci. Svaka od instanci je predstavljena pomoću niza vrednosti atributa,* $\boldsymbol{a}_i$*, praćenog klasom kojoj ta instanca pripada,* $c_i \in C = \{c_1, ..., c_K\}$.
Slab_prediktor	*Odabrani algoritam koji će biti korišćen za formiranje individualnih prediktivnih modela.*
T	*Broj iteracija Bagging algoritma, broj članova ansambla.*
P	*Procenat trening instanci iz skupa T_S koje čine svaki trening podskup.*

Izlazi:

E	*Ansambl prediktivnih modela*

- Za svako *t*, *t=1, ..., T*
  - Formiraj trening podskup T_S_t izvlačeći na slučajan način, sa vraćanjem, $\left\lceil \frac{P \cdot T\_S}{100} \right\rceil$ instanci iz skupa *T_S*.
  - Koristeći algoritam *Slab_prediktor* i trening podskup T_S_t formiraj prediktivni model, h_t.
  - Dodaj h_t u ansambl *E*.
- Vrati *E*

Testiranje: Jednostavno većinsko odlučivanje – Za datu ulaznu instancu *x*

- Izračunaj predikcije svakog člana ansambla $E=\{h_1, ..., h_T\}$, korišćenjem instance *x*.
- Neka je $v_{t,j} = \begin{cases} 1, & \text{ako je predikcija } h_t \text{ klasa } c_j \\ 0, & \text{inače} \end{cases}$, predikcija klasifikatora h_t.
- Izračunaj ukupan broj glasova koji je dobila svaka klasa, $V_j = \sum_{t=1}^{T} v_{t,j}$, $j = 1, ..., K$.
- Kao predikciju celog ansambla, odaberi klasu koja je dobila najveći broj glasova.

Algoritam 2.2 Bagging *algoritam za formiranje ansambla*

Posebna pogodnost *Bagging* algoritma je da se vrlo lako može paralelizovati. *Bagging* algoritam je takođe pogodan u slučaju da je ukupan broj trening instanci koje nam stoje na raspolaganju vrlo mali. Da bi se obezbedilo da svaki trening podskup sadrži dovoljan broj instanci, koristi se relativno veliki procenat trening skupa (75 % do 100 %). Ovo sa druge strane dovodi do toga da se formirani trening podskupovi u velikoj meri preklapaju, što može smanjiti raznolikost ansambla. Da bi se ipak obezbedila dovoljna raznolikost, potrebno je koristiti one prediktivne modele koji su relativno *nestabilni* u pogledu načina na koji formiraju razdvajajuće površi tokom obučavanja. Kao što je ranije rečeno, veštačke neuronske mreže i stabla odluke su dobri kandidati, jer se promenom njihovih parametara obučavanja može uticati na konačni izgled razdvajajuće površi.

Pored osnovnog *Bagging* algoritma, u dostupnoj literaturi postoji veliki broj njegovih varijacija, od kojih su najpoznatije *Random forests* [70] i *Pasting small votes* [71].

Boosting

1990 godine Schapire je dokazao da se slab prediktivni model (*weak learner*), odnosno prediktivni model koji je samo malo bolji od prostog slučajnog pogađanja, može prevesti u jak prediktivni model (*strong learner*), koji tačno klasifikuje sve osim proizvoljno malog broja test instanci [37]. Formalne difinicije slabog i jakog prediktivnog modela mogu se naći u [37], gde je takođe prikazan i algoritam za *pojačavanje* (*boosting*) slabog klasifikatora do nivoa jakog klasifikatora. Predloženi *Boosting* algoritam danas se smatra jednim od najvećih doprinosa u skorijoj istoriji mašinskog učenja.

Slično *Bagging* algoritmu, i *Boosting* algoritam formira ansambl prediktivnih modela koristeći odabiranje trening instanci. Međutim, kod *Boosting* algoritma odabiranje se ne vrši na slučajan način, već je strateški usmereno na odabir onih instanci koje će omogućiti najkvalitetnije formiranje narednih prediktivnih modela. U osnovi, *Boosting* algoritam formira tri slaba prediktivna modela: prvi model C_1 formira se korišćenjem slučajno izabranog podskupa trening instanci. Trening podskup za drugi prediktivni model, C_2, bira se na način da sadrži samo 50 % trening instanci koje C_1 klasifikuje ispravno, i 50 % trening instanci koje C_1 klasifikuje pogrešno. Treći prediktivni model, C_3, formira se korišćenjem trening podskupa koji sadrži samo instance na kojima se C_1 i C_2 ne slažu. Tri prediktivna modela se zatim kombinuju pomoću većinskog odlučivanja.

E= Boosting (T_S, Slab_prediktor)

Ulazi:

T_S	*Skup od N trening instanci. Svaka od instanci je predstavljena pomoću niza vrednosti atributa,* $\mathbf{a}_i$*, praćenog klasom kojoj ta instanca pripada,* $c_i \in C = \{c_1, c_2\}$.
Slab_prediktor	*Odabrani algoritam koji će biti korišćen za formiranje individualnih prediktivnih modela.*

Izlazi:

E	*Ansambl prediktivnih modela*

- Odaberi na slučajan način, bez vraćanja, $N_1<N$ trening instanci iz trening skupa *T_S*. Odabrani trening podskup označi sa S_1.
- Koristeći algoritam *Slab_prediktor* i trening skup S_1 formiraj prvi prediktivni model, C_1.
- Formiraj trening podskup S_2 koji sadrži 50 % instanci koje tačno klasifikuje C_1 i 50 % instanci koje pogrešno klasifikuje C_1 na sledeći način:
  - Na slučajan način izaberi realan broj iz intervala [0, 1]. Ukoliko je broj veći od 0.5 biraj instance iz *T_S* i prezentuj ih modelu C_1 sve dok ne naiđeš na prvu koju model C_1 pogrešno klasifikuje. Dodaj tu instancu u S_2.
  - Ukoliko je broj manji od 0.5 biraj instance iz *T_S* i prezentuj ih modelu C_1 sve dok ne naiđeš na prvu koju model C_1 ispravno klasifikuje. Dodaj tu instancu u S_2.
  - Ponavljaj postupak sve dok je moguće dodati nove instance u S_2.
- Koristeći algoritam *Slab_prediktor* i trening skup S_2 formiraj drugi prediktivni model, C_2.
- Formiraj trening podskup S_3 birajući one instance na kojima se C_1 i C_2 ne slažu. Koristeći algoritam *Slab_prediktor* i trening skup S_3 formiraj treći prediktivni model, C_3.

Testiranje: Za datu ulaznu instancu *x*

- Klasifikuj *x* pomoću C_1 i C_2. Ukoliko su njihove predikcije iste, to je ujedno i predikcija za instancu *x*.
- Ako se predikcije C_1 i C_2 razlikuju, klasifikuj *x* u onu klasu koju predviđa C_3.

Algoritam 2.3 Boosting *algoritam za formiranje ansambla*

Može se pokazati da je klasifikaciona greška ovako formiranog ansambla ograničena odgore, i da je manja od greške najboljeg klasifikatora u ansamblu, pod uslovom da svaki od tri klasifikatora ima grešku klasifikacije koja je manja od 0.5. Za problem klasifikacije u dve klase, greška od 0.5 je najmanje što možemo očekivati od klasifikatora, jer ona predstavlja grešku koja se pravi ukoliko se klasifikacija izvodi na slučajan način. Ovim je pokazano da se kombinovanjem slabih klasifikatora može formirati klasifikator koji će biti bolji od svakog od njih (jak klasifikator). Iterativnim ponavljanjem ovog algoritma moguće je napraviti proizvoljno tačan klasifikator.

AdaBoost

1997, Freund i Schapire su predložili *AdaBoost* algoritam [56], koji je od tada doživeo ogromnu popularnost. *AdaBoost* predstavlja varijantu originalnog *Boosting* algoritma. Među mnogobrojnim varijantama, najčešće se koriste *AdaBoost.M1* i *AdaBoost.R* verzije koje su u stanju da rade sa problemima klasifikacije u više od dve klase, odnosno sa regresivnim problemima. U nastavku će biti opisana samo *AdaBoost.M1* varijanta, jer je ona relevantna za ovaj rad.

AdaBoost algoritam formira ansambl prediktivnih modela, koje zatim kombinuje korišćenjem težinskog većinskog odlučivanja. Prediktivni modeli se formiraju korišćenjem nekog od algoritama za formiranje slabog klasifikatora, koristeći težinski modifikovan trening set. Svakoj od instanci iz trening skupa dodeljuje se težina koja odgovara verovatnoći da će biti odabrana u tekući trening podskup na osnovu kojega će biti formiran naredni prediktivni model. Ova distribucija težina se ažurira nakon što se formira tekući prediktivni model, na takav način da se težine instanci koje je tekući prediktivni model pogrešno klasifikovao povećavaju. Instance koje su bile pogrešno klasifikovane od strane prethodno formiranog prediktivnog modela će stoga imati veću verovatnoću da budu uključene u trening podskup koji će biti korišćen za formiranje narednog prediktivnog modela. Na ovaj način će se u narednim trening podskupovima nalaziti sve veći broj instanci koje je teško klasifikovati. *Adaboost.M1* algoritam prikazan je na sledećoj slici.

E= AdaBoost.M1(T_S, Slab_prediktor, T)

Ulazi:	
T_S	*Skup od N trening instanci,* $T\_S = \left[\left(a_i, y_i\right)\right], i = 1,...,N$ *. Svaka od instanci je predstavljena pomoću niza vrednosti atributa,* $\mathbf{a_i}$*, praćenog klasom kojoj ta instanca pripada,* $y_i \in Y = \left\{y_1,...,y_K\right\}$ *.*
Slab_prediktor	*Odabrani algoritam koji će biti korišćen za formiranje individualnih prediktivnih modela.*
T	*Ceo broj koji označava broj iteracija algoritma*
Izlazi:	
E	*Ansambl prediktivnih modela*

- Inicijalizuj težine trening instanci, $D_1(i) = \frac{1}{N}, i = 1,...,N$.
- Za svako *t*, *t=1, ..., T*
  - Formiraj trening podskup S_t, na slučajan način izvlačeći instance iz *T_S* koristeći težine $D_t(i)$ kao verovatnoće izvlačenja svake instance.
  - Koristeći algoritam *Slab_prediktor* i trening skup S_t formiraj prediktivni model, C_t.
  - Dodaj prediktivni model C_t u ansambl *E*.
  - Izračunaj klasifikacionu grešku prediktivnog modela kao

$$C_t : \varepsilon_t = \sum_{i:C_t(a_i)\neq y_i} D_t(i) . \qquad (2.11)$$

    Ako je $\varepsilon_t > \frac{1}{2}$, prekini dalji rad.

  - $\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$
  - Ažuriraj težine instanci iz celog trening skupa *T_S* na sledeći način:

$$D_t : D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t & ako\ je\ C_t(a_i) = y_i \\ 1 & inače \end{cases} , \qquad (2.12)$$

    gde je $Z_t = \sum_i D_t(i)$ normalizaciona konstanta izabrana na način da D_{t+1} može da se tretira kao raspodela verovatnoća.

Testiranje – težinsko većinsko odlučivanje: Za datu ulaznu instancu *x*

- Izračunaj ukupan broj glasova dodeljen svakoj od mogućih klasa od strane ansambla

$$V_j = \sum_{t:C_t(x)=y_j} \log\frac{1}{\beta_t}, j = 1,...,K .$$

- Odaberi klasu koja je dobila najveći broj glasova kao konačnu klasifikaciju od strane ansambla.

Algoritam 2.4 AdaBoost.M1 *algoritam za formiranje ansambla*

AdaBoost algoritam koristi distribuciju težina $D_t(i)$ pomoću koje određuje koje će trening instance biti uključene u formiranje narednog klasifikatora iz ansambla. Na početku je ova distribucija uniformna, tako da sve instance imaju jednaku šansu da budu uključene u prvi trening podskup. Greška na trening skupu ε_t prediktivnog modela h_t takođe uzima u obzir ovu distribuciju (2.11). Kao i kod osnovne verzije *Boosting* algoritma, neophodan uslov je da je ε_t manje od ½. Sledeći korak u algoritmu je računanje normalizovane greške, β_t. Pravilo za ažuriranje distribucije težina opisano je pomoću jednačine (2.12). Težine onih instanci koje su ispravno klasifikovane od strane tekućeg klasifikatora su umanjene za faktor β_t, dok su težine pogrešno klasifikovanih instanci nepromenjene. Kada se izvrši normalizacija ažuriranih težina, težine pogrešno klasifikovanih instanci će zapravo biti povećane. Stoga će se, iteraciju po iteraciju, *AdaBoost* algoritam sve više fokusirati na teške instance iz trening skupa.

Nakon što se formira ansambl od *T* klasifikatora, može biti upotrebljen za klasifikovanje test instance. Formirani ansambl koristi težinsko većinsko odlučivanje prilikom donošenja kolektivne odluke. Težine pridružene svakom od članova ansambla su odabrane tako da se klasifikatorima koji su pokazali dobre karakteristike prilikom treninga dodeljuje veća težina, računata kao recipročna vrednost normalizovane greške β_t. Pošto vrednost β_t može biti bliska nuli, njena recipročna vrednost može biti vrlo velika. Da bi se izbegle numeričke greške kao posledica rada sa velikim brojevima, najčešće se kao težina pridružena klasifikatoru prilikom glasanja računa kao logaritam recipročne vrednosti β_t. Klasa koja dobije najveći broj glasova predstavlja konačnu odluku ansambla.

Druga ključna komponenta svakog ansambla prediktivnih modela jeste način na koji se predikcije individualnih prediktivnih modela kombinuju u jednu, kolektivnu odluku. Algoritmi za kombinovanje prediktivnih modela mogu se podeliti na algoritme koji zahtevaju obučavanje i algoritme koji ne zahtevaju obučavanje. Algoritme za kombinovanje prediktivnih modela takođe možemo podeliti i na algoritme koji se koriste za kombinovanje prediktivnih modela sa diskretnim izlazom i na algoritme koji se koriste za kombinovanje prediktivnih modela sa kontinualnim izlazom.

U slučaju algoritama za kombinovanje prediktivnih modela koji zahtevaju dodatno obučavanje, odgovarajući parametri sistema koji se koristi za kombinovanje, tipično su to neke vrste težina, se optimizuju pomoću posebnog postupka obučavanja. Ovako određeni parametri obično zavise od skupa instanci koji je korišćen za njihovo određivanje.

U slučaju druge podele, algoritmi za kombinovanje prediktivnih modela sa diskretnim izlazom kombinuju klase prognozirane od strane individualnih prediktivnih modela, $c_i \in C = \{c_1, c_2\}$. Sa druge strane, algoritmi za kombinovanje prediktivnih modela sa kontinualnim izlazom koriste kontinualne vrednosti generisane od strane individualnih prediktivnih modela. Ove kontinualne vrednosti često predstavljaju stepen podrške koju svakoj od mogućih klasa daje individualni prediktivni model, ili čak mogu biti interpretirane kao aposteriorne uslovne verovatnoće, $P(c_i \mid x)$. Ovo je moguće ukoliko su izlazi prediktivnih modela normalizovani tako da u ukupnoj zbiru daju broj jedan i ukoliko su individualni prediktivni modeli obučavani sa dovoljno reprezentativnim trening skupom. Mnogi od prediktivnih modela, kao što su veštačke neuronske mreže, generišu kontinualne izlaze koji se mogu interpretirati kao aposteriorne uslovne verovatnoće.

Obzirom da stabla odluka generišu diskretne izlaze, prilikom njihovog kombinovanja mogu se koristiti samo algoritmi iz prve grupe. Zbog toga će u nastavku biti predstavljeni samo najpoznatiji algoritmi za kombinovanje prediktivnih modela sa diskretnim izlazom.

Pretpostavimo da su nam od svakog individualnog prediktivnog modela iz ansambla dostupne samo predikcije klase kojoj tekuća instanca pripada, pri čemu je broj različitih klasa je konačan. Označimo predikciju *i*-tog prediktivnog modela sa vektorom $d_i \in \{0,1\}^C$, *i=1, ..., T*, gde je *T* broj prediktivnog modela u ansamblu, a *C* je broj različitih klasa. Ako *i*-ti prediktivni model klasifikuje tekuću instancu u klasu c_k tada je $d_i = \begin{cases} 1, & j = k \\ 0, & j \neq k \end{cases}, j = 1,...,C$.

Većinsko odlučivanje

Postoje tri varijante većinskog odlučivanja::

- *jednoglasno odlučivanje* – bira se klasa za koju glasaju svi članovi ansambla,
- *odlučivanje prostom većinom* - bira se klasa za koju glasa 50% +1 član ansambla,
- *većinsko odlučivanje* - bira se klasa za koju glasa najveći broj članova ansambla, bez obzira da li je taj broj veći od 50%.

Algoritam na osnovu kojega ansambl donosi odluku korišćenjem većinskog odlučivanja može se opisati na sledeći način:

Odaberi klasu c_K ako važi

$$\sum_{t=1}^{T} d_{t,K} = \max_{j=1}^{C} \sum_{t=1}^{T} d_{t,j} \tag{2.13}$$

Može se pokazati da je većinsko odlučivanje optimalno pravilo za kombinovanje izlaza prediktivnih modela pod uslovom da važe sledeće pretpostavke:

- Ansambl ima neparan broj članova u slučaju problema klasifikacije u dve različite klase,
- verovatnoća da će svaki prediktivni model tačno klasifikovati instancu x iznosi p za svaku instancu x,
- izlazi prediktivnih modela su međusobno nezavisni.

Pod ovim pretpostavkama, odlučivanje prostom većinom i većinsko odlučivanje su identični, a ansambl donosi tačnu odluku ukoliko barem $\lfloor T/2 \rfloor + 1$ prediktivnih modela izabere tačnu klasu. Verovatnoća da ansambl donese ispravnu odluku može se predstaviti pomoću binomne raspodele verovatnoće

$$P_{ans} = \sum_{k=(T/2)+1}^{T} \binom{T}{k} p^k (1-p)^{T-k} \tag{2.14}$$

P_{ans} teži ka 1 kada $T \to \infty$, ako je $p > 0.5$, a teži ka 0 ako je $p < 0.5$. Ovaj rezultat je poznat iz teorije verovatnoće i statistike [72-73]. Upravu ovde leži snaga većinskog odlučivanja: ako za svakog člana ansambla možemo garantovati da je njegova verovatnoća davanja ispravnog odgovora veća od 0.5, tada za dovoljno veliki ansambl (recimo od 100 ili više članova), verovatnoća donošenja ispravne odluke čitavog ansambla teži ka 1. Ova analiza odnosi se na slučaj klasifikacionih problema sa samo dve moguće klase. Može se pokazati da u slučaju klasifikacionih problema sa više od dve moguće klase, verovatnoća ispravne klasifikacije individualnih klasifikatora ne mora biti veća od 0.5, već zapravo može biti i znatno manja, a da tačnost ansambla i dalje teži ka 1. Za odličnu analizu većinskog odlučivanja čitalac može pogledati [74].

Težinsko većinsko odlučivanje

Ukoliko postoje dokazi da su neki od članova ansambla pouzdaniji u donošenju ispravnih odluka od drugih, davanje veće težine njihovim prognozama moglo bi dodatno popraviti performanse čitavog ansambla. Težinsko većinsko odlučivanje dodeljuje težinu w_j prediktivnom modelu h_j koja je proporcionalna poverenju koje imamo njega. Algoritam težinskog većinskog odlučivanja bira klasu c_K ako važi

$$\sum_{t=1}^{T} w_t \cdot d_{t,K} = \max_{j=1}^{C} \sum_{t=1}^{T} w_t \cdot d_{t,j} \tag{2.15}$$

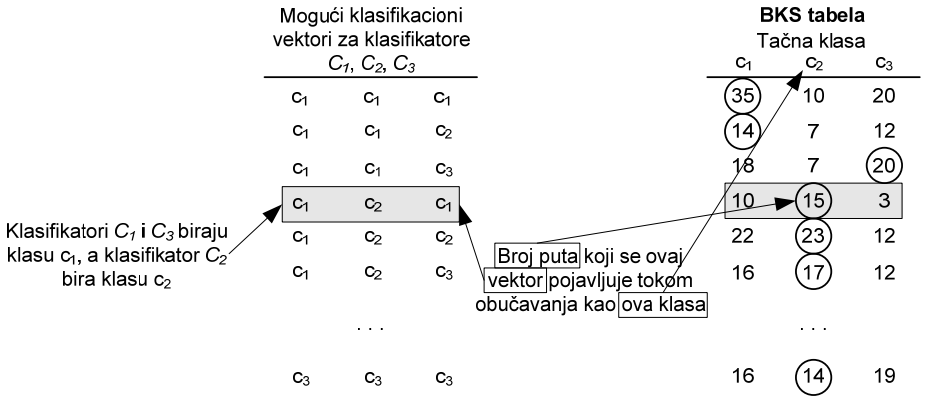
Odnosno, bira klasu c_K ako je ukupna težinska suma dodeljena klasi c_K veća od ukupne težinske sume dodeljene bilo kojoj drugoj klasi.

Osnovno pitanje koje se postavlja kod težinskog većinskog odlučivanja jeste na koji način pridružiti težine pojedinim prediktivnim modelima. Ukoliko bi smo posedovali informaciju o tome koji prediktivni modeli imaju veću tačnost klasifikacije, njima bi smo pridružili veću težinu, ili bi smo koristili samo njih. Međutim, mi ovu informaciju nemamo. Jedan od mogućih pristupa jeste da se kao merilo tačnosti svakog prediktivnog modela iz ansambla koristi njegova tačnost estimirana na posebno, validacionom skupu instanci, ili njegova tačnost na trening skupu. Ovaj drugi pristup koristi se u *AdaBoost* algoritmu. Detaljna diskusija o težinskom većinskom odlučivanju može se pronaći u [75].

Behavior Knowledge Space (BKS)

BKS koristi tabelu, formiranu na osnovu klasifikacija trening skupa, u kojoj se vodi evidencija koliko često se pojavljuje svaki od klasifikacionih vektora, [76-77]. Pod klasifikacionim vektorom podrazumeva se vektor formiran od predikcija svakog od članova ansambla, za datu ulaznu instancu. Zatim se svakom od klasifikacionih vektora pridružuje ona klasa za koju se posmatrani vektor najviše puta pojavljuje tokom obučavanja. Prilikom korišćenja, svaki put kada se odgovarajući klasifikacioni vektor pojavi na izlazima

ansambla, njemu pridružena klasa se bira kao klasa u koju ansambl klasifikuje tekuću instancu. Opisana procedura se najbolje razume pomoću primera, prikazanog na slici 2.3.



Slika 2.3 Ilustracija BKS *algoritma*

Pretpostavimo da se ansambl sastoji od tri prediktivna modela, C_1, C_2, C_3 i da se koristi za klasifikaciju instanci u jednu od tri klase, c_1, c_2, c_3. U ovom slučaju postoji ukupno 27 različitih klasifikacionih vektora koji se mogu pojaviti na izlazima tri klasifikatora. Tokom obučavanja, potrebno je voditi evidenciju koliko često se svaki klasifikacioni vektor pojavljuje zajedno sa odgovarajućom tačnom klasom za svako od pojavljivanja. Na slici 2.3 prikazana je jedna hipotetička situacija. Na primer, pretpostavimo da se klasifikacioni vektor $\{c_1, c_2, c_1\}$ pojavljuje ukupno 28 puta, od toga u 10 slučaja kada se ovaj vektor pojavio ispravna klasa je bila c_1, u 15 slučaja tačna klasa je bila c_2, dok je u 3 slučaja tačna klasa bila c_3. Obzirom da se ovaj vektor najčešće pojavljivao kada su u pitanju bile instance koje pripadaju klasi c_2, njemu će biti asocirana klasa c_2. Tokom testiranja, svaki put kada se na izlazima tri klasifikatora pojavi klasifikacioni vektor $\{c_1, c_2, c_1\}$ ansambl će tekuću instancu klasifikovati u klasu c_2. Potrebno je primetiti da je izbor klase c_2 drugačiji od izbora klase c_1 koja bi bila izabrana u slučaju većinskog odlučivanja, što jasno ukazuje da *BKS* algoritam koristi potpuno drugačiji pristup od svih prethodno razmatranih algoritama za kombinovanje klasifikatora.

Koliko je autorima poznato, u dostupnoj naučnoj literaturi ne postoji ni jedna referenca koja se bavi hardverskom implementacijom nekog algoritma za generisanje ansambaa stabala odluka.

3. IP jezgra za hardversko generisanje ansambala stabala odluka

3.1 *H_DTE* arhitektura za hardversko generisanje ansambala stabala odluka

U glavi 2 izloženi su najvažniji algoritmi za formiranje ansambala prediktivnih modela. Neki od njih su relativno jednostavni za hardversku implementaciju. *Bagging* algoritam je vrlo jednostavan za hardversku implementaciju baziranu na korišćenju hardverske arhiteture za formiranje individualnog stabla odluke iz ansambla, pri čemu su performanse formiranog ansambla u većini slučajeva vrlo dobre. Predložena *H_DTE* arhitektura za hardversko formiranje ansambla stabala odluka bazira se na hardverskoj implementaciji *Bagging* algoritma. On je već prikazan u glavi 2, ali je ovde prikazan još jednom radi lakšeg razumevanja načina rada predložene *H_DTE* arhitekture. Od sada pa na dalje ovaj algoritam ćemo označavati kao *DTE* algoritam za formiranje ansambla stabala odluke.

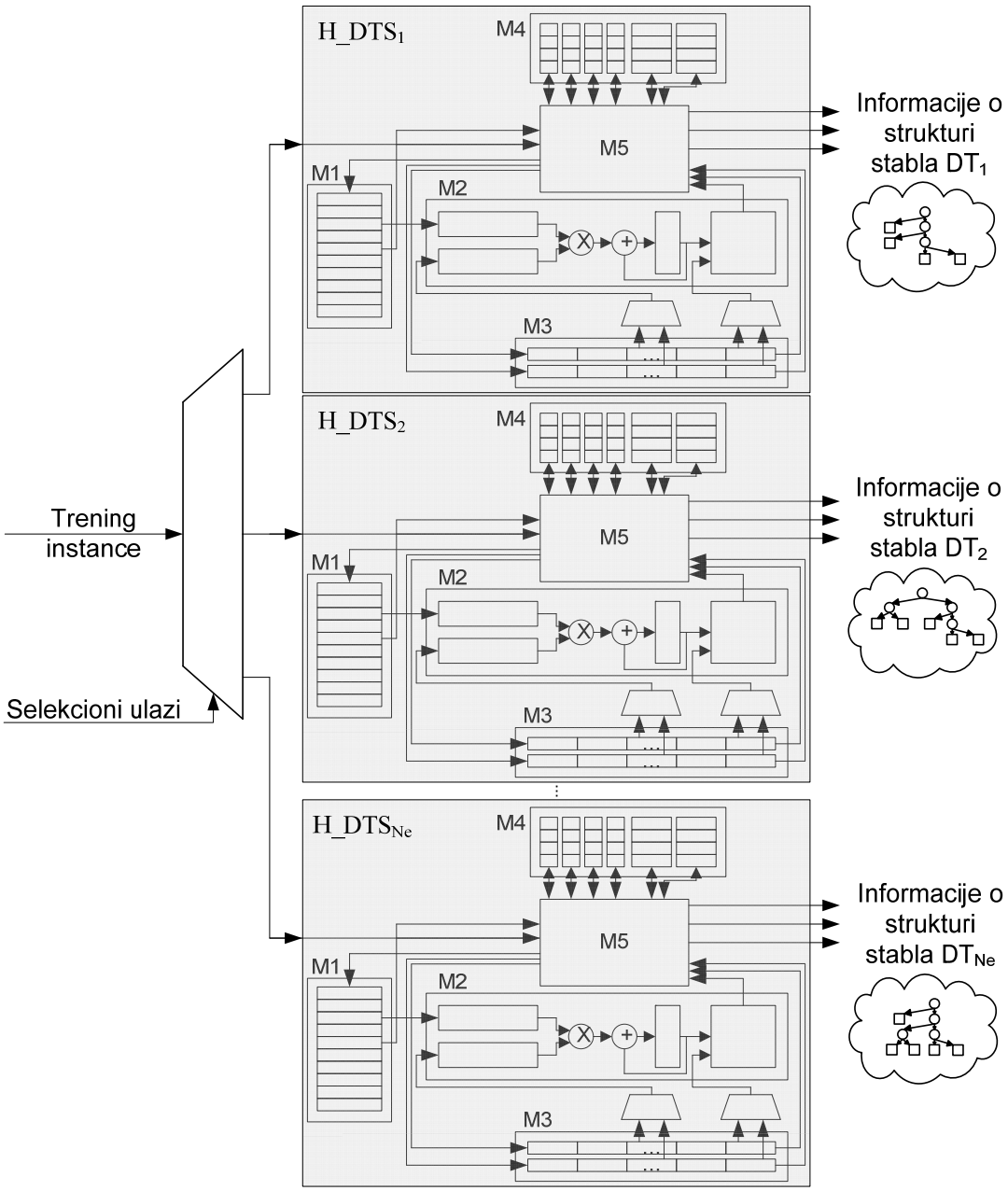
E = DTE (T_S, Slab_prediktor, N, P)

Ulazi:	
T_S	*Skup trening instanci. Svaka od instanci je predstavljena pomoću niza vrednosti atributa,* ***d****, praćenog klasom kojoj ta instanca pripada,* $c_i \in C = \{c_1, ..., c_K\}$.
Slab_prediktor	*Odabrani algoritam koji će biti korišćen za formiranje individualnih prediktivnih modela.*
N_e	*Broj iteracija Bagging algoritma, broj članova ansambla.*
P	*Procenat trening instanci iz skupa T_S koje čine svaki trening podskup.*
Izlazi:	
E	*Ansambl prediktivnih modela*

- Za svako *t*, *t=1, ..., N_e*
  - Formiraj trening podskup T_S_t izvlačeći na slučajan način, sa vraćanjem, $\left\lceil \frac{P \cdot T\_S}{100} \right\rceil$ instanci iz skupa *T_S*.
  - Koristeći algoritam *Slab_prediktor* i trening podskup T_S_t formiraj prediktivni model, h_t.
  - Dodaj h_t u ansambl *E*.
- Vrati *E*

Algoritam 3.1 DTE algoritam za formiranje ansambla prediktivnih modela

Kao i svi algoritmi za formiranje ansambala prediktivnih modela i *DTE* algoritam formira ansambl iterativnim postupkom. U svakoj iteraciji se koristeći podskup skupa trening instanci, formira jedan prediktivni model, u našem slučaju jedno stablo odluke. Za formiranje prediktivnog modela može se iskoristiti bilo koji od postojećih algoritama, tako da se mogu koristiti i ranije predloženi *DTL*, *DTN* i *DTS* algoritmi za formiranje neortogonalnih odnosno nelinearnih stabala odluke, [78]. Glavna prednost *DTE* algoritma u odnosu na većinu drugih algoritama za formiranje ansambala leži u činjenici da je formiranje individualnog prediktivnog modela u tekućoj iteraciji nezavisno od svih ostalih. Ova činjenica omogućava jednostavno paralelizovanje *DTE* algoritma, što rezultuje značajnim skraćenjem vremena potrebnog za formiranje ansambla, naravno na uštrb povećanja potrebnih hardverskih resursa. Ova ideja iskorišćena je za razvoj *H_DTE* arhitekture namenjene za hardversko formiranje ansambala stabala odluka koja je bazirana na paralelnoj verziji *DTE* algoritma. *H_DTE* arhitektura prikazana je na slici 3.1.



Slika 3.1 H_DTE *arhitektura za hardversko formiranje ansambala neortogonalnih ili nelinearnih stabala odluke*

H_DTE arhitektura se sastoji od ukupno N_e *H_DTS* modula, pri čemu je N_e broj članova ansambla koji se formira. *H_DTS* moduli su u stanju da formiraju neortogonalna ili nelinearna stabla odluke kao što je pokazano u [78]. Jedina razlika je u tome da je u svakom *H_DTS* modulu, memorija za smeštanje trening instanci (M1 modul) ovoga puta značajno manja i iznosi $\lceil P \cdot T\_S / 100 \rceil$ instanci.

Nakon što se u svaki od *H_DTS* modula učitaju odgovarajuće instance iz originalnog trening skupa pristupa sa formiranju ansambla. Svaki modul formira po jedno stablo odluke, nezavisno od drugih, koristeći raspoložive trening instance. Informacije o generisanim čvorovima stabla odluke prosleđuju se kroz odgovarajuće izlazne portove *H_DTS* modula, pri čemu ovaj proces nije sinhronizovan. Vremenski trenutci u kojima se na izlaznim portovima *H_DTS* modula pojavljuju informacije o formiranim čvorovima se razlikuju od modula do modula, jer svaki od njih formira neko drugo stablo odluke.

3.2 *H_DTS* arhitektura za hardversku generisanje individualnog stabla iz ansambla

H_DTS arhitektura predstavlja hardversku implementaciju *DTS* algoritma za evolutivno formiranje individualnog stabla odluke. Na ovom mestu biće opisani samo najvažniji detalji *H_DTS* arhitekture

neophodni za razumevanje rada *H_DTE* arhitekture kao i za razumevanje teorijskih i eksperimentalnih ocena potrebnih hardverskih resursa i performansi različitih *H_DTE* arhitektura. Zainteresovani čitalac koji se želi detaljnije upoznati sa *H_DTS* arhitekturom za hardversko generisanje individualnog stabla odluke se upućuje na [78]. Sam *DTS* algoritam za formiranje individulanog stabla prikazan je na sledećoj slici.

koren = DTS (*trening_skup*)

Ulazi:	
trening_skup	*Skup trening instanci. Svaka od instanci je predstavljena pomoću vektora realnih brojeva dužine **n**, praćenog brojem klase kojoj ta instanca pripada*
Izlazi:	
koren	*Koren formiranog neortogonalnog stabla odluke*

- Kreiraj čvor *koren*

- Ako sve instance iz skupa *trening_skup* pripadaju istoj klasi, vrati stablo sa jednim čvorom, *koren*, koji je numerisan brojem klase kojoj pripadaju trening instance.

- Inače

  hiperravan = Inicijalizuj_hiperravan (*trening_skup*)

  Koristeći *HereBoy* algoritam i fitnes funkciju, pronađi optimalnu poziciju hiperravni unutar hiperprostora definisanog atributima problema, odnosno pronađi optimalne vrednosti koeficijenata iz jednačine (2.2) za koje će *impurity* funkcija imati minimalnu vrednost.

  Koristeći optimalnu hiperravan, podeli skup *trening_skup* na dva podskupa, jedan u kome će se nalaziti sve instance locirane iznad optimalne hiperravni, $trening\_skup_{iznad}$, i drugi, u kojem će se nalaziti sve instance koje su locirane ispod optimalne hiperravni, $trening\_skup_{ispod}$. Smesti ova dva skupa u skup *asocirane_instance*.

  broj_čvorova_za_obradu = 2

- Saopšti relevantne informacije (optimalne koeficijente hiperravni i podatke o čvorovima naslednicima)
- Sve dok je *broj_čvorova_za_obradu* nije jednak nuli

  broj_novih_čvorova = 0

  nove_asocirane_instance = prazan skup

  Za svaki čvor od ukupno *broj_čvorova_za_obradu*

  - *trening_skup* = sledeći skup iz skupa *asocirane_instance*

  - Ako sve instance iz skupa *trening_skup* pripadaju istoj klasi, tekući čvor postaje list kome je pridružen kod klase kojoj pripadaju trening instance

  - Inače

    hiperravan = Inicijalizuj_hiperravan (*trening_skup*)

    Koristeći *HereBoy* algoritam i fitnes funkciju, pronađi optimalnu poziciju hiperravni unutar hiperprostora definisanog atributima problema, odnosno pronađi optimalne vrednosti koeficijenata iz jednačine (2.2) za koje će *impurity* funkcija imati minimalnu vrednost.

    Koristeći optimalnu hiperravan, podeli skup *trening_skup* na dva podskupa, jedan u kome će se nalaziti sve instance locirane iznad optimalne hiperravni, $trening\_skup_{iznad}$, i drugi, u kojem će se nalaziti sve instance koje su locirane ispod optimalne hiperravni, $trening\_skup_{ispod}$. Smesti ova dva skupa u skup *nove_asocirane_instance*.

    broj_novih_čvorova = broj_novih_čvorova + 2

- Saopšti relevantne informacije (optimalne koeficijente hiperravni i podatke o čvorovima naslednicima)

broj_čvorova_za_obradu = broj_novih_čvorova

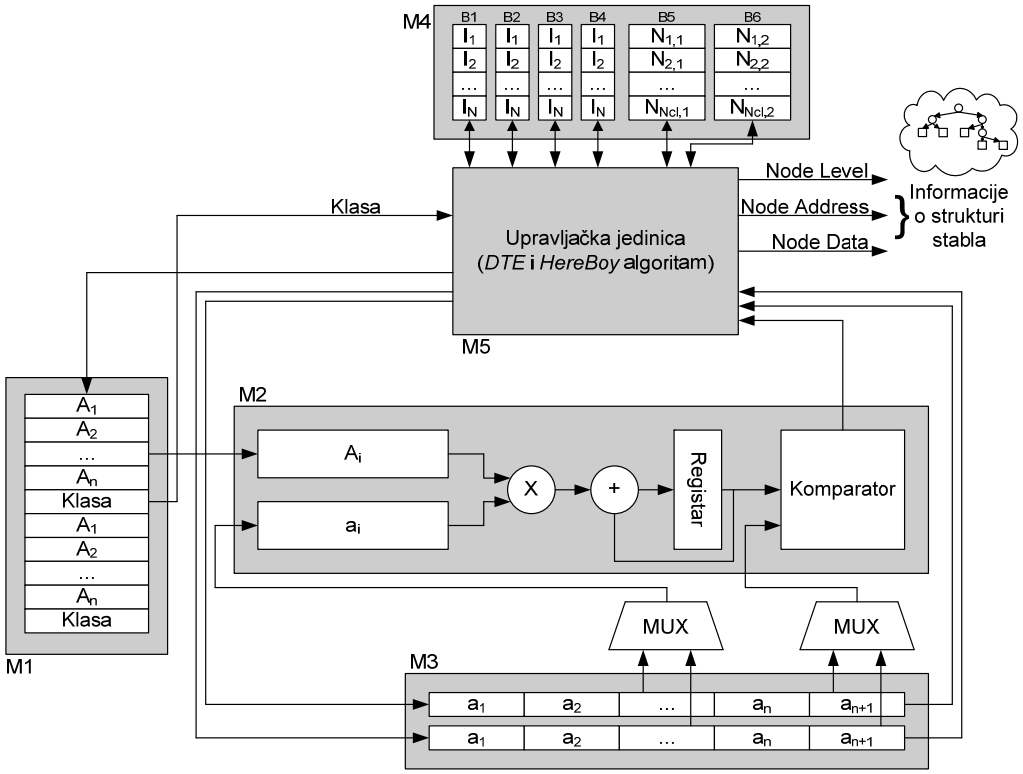
asocirane_instance = nove_asocirane_instance

- Vrati *koren*

Algoritam 3.2 DTS *Algoritam za formiranje neortogonalnog stabla odluke baziran na* breath-first *tehnici*

Glavna karakteristika DTS algoritma za formiranje neortogonalnih stabala odluka jeste da se za razliku od većine ostalih algoritama za formiranje stabla odluke stablo formira nivo po nivo, odnosno primenjuje se breath-first metoda. Ovakav pristup znatno olakšava manipulaciju skupovima trening instanci asociranih različitim čvorovima u stablu odluke, a time i hardversku realizaciju.

Na slici 3.2 prikazana je *H_DTS* arhitektura za hardversku implementaciju *DTS* algoritma. Pomoću ove arhitekture moguće je implementirati i *DTN* algoritam [78], u slučaju kada se kao nelinearne hiperpovrši koriste polinomijalne funkcije uz prethodno generisanje veštačkih atributa.



Slika 3.2 H_DTS *arhitektura za hardversku realizaciju* DTS *algoritma*

Kao što se slike 3.2 može videti, *H_DTS* arhitektura se sastoji od pet glavnih modula: memorije za smeštanje instanci trening skupa (*M1*), modula za evaluaciju instance (*M2*), memorije za smeštanje hromozoma (*M3*), memorija za čuvanje indeksa instanci i brojača (*M4*) i kontrolne jedinice (*M5*).

3.2.1 M1 Modul - Memorija za smeštanje instanci trening skupa

Memorija za smeštanje instanci trening skupa (*M1*) koristi se sa čuvanje trening skupa. Svaka instanca predstavljena je pomoću vektora od *n* konkretnih vrednosti atributa praćenih sa klasom kojoj instanca pripada. Ovo je organizacija u slučaju da se želi formirati neortogonalno stablo odluke. U slučaju da se želi formirati nelinearno stablo odluke, onda se vektoru vrednosti originalnih atributa mora pridružiti dodatni niz vrednosti veštački generisanih atributa. Veličina memorija za smeštanje trening skupa instanci određena je sledećim izrazom.

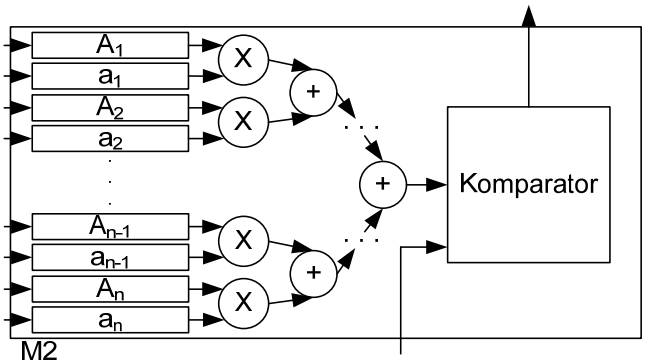
$$RAM\_Size_{M1} = N_{ts} \cdot (n+1) \tag{3.1}$$

U izrazu (3.1), sa N_{ts} je označen broj instanci u trening skupu. Svaka lokacija u ovom memoriji je veličine $\max\{N_a, \lceil ld(N_{cl})\rceil\}$ bita, gde je N_a broj bitova koji se koristi za predstavljanje vrednosti atributa, a N_{cl} označava broj različitih klasa kojima mogu pripadati instance problema.

3.2.2 M2 Modul - Modul za određivanje pozicije instance u odnosu na hiperravan

Modul *M2* koristi se za određivanje pozicije instanci u odnosu na tekuću poziciju hiperravni koja se optimizuje. Ova informacija je neophodna da bi se mogla odrediti vrednost *impurity* funkcije definisane izrazom (3.2). Modul *M2* koristi jedan N_axN_a-bitni množač, jedan (N_a+N_c)xN_r-bitni sabirač i N_r-bitni registar da serijski evaluira izraz (2.2). Veličina registra za čuvanje konačnog rezultata evaluacije izraza (2.2), N_r bita, mora biti odabrana na takav način da se obezbedi da ukupna akumulirana greška tokom serijske evaluacije izraza (2.2) bude dovoljno mala. Za određivanje pozicije instance u odnosu na hiperravan koristi se jedan N_cxN_c-bitni komparator, pri čemu je sa N_c označen broj bitova koji se koristi za predstavljanje vrednosti koeficijenata hiperravni.

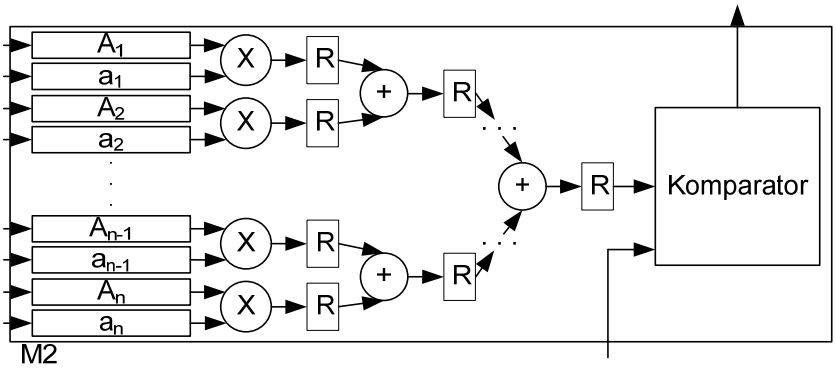
Korišćenjem ovakvog modula za evaluaciju instanci, određivanje položaja instance u odnosu na hiperravan traje ukupno *n+1* taktova. Ovo je posledica serijske evaluacije jednačine (2.2). Sa druge strane, evaluacija pozicija instanci predstavlja kritični korak u čitavom *DTS* algoritmu za formiranje neortogonalnog stabla odluke. Ukoliko se želi ubrzati rad *DTS* algoritma, jasno je da treba ubrzati određivanje pozicije instanci. Jedan od načina na koji se ova evaluacija može ubrzati jeste da se jednačina (2.2) evaluira u paraleli. Ukoliko bi se čitava suma izračunala odjednom, tada bi određivanje pozicije instance u odnosu na hiperravan trajalo samo jedan takt umesto *n+1* taktova. Na slici 3.3 prikazana je nova arhitektura modula *M2* koja omogućuje paralelnu evaluaciju jednačine (2.2).



Slika 3.3 Arhitektura modula M2 *koja omogućava paralelnu evaluaciju pozicije instance*

Jasno je da ovakva arhitektura *M2* modula zahteva daleko veći broj resursa kada se uporedi sa arhitekturom za serijsku evaluaciju pozicije instance koja je predstavljena ranije. Paralelna arhitektura zahteva *n* množača i *n-1* sabirača ogranizovanih u stablo za izračunavanje pozicije instance u samo jednom taktu.

Iako predložena arhitektura za paralelnu evaluaciju pozicije instance teorijski nudi ubrzanje od *n+1* puta, u praksi će ono biti znatno manje, usled toga što serijska i paralelna arhitektura ne mogu da radi sa istom učestanošću takta. Serijska arhitektura može da radi na daleko većoj učestanosti jer je kašnjenje duž kritičnog kombinacionog puta u slučaju serijske arhitekture jednako zbiru kašnjenja kroz jedan množač i jedan sabirač. U slučaju paralelne arhitekture ovo kašnjenje je značajno veće i jednako je zbiru kašnjenja kroz jedan množač i $\lceil ld(n)\rceil$ sabirača. Jedan od načina da se omogući da i paralelna arhitektura radi na višim učestanostima takta jeste da se dodatno modifikuje korišćenjem tehnika protočne obrade podataka. Umetanjem registara nakon nivoa množača i nakon svakog nivoa u stablu sabirača, kašnjenje duž kritičnog puta može se drastično smanjiti na kašnjenje samo kroz jedan množač. Paralelna arhitektura za evaluaciju pozicije instance sa protočnom obradom prikazana je na sledećoj slici.



Slika 3.4 Arhitektura modula M2 *za paralelnu evaluaciju pozicije instance bazirana na protočnoj obradi podataka*

Iako protočna arhitektura omogućava povećanje učestanosti na kojoj sistem može raditi, mora se napomenuti da ona unosi dodatno kašnjenje u sistem i zahteva složeniju upravljačku jedinicu. Međutim, zbog izrazite regularnosti operacije (evaluacija instanci) koja je realizovana korišćenjem tehnika protočne obrade, u ovom slučaju navedena dva nedostatka ne dolaze do izražaja.

3.2.3 M3 Modul - Memorija za čuvanje hromozoma

Modul *M3* predstavlja memoriju u kojoj se čuva hromozom koji kodira najbolju do sada pronađenu hiperravan i hromozom koji kodira trenutnu hiperravan koja se razmatra. Dakle, ova memorija zapravo čuva podatke o dve hiperravni u svakom trenutku. Veličina potrebne memorije zavisi od problema koji se rešava i može se odrediti pomoću sledeće jednačine.

$$RAM\_Size_{M3} = 2 \cdot (n+1) \tag{3.2}$$

Za kodiranje pozicije svake od dve hiperravni potrebno je kodirati vrednosti *n+1* koeficijenata, pri čemu se svaki koeficijent kodira pomoću N_c bita.

3.2.4 M4 Modul - Memorija za čuvanje indeksa asociranih instanci

Modul *M4* sastoji se od šest memorija. Četiri memorije koriste se za smeštaj indeksa instanci koje su asocirane čvorovima lociranim na tekućem nivou i sledećem nivou u stablu koje se formira. Veličina svake od ove četiri memorije određena je sledećom jednačinom

$$RAM\_Size_{M4\_B1-B4} = N_{ts} \tag{3.3}$$

Za predstavljanje vrednosti indeksa potrebno je $\lceil ld(N_{ts}) \rceil$ bita.

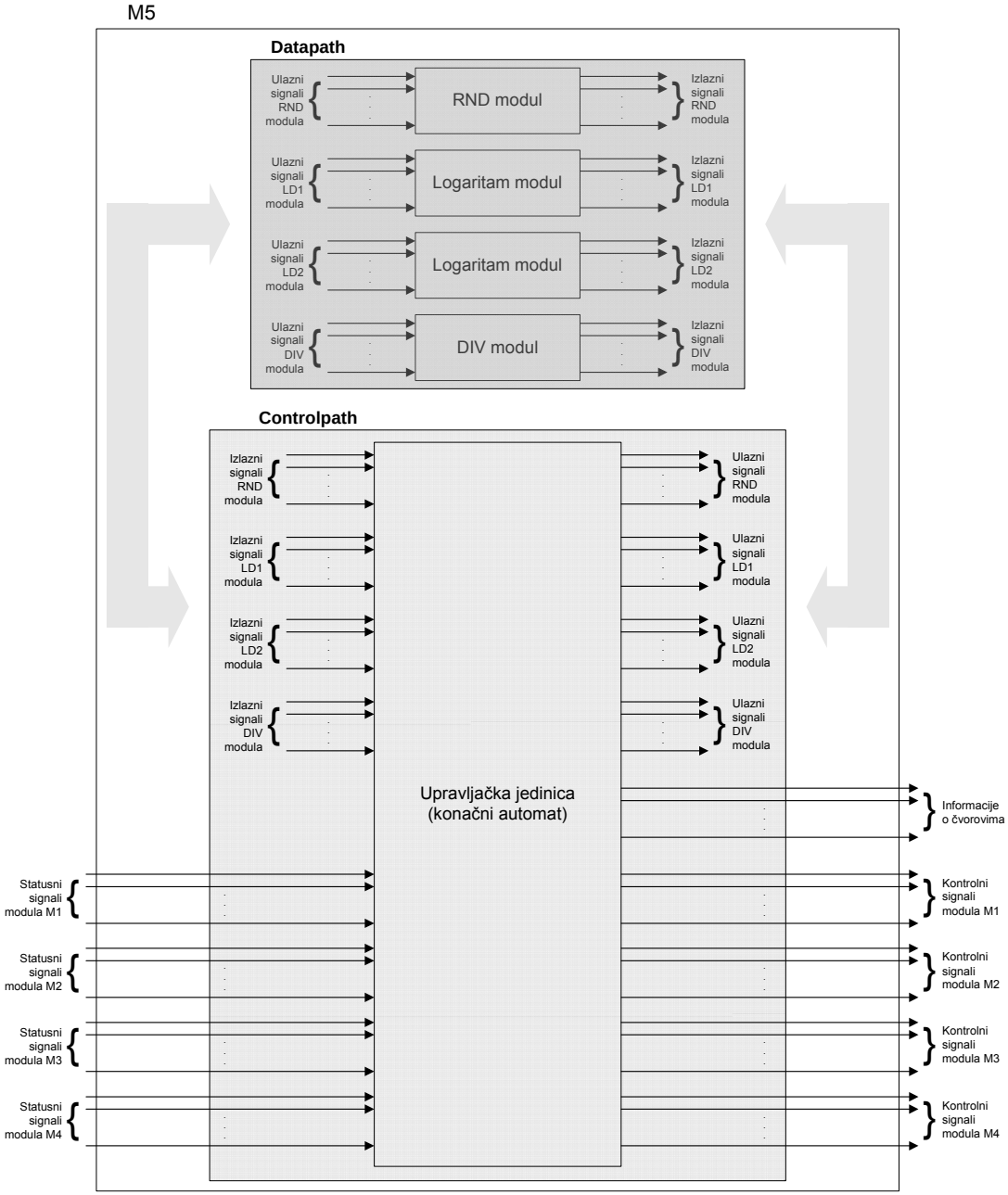
Da bi se mogla izračunati vrednost fitnes funkcije tokom optimizacije hiperravni u svakom od čvorova, pored indeksa asociranih instanci potrebno je znati koliko instanci iz svake klase je asocirano posmatranom čvoru. Ove informacije se čuvaju u dodatne dve banke iz modula M4. Veličina ove dve dodatne banke može se izračunati korišćenjem sledeće formule.

$$RAM\_Size_{M4\_B5-B6} = N_{cl} \tag{3.4}$$

Za predstavljanje ovih vrednosti potrebno je takođe po $\lceil ld(N_{ts}) \rceil$ bita.

3.2.5 M5 Modul - Upravljačka jedinica

Modul M5 je upravljačka jedinica zadužena za obezbeđivanje korektnog funkcionisanja čitavog sistema. Upravljačka jedinica izvodi korake definisane *DTS* algoritmom, a pored toga izvodi i korake definisane *HereBoy* algoritmom [79]. Upravljačka jedinica je zapravo jedan složeni digitalni sistem sastavljen od konačnog automata koji implementira *DTS* algoritam i odgovarajućeg broja modula koji služe za izvođenje aritmetičkih operacija koje su neophodne. Struktura modula M5 prikazana je na slici 3.5.



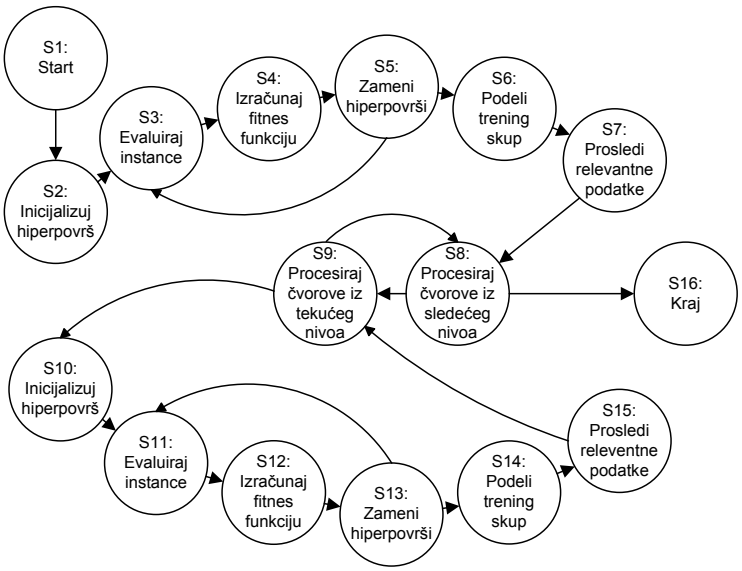
Slika 3.5 Detaljna struktura modula M5

Obzirom da *HereBoy* algoritam kao i svaki evolutivni algoritam, u svom radu koristi slučajne brojeve, modul M5 sadrži poseban modul namenjen generisanju pseudoslučajnih brojeva, nazvan *RND* modul na slici 3.5.

Pored *RND* modula, M5 modul sadrži i instance modula za izračunavanje logaritma, *Logaritam* modul, kao i jedan modul za izvođenje operacije deljenja, *DIV* modul. Ovi moduli su neophodni da bi se mogla izračunati vrednost fitnes funkcije.

Upravljačka jedinica

Upravljačka jedinica realizovana je kao konačni automat čiji je uprošćeni dijagram stanja i prelaza prikazan na slici 3.6.



Slika 3.6 Uprošćeni dijagram stanja upravljačke jedinice

Dijagram stanja prikazan na slici 3.6 prikazuje samo najvažnije korake koje izvodi upravljačka jedinica tokom svog rada. Funkcija koja se izvodi u svakom od stanja takođe je naznačena na slici 3.6. Svako prikazano stanje zapravo se sastoji iz većeg broja stanja koja zajedno ostvaruju posmatranu funkciju.

Konačni automat može se podeliti na dve celine. Prva celina, koju čine stanja S2-S7, optimizuje koren stabla odluke. Nakon završene optimizacije korena stabla, upravljački automat ulazi u jedan cikličan proces optimizacije čvorova iz jednog po jednog sloja stabla odluke. U trenutku kada u narednom sloju nema čvorova koje je potrebno optimizovati upravljački automat završava svoj rad.

Prilikom optimizacije korena stabla, prvi korak predstavlja inicijalizacija hiperravni asocirane korenu stabla izvršavanjem *Inicijalizuj_hiperravan* algoritma. Tokom inicijalizacije hiperravni asocirane korenu stabla upravljački automat nalazi se u stanju S2. Nakon izvršene inicijalizacije hiperravni, upravljački automat prelazi na optimizaciju njenog položaja izvršavanje *HereBoy* algoritma tokom specificiranog broja iteracija. U svakoj iteraciji, određuju se pozicije instanci asociranih korenu stabla (u ovom slučaju to su sve instance iz originalnog trening skupa) u odnosu na tekući položaj hiperravni, stanje S3. Korišćenjem ove informacije, upravljački automat u sledećem koraku računa vrednost fitnes funkcije za tekuću poziciju hiperravni, stanje S4. Izračunata vrednost fitnes funkcije za tekuću hiperravan se zatim poredi sa najboljom vrednošću fitnes funkcije koja je do tog trenutka postignuta, stanje S5. U ovom stanju se i izvodi skladištenje koeficijenata tekuće hiperravni u okviru modula M3 koja samim tim postaje i najbolja hiperravan do sada, ukoliko su ispunjeni uslovi definisani u *HereBoy* algoritmu. Ovaj niz koraka, definisan stanjim S3, S4 i S5, ponavlja se odgovarajući broj puta, koji zavisi od zadatog broja iteracija koji je potrebno izvršiti. Nakon izvršavanja zadatog broja iteracija, završava se faza optimizacije položaja hiperravni u korenu stabla i prelazi se u naredno stanje, S6, u kojem se vrši podela instanci iz trening skupa na dva podskupa, podskup instanci koje se nalaze sa jedne i podskup instanci koje se nalaze sa druge strane optimalne hiperravni. Za smeštanje indeksa instanci lociranih sa jedne, odnosno druge strane optimalne hiperravni koriste se memorijske banke B1-B4 iz modula M4, na način koji je opisan u odeljku 3.2.4. Pored toga, u stanju S6 kreiraju se i dva naslednika korena stabla koji su locirani u narednom sloju stabla odluke. Poslednji korak u formiranju korena stabla jeste prosleđivanje relevantnih informacija o korenu stabla na izlaze čitavog sistema, stanje S7. Po unapred određenom protokolu preko izlaza sistema šalju se vrednosti koeficijenata optimalne hiperravni i podaci o dva naslednika korena stabla.

Kao što je ranije rečeno, nakon završene optimizacije korena stabla, upravljački automat ulazi u petlju u kojoj se vrši optimizacija čvorova iz sukcesivnih nivoa stabla odluke. U stanju S8 proverava se da li postoje čvorovi iz narednog sloja koje je potrebno optimizovati. Ukoliko takvi čvorovi ne postoje, to znači da je proces formiranja stabla završen i upravljački automat prelazi u poslednje stanje i završava svoj rad. Ukoliko postoji barem jedan čvor iz narednog sloja koji je potrebno optimizovati automat prelazi u stanje S9. Za svaki čvor iz tekućeg nivoa stabla vrši se optimizacija pozicije hiperravni koja mu je asocirana na identičan način koji je bio opisan u slučaju korena stabla. Jedina razlika je u tome što se ovaj put prilikom optimizacije položaja hiperravni ne koristi čitav trening skup, već samo njegov podskup instanci koje su asocirane tekućem čvoru koji se optimizuje. Indeksi asociranih instanci nalaze se smešteni u jednoj od četiri banke B1-B4 unutar modula M4. Optimizacija položaja hiperravni počinje njenom inicijalizacijom, stanje S10, a zatim se tokom unapred definisanog broja iteracija vrši određivanje položaja svake od asociranih instanci u odnosu na tekuću poziciju hiperravni, stanje S11, računanje vrednosti fitnes funkcije, stanje S12 i smeštanje vrednosti koeficijenata u memoriju unutar modula M3 ukoliko je to potrebno, stanje S13. Nakon što se završi

optimizacija položaja hiperravni za dati čvor, vrši se podela asociranih instanci na dva podskupa, stanje S14. Optimizacija tekućeg čvora stabla odluke završava se postavljanjem relevantnih informacija o tekućem čvoru i njegovim naslednicima na izlaze čitavog sistema, stanje S15. Kao i u slučaju korena stabla, preko izlaza sistema šalju se vrednosti koeficijenata optimalne hiperravni i podaci o dva naslednika. Nakon toga upravljački automat vraća se u stanje S9. Ukoliko ne postoje čvorovi iz tekućeg nivoa stabla koje je potrebno optimizovati, iz stanja S9 vraćamo se u stanje S8. U stanju S8, kao što je ranije već rečeno, vrši se provera da li u narednom sloju stabla odluke postoje čvorovi koje je neophodno optimizovati. Ukoliko ih ima, upravljački automat ponovo prelazi u stanje S9 i započinje optimizaciju čvorova iz narednog nivoa stabla. Ukoliko ih nema, upravljački automat prelazi u stanje S16 i završava svoj rad. Ovim je i postupak formiranja stabla odluke završen.

3.3 Potrebni hardverski resursi za implementaciju i performanse *H_DTE* arhitektura

Prilikom hardverske implementacije bilo kog algoritma, od interesa je proceniti potrebne hardverske resurse. Najčešće se ovi resursi izražavaju u terminima veličine neophodnih memorijskih resursa i neophodnih resursa za realizaciju zahtevanih aritmetičkih operacija, tipično izraženih u broju sabirača, množača, delitelja, itd. U slučaju hardverske implementacije *H_DTE* algoritma neophodni hardverski resursi biće izraženi u veličini neophodne memorije i u broju neophodnih sabirača i množača.

3.3.1 Teorijska analiza potrebnih resursa i performansi

Kao što je prilikom predstavljanja *H_DTE* arhitektura naglašeno, svaka od ovih arhitektura može se realizovati u dve varijante, u zavisnosti od toga kako se realizuje modul M2, zadužen za određivanje pozicije trening instanci u odnosu na hiperpovrš u svakom od čvorova stabla odluke. Pozicija instance može se izračunati serijski ili paralelno. Shodno tome, prilikom analize neophodnih hardverskih resursa i performansi razmatrane su četiri različite arhitekture:

1. H_DTE1 – arhitektura koja realizuje DTE algoritam za formiranje ansambla stabala odluka, pri čemu se pozicija instance u odnosu na hiperpovrš računa serijski
2. H_DTE2 – arhitektura koja realizuje DTE algoritam za formiranje ansambla stabala odluka, pri čemu se pozicija instance u odnosu na hiperpovrš računa paralelno

Neophodni hardverski resursi, kao i performanse četiri navedene arhitekture izraženi su u terminima:

- ukupnog broja instanci trening skupa, N_{ts}
- broja atributa posmatranog klasifikacionog problema, n
- broja klasa posmatranog klasifikacionog problema, N_{cl}
- broja stabala u ansamblu, N
- procenta instanci iz trening skupa koji se koristi za formiranje svakog stabla iz ansambla, P
- broja instanci iz trening skupa asociranih i-tom čvoru stabla odluke, N_i
- dubini stabla odluke, M
- broja čvorova u stablu odluke, N_{dt}
- broja bitova koji se koriste za reprezentaciju vrednosti atributa klasifikacionog problema, N_a
- broja bitova koji se koriste za reprezentaciju vrednosti koeficijenata hiperpovrši u svakom od čvorova stabla odluke, N_c
- broju iteracija *HereBoy* algoritma koje se izvrše prilikom optimizacije položaja hiperpovrši u svakom o čvorova stabla odluke, N_{HB}
- trajanja periode globalnog sinhronizacionog signala (takta), CP

Tabela 3.1 sadrži podatke o neophodnim hardverskim resursima za implementaciju *H_DTE* arhitektura. U koloni koja sadrži podatke o potrebnim memorijski resursima za svaku od četiri arhitekture navedena su po tri podatka. Prvi se odnosi na veličinu potrebne memorije za smeštanje instanci trening skupa, koja se nalazi unutar modula M1. Drugi podatak odnosi se na veličinu memorije potrebne za smeštanje hromozoma koji koduju poziciju najbolje i tekuće hiperpovrši, modul M3. Treći podatak odnosi se na veličinu memorije neophodne za čuvanje indeksa asociranih instanci, modul M4. Treća i četvrta kolona prikazuju broj neophodnih sabirača i množača koji se nalaze unutar modula M2.

Tabela 3.1 Potrebni resursi za hardversku implementaciju *H_DTE* arhitektura

Arhitektura	Memorija	Sabirači	Množači
H_DTE1	$N \cdot P \cdot N_{ts} \cdot (n+1) \cdot N_a$, $N \cdot 2 \cdot (n+1) x N_c$, $N \cdot (4 \cdot P \cdot N_{ts} + 2N_{cl}) x \lceil ld(P \cdot N_{ts}) \rceil$	N	N
H_DTE2	$N \cdot P \cdot N_{ts} \cdot (n+1) \cdot N_a$, $N \cdot 2 \cdot (n+1) x N_c$, $N \cdot (4 \cdot P \cdot N_{ts} + 2N_{cl}) x \lceil ld(P \cdot N_{ts}) \rceil$	$N \cdot (n-1)$	$N \cdot n$

Na osnovu tabele 3.1 može se zaključiti sledeće. Većina memorijskih resursa ima linearan porast po svakom od relevantnih parametara. Izuzetak čini samo memorija za čuvanje indeksa asociranih instanci, smeštena unutar modula M4. Njena veličina raste kao $N_{ts} \cdot ld(N_{ts})$ po parametru N_{ts} koji predstavlja ukupan broj instanci unutar trening skupa. Međutim i ova stopa rasta je još uvek prihvatljiva jer nije eksponencijalna. Što se tiče stope rasta neophodnog broja sabirača i množača, analizom podataka može se zaključiti da je ona u najgorem slučaju linearna po svakom od relevantnih parametara. Može se zaključiti da se sve četiri predložene arhitekture izuzetno dobro skaliraju sa porastom složenosti problema koji se rešava, što predstavlja izuzetno dobru osobinu pogotovo u slučaju hardverske implementacije.

Analizom *H_DTE* arhitektura može se zaključiti da *H_DTE1* i *H_DTE2* arhitekture imaju identične memorijske zahteve, ali da se broj neophodnih sabirača i množača drastično razlikuje. Kao što se moglo i očekivati, *H_DTE2* arhitektura koja koristi paralelni način evaluacije pozicije instance u odnosu na hiperravan zahteva značajno veći broj množača i sabirača u poređenju sa *H_DTE1* arhitekturom. *H_DTE1* arhitektura ima izuzetno dobru osobinu da je broj neophodnih sabirača i množača nezavisan od bilo kojeg parametra problema. Prilikom korišćenja *H_DTE1* arhitekture za formiranje stabla odluke uvek je neophodan jedan sabirač i jedan množač, nezavisno od karakteristika problema koji se trenutno rešava.

Pored analize neophodnih hardverskih resursa za implementaciju *H_DTE* arhitektura od velikog značaja jeste i određivanje vremena potrebnog za formiranje stabla odluke u slučaju korišćenja bilo koje od četiri predložene arhitekture. Tabela 3.2 sadrži podatke o vremenima potrebnim za formiranje stabla odluke odnosno ansambala stabala odluka za svaku od četiri predložene arhitekture.

Tabela 3.2 Vreme potrebno za formiranje stabla odluke odnosno ansambla korišćenjem *H_DTE* arhitektura

Arhitektura	Vreme formiranja (najgori slučaj)
H_DTE1	$\max_{i=1}^{N} \left\{ \left[\sum_{j=1}^{N_{dt_i}} \left[(2n+2) \cdot N_{ts_{i,j}} + 3n + 8 + N_{HB} \left[(n+2) \cdot N_{ts_{i,j}} + N_{cl} + n + 6 \right] \right] + M_i \right] \cdot CP \right\}$
H_DTE2	$\max_{i=1}^{N} \left\{ \left[\sum_{j=1}^{N_{dt_i}} \left[2N_{ts_{i,j}} + 3n + 12 + ld(n) + N_{HB} \left[N_{ts_{i,j}} + N_{cl} + n + 7 + ld(n) \right] \right] + M_i \right] \cdot CP \right\}$

U tabeli 3.2 prikazani su egzaktni izrazi za vreme potrebno za formiranje stabla odluke ili ansambla stabala odluke. U slučaju *H_DTE1* arhitekture izraz iz tabele 3.2 može se transformisati na sledeći način.

$$
\begin{aligned}
T_{H\_DTE1} &= O\left(\max_{i=1}^{N} \left\{ \left[\sum_{j=1}^{N_{dt_i}} \left[(2n+2) \cdot N_{ts_{i,j}} + 3n + 8 + N_{HB} \left[(n+2) \cdot N_{ts_{i,j}} + N_{cl} + n + 6 \right] \right] + M_i \right] \cdot CP \right\} \right) = \\
&= O\left(\max_{i=1}^{N} \left\{ \sum_{j=1}^{N_{ts_i}} \left[(2n+2) \cdot j + 3n + 8 + N_{HB} \left[(n+2) \cdot j + N_{cl} + n + 6 \right] \right] \right\} \right) = \\
&= O\left(\max_{i=1}^{N} \left\{ N_{HB} (n+2) \cdot \frac{N_{ts_i} \cdot (N_{ts_i} + 1)}{2} \right\} \right) = \\
&= O\left(N_{HB} \cdot \max_{i=1}^{N} \left\{ N_{ts_i}^2 \right\} \cdot n \right)
\end{aligned}
\tag{3.5}
$$

Izraz (3.5) izveden je pod pretpostavkom da razmatramo najgori mogući slučaj formiranja stabla odluke. U ovom slučaju broj čvorova formiranog stabla odluke N_{dt} odgovara broju instanci trening skupa N_{ts}, a distribucija instanci trening skupa po čvorovima stabla linearno opada kako se krećemo od korena ka dnu stabla.

U slučaju *H_DTE2* arhitekture sličnim transformacijama može se doći do sledeće procene vremenske kompleksnosti.

$$
\begin{aligned}
T_{H\_DTE2} &= O\left(\max_{i=1}^{N} \left\{ \left\lceil \sum_{j=1}^{N_{dt_i}} \left[2N_{ts_{i,j}} + 3n + 12 + ld(n) + N_{HB} \left[N_{ts_{i,j}} + N_{cl} + n + 7 + ld(n) \right] \right] + M_i \right\rceil \cdot CP \right\} \right) = \\
&= O\left(\max_{i=1}^{N} \left\{ \sum_{j=1}^{N_{ts_i}} \left[2j + 3n + 12 + ld(n) + N_{HB} \left[j + N_{cl} + n + 7 + ld(n) \right] \right] \right\} \right) = \\
&= O\left(\max_{i=1}^{N} \left\{ N_{HB} \cdot \frac{N_{ts_i} \cdot \left(N_{ts_i} + 1 \right)}{2} \right\} \right) = \\
&= O\left(N_{HB} \cdot \max_{i=1}^{N} \left\{ N_{ts_i}^2 \right\} \right)
\end{aligned}
\tag{3.6}
$$

Vremenska kompleksnost *H_DTE2* arhitekture ne zavisi od broja atributa klasifikacionog problema koji se rešava, *n*. Do ovoga poboljšanja vremenske kompleksnosti došlo je stoga što se u slučaju *H_DTE2* arhitekture evaluacija pozicije instance u odnosu na hiperpovrš izračunava u paraleli, a ne sekvencijalno. Ovo ubrzanje je „plaćeno“ značajnim povećanjem hardverske kompleksnosti, što se može videti u tabeli 3.1, kao što je i bilo očekivano.

4. Primena

Nakon teorijske analize potrebnih hardverskih resursa i brzine klasifikacije *SMPL* i *UN* arhitektura, interesantno je odrediti njihove performanse i na realnim problemima iz prakse. Za ove poterbe korišćen je skup od ukupno 29 standardnih test problema odabranih iz standardne baze podatake za testiranje algoritama mašinskog učenja, „*UCI Machine Learning Repository*“, [80]. Glavne karakteristike odabranih problema prikazane su u tabeli 4.1.

Tabela 4.1 Karakteristike realnih klasifikacionih problema preuzetih iz UCI baze podataka

Problem	Oznaka	Broj atributa	Broj klasa	Broj instanci
Australian Credit Approval	AUSC	14	2	690
Balance Scale	BC	4	3	625
Breast Cancer Wisconsin	BCW	9	2	699
Breast Cancer	BSC	9	2	264
Car Evaluation	CAR	6	4	1728
Contraceptive Method Choice	CMC	9	3	1333
German Credit	GER	24	2	1000
Glass Identification	GLS	9	6	214
Hepatitis	HEP	19	2	155
Cleveland Heart Disease	HRTC	13	5	297
Statlog Heart Disease	HRTS	13	2	270
Ionosphere	ION	34	2	351
Iris	IRS	4	3	150
Liver Disorders	LIV	6	2	345
Lymphography	LYM	18	4	148
Page Blocks	PAGE	10	5	5427
Pima Indians Diabetes	PID	8	2	768
Sonar	SON	60	2	208
Thyroid Disease	THY	5	3	215
Tic-Tac-Toe Endgame	TTT	9	2	958
Statlog Vehicle Silhouettes	VEH	18	4	846
Congressional Voting Records	VOTE	16	2	232
Vowel Recognition	VOW	13	11	990
Waveform21	W21	21	3	5000
Waveform40	W40	40	3	5000
Wisconsin Diagnostic Breast Cancer	WDBC	30	2	569
Wine Recognition	WINE	13	3	178
Wisconsin Prognostic Breast Cancer	WPBC	33	2	194
Zoo	ZOO	17	7	101

Za svaku od arhitektura razvijen je RTL model pomoću VHDL jezika za opis hardvera koji je zatim sintetizovan pomoću *Xilinx ISE Foundation* 13.1 programskog paketa kompanije *Xilinx*. Razvijeni RTL modeli projektovani su sa mogućnošću parametrizacije, što omogućava njihovo jednostavno prilagođavanje karakteristikama problema koji se trenutno rešava. *Entity* deklaracija za *H_DTE1* arhitekturu ima sledeći izgled.

```
entity hdte1 is
   generic (
      num_members_g: integer := 10;            -- number of ensemble members
      num_classes_g: integer := 2;             -- number of classes
      num_instances_g: integer := 6;           -- number of instances
      num_attributes_g: integer := 33;         -- number of attributes
      attribute_res_g: integer := 10;          -- number of bits used to represent attributes
      coef_res_g: integer := 20;               -- number of bits used to represent coefficients of the
                                                  hyperplanes
      index_res_g: integer := 10;              -- number of bits used to represent class
      count_res_g: integer := 10;              -- number of bits used to represent class
      class_res_g: integer := 10;              -- number of bits used to represent class
      att_mem_bus_size_g: integer := 10;       -- size of the address bus for the attribute memory
```

```
    coef_mem_bus_size_g: integer := 10;          -- size of the address bus for the coefficient memory
    index_mem_bus_size_g: integer := 10;         -- size of the address bus for the attribute memory
    count_mem_bus_size_g: integer := 10;         -- size of the address bus for the attribute memory
    node_rln_mem_bus_size_g: integer := 10;      -- size of the address bus for the rigth-left node memory
    node_coef_mem_bus_size_g: integer := 10; -- size of the address bus for the coefficient memory for the built
                                             --   node
    max_iterations_g: integer := 100;            -- number of iterations of the HereBoy algorithm in every node
    max_prob_g: real := 0.1                      -- maximum probability of the byte mutation  operation
        );

  port (
    clk:
        in  std_logic;                                              -- clock input

    node_level_o:
        out std_logic_vector (num_members_g *7 downto 0);           -- next node level in the decision
                                                                    -- tree
    node_coef_adr_o:
        out std_logic_vector (num_members_g *(node_coef_mem_bus_size_g – 1) downto 0); -- address of the next
                                                                                       -- coeff
    node_coef_o:
        out std_logic_vector (num_members_g *(coef_res_g – 1) downto 0);            -- next coeff value
    node_rln_adr_o:
        out std_logic_vector (num_members_g *(node_rln_mem_bus_size_g – 1) downto 0);   -- address of the right-left
                                                                                        -- node address memory
    node_new_rln_addr_o:
        out std_logic_vector (num_members_g *(2*node_rln_mem_bus_size_g – 1) downto 0); -- next node address
     node_class_adr_o:
        out std_logic_vector (num_members_g *(node_rln_mem_bus_size_g – 1) downto 0);   -- address of the class
                                                                                        -- memory
    node_class_o:
        out std_logic_vector (num_members_g *(2*class_res_g – 1) downto 0)          -- possible output class
      );
end entity hdte1;
```

Izgled entity *deklaracije u slučaju* H_DTE1 *arhitekture*

Kao ciljna familija na kojoj su implementirane predložene arhitekture odabrana je *Xilinx Virtex5* familija.

4.1 Neophodni hardverski resursi

Tabele 4.2-4.3 sadrže podatke o neophodnim hardverskim resursima za implementaciju *H_DTS* arhitektura za svaki od 29 odabranih UCI problema. Kao i ranije, hardverski resursi iskazani su veličinom potrebne memorije, pri čemu su odvojeno prikazane veličine memorija potrebnih za implementaciju M1, M2 i M3 modula, i brojem potrebnih sabirača odnosno množača. Svaka od dve tabele sadrži informacije o neophodnim hardverskim resursima za jednu *H_DTE* arhitekturu.

Tabela 4.2 Neophodni hardverski resursi u slučaju FPGA implementacije *H_DTE1* arhitekture za 29 odabranih UCI test problema

Problem	*M1 (kbits)*	*M2 (kbits)*	*M3 (kbits)*	*Sabirači*	*Množači*
ausc	90.97	8.79	12.71	30.00	30.00
bc	27.47	2.93	11.87	30.00	30.00
bsc	61.44	5.86	12.87	30.00	30.00
bcw	23.20	5.86	3.14	30.00	30.00
car	106.31	4.10	37.86	30.00	30.00
cmc	117.16	5.86	29.17	30.00	30.00
ger	219.73	14.65	18.16	30.00	30.00
gls	18.81	5.86	3.31	30.00	30.00
hep	27.25	11.72	1.99	30.00	30.00
hrtc	36.54	8.20	5.35	30.00	30.00
hrts	33.22	8.20	4.27	30.00	30.00
ion	107.97	20.51	5.40	30.00	30.00
irs	6.59	2.93	2.11	30.00	30.00
liv	21.23	4.10	5.32	30.00	30.00
lym	24.71	11.13	2.26	30.00	30.00
page	524.68	6.45	154.98	30.00	30.00
pid	60.75	5.27	14.09	30.00	30.00

son	111.52	35.74	2.55	30.00	30.00
thy	11.34	3.52	2.79	30.00	30.00
ttt	84.20	5.86	17.43	30.00	30.00
veh	141.28	11.13	16.04	30.00	30.00
vote	34.66	9.96	2.80	30.00	30.00
vow	121.82	8.20	20.63	30.00	30.00
w21	966.80	12.89	142.03	30.00	30.00
w40	1801.76	24.02	142.03	30.00	30.00
wdbc	155.03	18.16	10.59	30.00	30.00
wine	21.90	8.20	2.40	30.00	30.00
wpbc	57.97	19.92	2.40	30.00	30.00
zoo	15.98	10.55	1.53	30.00	30.00

Tabela 4.3 Neophodni hardverski resursi u slučaju FPGA implementacije *H_DTE2* arhitekture za 29 odabranih UCI test problema

Problem	***M1 (kbits)***	***M2 (kbits)***	***M3 (kbits)***	***Sabirači***	***Množači***
ausc	90.97	8.79	12.71	390.00	420.00
bc	27.47	2.93	11.87	90.00	120.00
bsc	61.44	5.86	12.87	240.00	270.00
bcw	23.20	5.86	3.14	240.00	270.00
car	106.31	4.10	37.86	150.00	180.00
cmc	117.16	5.86	29.17	240.00	270.00
ger	219.73	14.65	18.16	690.00	720.00
gls	18.81	5.86	3.31	240.00	270.00
hep	27.25	11.72	1.99	540.00	570.00
hrtc	36.54	8.20	5.35	360.00	390.00
hrts	33.22	8.20	4.27	360.00	390.00
ion	107.97	20.51	5.40	990.00	1020.00
irs	6.59	2.93	2.11	90.00	120.00
liv	21.23	4.10	5.32	150.00	180.00
lym	24.71	11.13	2.26	510.00	540.00
page	524.68	6.45	154.98	270.00	300.00
pid	60.75	5.27	14.09	210.00	240.00
son	111.52	35.74	2.55	1770.00	1800.00
thy	11.34	3.52	2.79	120.00	150.00
ttt	84.20	5.86	17.43	240.00	270.00
veh	141.28	11.13	16.04	510.00	540.00
vote	34.66	9.96	2.80	450.00	480.00
vow	121.82	8.20	20.63	360.00	390.00
w21	966.80	12.89	142.03	600.00	630.00
w40	1801.76	24.02	142.03	1170.00	1200.00
wdbc	155.03	18.16	10.59	870.00	900.00
wine	21.90	8.20	2.40	360.00	390.00
wpbc	57.97	19.92	2.40	960.00	990.00
zoo	15.98	10.55	1.53	480.00	510.00

Rezultati eksperimenata izvedenih sa 29 odabranih UCI test problema prikazani u tabelama 4.2-4.3 u saglasnosti su sa opštim izrazima iz tabele 3.1. Ovi rezultati takođe mogu da posluže za bolju procenu neophodnih hardverskih resursa u slučaju rešavanja konkretnih, realnih problema. Na osnovu podataka iz tabela 4.2-4.3 može se zaključiti da zahtevani resursi ne prevazilaze resurse koje obezbeđuju savremene FPGA komponente. Na primer, najveća FPGA komponenta iz familije *Virtex5*, *XC5VSX240T*, sadrži ukupno 18576 kilobita memorije i 1056 posvećenih hardverskih sabirača i množača. Pomoću ove komponente se bez većih problema mogu implementirati *H_DTE* arhitekture za svaki od 29 razmatranih UCI test problema.

Ukoliko analiziramo količinu neophodne veličine memorijskih resursa najzahtevnije su *H_DTE1* i *H_DTE2* arhitekture za *w40* test problem, koje zahtevaju po 1967.81 kilobita. Ovaj broj je daleko manji od ukupno 18576 kilobita memorije koje nam stoje na raspolaganju u *XC5VSX240T* komponenti.

Naviše sabirača zahteva *H_DTE2* arhitektura za *son* test problem, 1770. *XC5VSX240T* komponenta na raspolaganju ima ukupno 1056 hardverskih sabiračkih modula. Očigledno da ovaj broj ne bi bio dovoljan za implementaciju posmatrane *H_DTE2* arhitekture. Međutim ne treba gubiti iz vida činjenicu da nam pored hardverskih sabiračkih modula na raspolaganju stoji i programibilna logika unutar FPGA komponente

organizovana u konfigurabilne logičke blokove, *CLB*. Ovi blokovi se mogu konfigurisati i povezati tako da realizuju proizvoljni digitalni sistem, pa i sabirače koji su nama neophodni. *XC5VSX240T* komponenta nudi ukupno 18720 *CLB* blokova. Potreban broj *CLB* blokova da bi se realizovao jedan 10-bitni sabirač iznosi ne više od dva. U ovom konkretnom slučaju potrebno je oko 7200 *CLB* blokova da bi se implementiralo zahtevanih 3600 sabirača, što iznosi oko 40% ukupnih programibilnih resursa koje ova komponenta nudi.

Najveći broj množača potrebno je implementirati u slučaju *H_DTE2* arhitekture za *son* test problem, ukupno 1800. Situacija je slična kao i u slučaju sabirača, 1056 množača može se implementirati korišćenjem posvećenih hardverskih množača, dok se preostalih 744 mogu realizovati korišćenjem programabilne logike.

Već prilikom teorijske analize nagovešteno je da paralelne arhitektura *H_DTE2* zahteva daleko više sabirača i množača od serijskih. Takođe pokazano je da što se tiče potrebnih memorijskih resursa, obe *H_DTE* arhitekture imaju podjednake zahteve. Analizom eksperimentalnih rezultata prikazanih u tabelama 4.2-4.3, zaključci teorijske analize mogu se potvrditi. Ono što rezultati eksperimenata još bolje pokazuju jeste konkretan broj resursa koji su neophodni da bi se bilo koja od predloženih arhitektura realizovala u slučaju realnih problema. Na osnovu svih rezultata može se zaključiti da se *H_DTE1* arhitektura može koristiti bez većih problema, čak i u slučaju FPGA komponenti sa skromnijim mogućnostima. Izuzetak predstavlja *H_DTE2* arhitektura, koja zahteva značajno veći broj hardverskih resursa od *H_DTE1* arhitekture, pogotovo u zahtevanom broju sabirača i množača. Ovo se moglo i očekivati jer je u slučaju *H_DTE2* arhitekture izvršena paralelizacija svih operacija. Paralelno se formiraju stabla odluke iz ansambla, pri čemu se u svakom stablu prilikom optimizacije položaja hiperpovrši pozicije instanci evaluiraju korišćenjem paralelnog pristupa. Visoka cena koja se plaća u terminima hardverskih resursa trebalo bi da bude opravdana velikom brzinom formiranja ansambla u poređenju sa serijskom arhitekturom.

4.2 Performanse

Pored analize konkretnih hardverskih resursa koji su neophodni za realizaciju predloženih arhitektura za konkretne UCI test probleme, od interesa je takođe estimirati vremena potrebna za formiranje stabla odluke, odnosno ansambala stabala odluka za svaku od četiri predložene arhitekture. Takođe je interesantno uporediti dobijene rezultate sa rezultatima postignutim u slučaju softverske implementacije *DTE* algoritma koja se izvršava na savremenom desktop računaru. Za potrebe poređenja *DTS* algoritam implementiran je u C programskom jeziku. Implementacija u C jeziku je zatim kompajlirana korišćenjem *Microsoft Visual C++* kompajlera i izvršena na pod *Microsoft Windows XP* operativnim sistemom na PC računaru sa *Intel Pentium* D 820 procesorom koji je radio na 2.8 GHz i ukupno 3 GB operativne memorije.

Rezultati prikazani u tabeli 4.4 predstavljaju prosečna vremena formiranja individualnog stabla odluke, odnosno ansambla sastavljenog od 30 stabala odluke i slučaju softverske implementacije koja se izvršavala na PC računaru. Pored toga merena su i vremena formiranja i u slučaju hardverske implementacije *H_DTE1* i *H_DTE2,* arhitektura koje su bile implementirane na istoj FPGA komponenti familije *Virtex 5*. Za svaki od 29 UCI test problema, izvršeno je 50 eksperimenata. U svakom eksperimentu na slučajan način formiran je trening skup koji je činilo 70% raspoloživih instanci. Korišćenjem ovog trening skupa formirano je stablo odnosno ansambl pri čemu je mereno vreme potrebno za njegovo formiranje za svaku od razmatranih implementacija. Rezultati prikazani u tabeli 4.4 predstavljaju srednje vrednosti vremena formiranja prikupljenih tokom 50 iteracija za svaki test problem. Sva vremena izražena su u sekundama.

Tabela 4.4 Vreme potrebno za formiranje stabla odluke u slučaju softverske odnosno hardverske implementacije *DTE* algoritma

Test skup	**Ansambl stabala odluke**		
	PC [s]	***H_DTE1 [s]***	***H_DTE2 [s]***
ausc	50.01	0.97	0.09
bc	22.39	0.34	0.07
bcw	15.31	0.41	0.05
bsc	20.97	0.23	0.03
car	65.10	1.42	0.22
cmc	209.39	2.44	0.35
ger	128.41	2.55	0.15
gls	32.92	0.26	0.05
hep	7.26	0.06	0.01
hrtc	49.25	0.43	0.06
hrts	19.67	0.28	0.04
ion	32.38	0.72	0.04
irs	5.12	0.05	0.01
liv	26.73	0.30	0.06

lym	17.70	0.17	0.02
page	256.59	10.04	0.97
pid	56.97	0.86	0.12
son	38.56	0.68	0.03
thy	6.52	0.09	0.02
ttt	36.23	1.02	0.13
veh	129.47	2.09	0.17
vote	8.13	0.17	0.01
vow	170.20	2.20	0.28
w21	478.84	15.37	0.81
w40	811.66	28.10	0.93
wdbc	24.31	0.89	0.04
wine	10.99	0.12	0.01
wpbc	30.54	0.39	0.03
zoo	33.51	0.15	0.02

Već na osnovu rezultata prikazanih u tabeli 4.4 može se zaključiti da hardverske implementacije *DTE* algoritma nude značajna ubrzanja u odnosu na softversku implementaciju. Ovo ubrzanje bi bilo još značajnije ukoliko bi se softverska implementacija izvršavala na *embedded* mikroprocesoru. Da bi se bolje moglo proceniti ubrzanje hardverske implementacije u odnosu na softversku koje je moguće, tabela 4.5 sadrži estimirana ubrzanja u slučaju 29 odabranih UCI problema. Svaka od kolona prikazuje ubrzanje odgovarajuće arhitekture za hardversku implementaciju individualnog stabla odluke (*H_DTE1* i *H_DTE2*) u odnosu na softversku implementaciju (PC).

Tabela 4.5 Ubrzanje hardverske u odnosu na softversku implementaciju

Test skup	**Ansambl stabala odluke PC**	
	H_DTE1 [s]	***H_DTE2 [s]***
ausc	51.56	555.67
bc	65.85	319.86
bcw	37.34	306.20
bsc	91.17	700.07
car	45.85	295.91
cmc	85.82	598.26
ger	50.36	856.07
gls	126.62	658.40
hep	121.00	1209.01
hrtc	114.53	820.83
hrts	70.25	491.75
ion	44.97	809.50
irs	102.40	448.14
liv	89.10	445.50
lym	104.12	1034.91
page	25.56	264.53
pid	66.24	474.75
son	56.71	1486.28
thy	72.44	399.61
ttt	35.52	278.69
veh	61.95	761.59
vote	47.82	657.61
vow	77.36	607.86
w21	31.15	591.16
w40	28.88	872.75
wdbc	27.31	607.75
wine	91.58	985.65
wpbc	78.31	1018.00
zoo	223.40	1529.86
AVG	**73.28**	**692.63**

Rezultati prikazani u tabeli 4.5 dobijeni su pod sledećim uslovima:

- dve arhitekture za implementaciju *DTS* algoritma realizovane su pomoću FPGA komponente familije *Virtex 5* koja je u proseku radila na sledećim učestanostima:
    - *H_DTE1* arhitektura, 122 MHz
    - *H_DTE2* arhitektura, 113 MHz
- učestanost *Pentium D 820* mikroprocesora iznosila je 2.8 GHz,
- broj članova ansambla iznosio je 30.

Kao što se može videti iz tabele 4.5, obe arhitekture pružaju znatna ubrzanja u formiranju stabla odluke u odnosu na softverske implementacije.

U slučaju poređenja sa PC implementacijom, prosečno skraćenje vremena potrebnog za formiranje stabla odluke odnosno ansambla stabala odluka iznosilo je:

- 73.28 puta u slučaju korišćenja *H_DTE1* arhitekture,
- 692.63 puta u slučaju korišćenja *H_DTE2* arhitekture.

Čak i u poređenju sa jednim od trenutno najjačih desktop računara, svaka od predloženih arhitektura, implementirana pomoću FPGA komponenti, pruža značajna skraćenja u vremenu formiranja stabla odluke ili ansambla stabala odluka. Pored toga potrebno je napomenuti da su hardverske *H_DTE* implementacije radile u proseku na 23, odnosno 25 puta manjoj učestanosti od softverske implementacije.

Reference

[1] Quinlan J. R., Induction of decision trees, Machine Learning, 1, 1986., pp. 81-106.
[2] Quinlan J. R., C4.5: Programs for Machine Learning, San Mateo, CA: Morgan Kaufmann, 1993
[3] Shannon C. E., A mathematical theory of communication, Bell System Technical J., 27, 1948, pp. 379 423,623-656
[4] Sethi I. K., Sarvarayudu G. P. R., Hierarchical classifier design using mutual information, IEEE Trans. on Pattern Analysis and Machine Intelligence, PAMI-4(4), 1982, pp. 441-445
[5] Talmon J. L., A multiclass nonparametric partitioning algorithm, Pattern Recognition Letters, 4, 1986, pp. 31-38
[6] Hartmann C. R. P., Varshney P. K., Mehrotra K. G., Gerberich C. L., Application of information theory to the construction of efficient decision trees, IEEE Trans. on Information Theory, IT-28(4), 1982, pp. 565-577
[7] Wang Q. R., Suen C. Y., Analysis and design of a decision tree based on entropy reduction and its application to large character set recognition, IEEE Trans. on Pattern Analysis and Machine Intelligence, 6, 1984, pp. 406-417
[8] Breiman L., Friedman J., Olshen R., Stone C., Classification and Regression Trees, Wadsworth International Group, 1984.
[9] Gelfand S. B., Ravishankar C. S., Delp E. J., An iterative growing and pruning algorithm for classification tree design, IEEE Transaction on Pattern Analysis and Machine Intelligence, 13(2), 1991, pp. 163-174
[10] Lin Y. K., Fu K., Automatic classification of cervical cells using a binary tree classifier, Pattern Recognition, 16(1), 1983, pp. 69-80
[11] Friedman J. H., A recursive partitioning decision rule for nonparametric classifiers, IEEE Trans. on Comp., C-26, 1977, pp. 404-408
[12] Haskell R. E., Noui-Mehidi A., Design of hierarchical classifiers, In N. A. Sherwani, E. de Doncker, and J. A. Kapenga, editors, Computing in the 90's: The First Great Lakes Computer Science Conf. Proc., Berlin, 1991. Springer-Verlag. Conf. held in Kalamazoo, MI on 18th-20th, 1989, pp. 118-124
[13] Hart A., Experience in the use of an inductive system in knowledge eng., In M. Bramer, editor, Research and Development in Expert Systems. Cambridge Univ. Press, Cambridge, MA, 1984.
[14] Mingers J., Expert systems for rule induction with statistical data, J. of the Operational Research Society, 38(1), 1987, pp. 39-47
[15] Zhou X. J., Dillon T. S., A statistical-heuristic feature selection criterion for decision tree induction. IEEE Trans. on Pattern Analysis and Machine Intelligence, PAMI- 13(8), 1991, pp. 834-841
[16] Heath D., Kasif S., Salzberg S., "Induction of oblique decision trees", Proceedings of the 13th International Joint Conference on Artificial Intelligence, San Mateo, CA: Morgan Kaufmann, 1993, pp. 1002-1007.
[17] Henrichon E. G., Fu K., A nonparametric partitoning procedure for pattern classification, IEEE Trans. Comput., Vol. C-18, 1969, pp. 614-624
[18] Iyengar V. S., HOT: Heuristics for oblique trees, Eleventh International Conference on Tools with Artificial Intelligence, New York: IEEE Press, 1999, pp. 91-98
[19] You K. C., Fu K., An approach to the design of a linear binary tree classifier. In Proc. of the Third Symposium on Machine Processing of Remotely Sensed Data, West Lafayette, IN, 1976. Purdue Univ.
[20] Loh W. Y., Vanichsetakul N., Tree-structured classification via generalized discriminant analisys, J. Amer. Statist. Assoc., Vol. 83, no. 403, 1988, pp. 715-728
[21] Brown D. E., Pittard C. L., Park H., Classification trees with optimal multivariate decision nodes, Pattern Recognit. Lett., vol. 17, 1996, pp. 699-703
[22] Murthy S. K., Kasif S., Salzberg S., A system for induction of oblique decision trees, J. Artific. Intell. Res., vol. 2, no. 1, 1994, pp. 1-32
[23] Koza J. R., Concept formation and decision tree induction using the genetic programming paradigm, Parallel Problem Solving from Nature, Schwefel H. P. and Männer R., Eds. Berlin, Germany: Springer-Verlag, 1991, pp. 124-128
[24] Koza J. R., Genetic Programming: On the Programming of Computers by Means of Natural Selection, Cambridge, MA: The MIT Press, 1992
[25] Folino G., Pizzuti C., Spezzano G., A cellular genetic programming aproach to classification, Proceedings of the Genetic and Evolutionary Computation Conference1999: Volume 2, Banzhaf W., Daida J., Eiben A.

E., Garzon M. H., Honavar V., Jakiela M., Smith R. E., Eds., San Francisco, CA: Morgan Kufmann, 1999, pp. 1015-1020
[26] Bot M. C. J., Langdon W. B., Application of genetic programming to induction of linear decision trees, Genetic Programming: Third European Conference, Poli R., Banzhaf W., Langdon W. B., Miller J., Nordin P., Fogarty T. C., Eds. Berlin, Germany: Springer-Verlag, 2000, pp. 247-258
[27] Cantú-Paz E., Kamath C., Inducing Oblique Decision Trees With Evolutionary Algorithms, IEEE Trans. on Evolut. Comp., vol. 7, no. 1, 2003, pp. 54-67
[28] Minsky M., Papert S., Perceptrons. MIT Press, Cambridge, MA, 1969.
[29] Utgoff P. E., Perceptron trees: A case study in hybrid concept representations. Connection Science, 1(4):377-391, 1989.
[30] Utgoff P. E., Brodley C. E., An incremental method for finding multivariate splits for decision trees. In Proc. of the Seventh Int. Conf. on Machine Learning, pages 58-65, Los Altos, CA, 1990. Morgan Kaufmann.
[31] Sethi I. K., Yoo J. H., Design of multicategory, multifeature split decision trees using perceptron learning. Pattern Recognition, 27(7):939-947, 1994.
[32] Guo H., Saul B. Gelfand S. B., Classification trees with neural network feature extraction. IEEE Trans. on Neural Networks., 3(6):923-933, November 1992.
[33] Sethi I. K., Entropy nets: From decision trees to neural networks. Proc. of the IEEE, 78(10), October 1990.
[34] Ittner A., Schlosser M., Non-linear decision trees - NDT. In Int. Conf. on Machine Learning. 1996.
[35] B.V. Dasarathy and B.V. Sheela, “Composite classifier system design: Concepts and methodology,” Proceedings of the IEEE, vol. 67, no. 5, pp. 708–713, 1979.
[36] L.K. Hansen and P. Salamon, “Neural network ensembles,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 12, no. 10, pp. 993–1001, 1990.
[37] R.E. Schapire, “The strength of weak learnability,” Machine Learning, vol. 5, no. 2, pp. 197–227, 1990.
[38] R.A. Jacobs, M.I. Jordan, S.J. Nowlan, and G.E. Hinton, “Adaptive mixtures of local experts,” Neural Computation, vol. 3, pp. 79–87, 1991.
[39] M.J. Jordan and R.A. Jacobs, “Hierarchical mixtures of experts and the EM algorithm,” Neural Computation, vol. 6, no. 2, pp. 181–214, 1994.
[40] J.A. Benediktsson and P.H. Swain, “Consensus theoretic classification methods,” IEEE Trans. on Systems, Man and Cybernetics, vol. 22, no. 4, pp. 688–704, 1992.
[41] L. Xu, A. Krzyzak, and C.Y. Suen, “Methods of combining multiple classifiers and their applications to handwriting recognition,” IEEE Transactions on Systems, Man and Cybernetics, vol. 22, no. 3, pp. 418–435, 1992.
[42] T.K. Ho, J.J. Hull, and S.N. Srihari, “Decision combination in multiple classifier systems,” IEEE Trans. on Pattern Analy. Machine Intel., vol. 16, no. 1, pp. 66–75, 1994.
[43] G. Rogova, “Combining the results of several neural network classifiers,” Neural Networks, vol. 7, no. 5, pp. 777–781, 1994.
[44] L. Lam and C.Y. Suen, “Optimal combinations of pattern classifiers,” Pattern Recognition Letters, vol. 16, no. 9, pp. 945–954, 1995.
[45] K. Woods, W.P.J. Kegelmeyer, and K. Bowyer, “Combination of multiple classifiers using local accuracy estimates,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19, no. 4, pp. 405–410, 1997.
[46] I. Bloch, “Information combination operators for data fusion: A comparative review with classification,” IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans., vol. 26, no. 1, pp. 52–67, 1996.
[47] S.B. Cho and J.H. Kim, “Combining multiple neural networks by fuzzy integral for robust classification,” IEEE Transactions on Systems, Man and Cybernetics, vol. 25, no. 2, pp. 380–384, 1995.
[48] L.I. Kuncheva, J.C. Bezdek, and R. Duin, “Decision templates for multiple classifier fusion: An experimental comparison,” Pattern Recognition, vol. 34, no. 2, pp. 299–314, 2001.
[49] H. Drucker, C. Cortes, L.D. Jackel, Y. LeCun, and V. Vapnik, “Boosting and other ensemble methods,” Neural Computation, vol. 6, no. 6, pp. 1289–1301, 1994.
[50] R. Battiti and A.M. Colla, “Democracy in neural nets: Voting schemes for classification,” Neural Networks, vol. 7, no. 4, pp. 691–707, 1994.
[51] L.I. Kuncheva, “Classifier ensembles for changing environments,” 5th Int. Workshop on Multiple Classifier Systems, Lecture Notes in Computer Science, F. Roli, J. Kittler, and T. Windeatt, Eds., vol. 3077, pp. 1–15, 2004.
[52] E. Alpaydin and M.I. Jordan, “Local linear perceptrons for classification,” IEEE Transactions on Neural Networks, vol. 7, no. 3, pp. 788–792, 1996.
[53] F. Roli, G. Giacinto, and G. Vernazza, “Methods for designing multiple classifier systems,” 2nd Int. Workshop on Multiple Classifier Systems, in Lecture Notes in Computer Science, J. Kittler and F. Roli, Eds., vol. 2096, pp. 78–87, 2001.

[54] G. Giacinto and F. Roli, “Approach to the automatic design of multiple classifier systems,” Pattern Recognition Letters, vol. 22, no. 1, pp. 25–33, 2001.
[55] L. Breiman, “Bagging predictors,” Machine Learning, vol. 24, no. 2, pp. 123–140, 1996.
[56] Y. Freund and R.E. Schapire, “Decision-theoretic generalization of on-line learning and an application to boosting,” Journal of Computer and System Sciences, vol. 55, no. 1, pp. 119–139, 1997.
[57] F.M. Alkoot and J. Kittler, “Experimental evaluation of expert fusion strategies,” Pattern Recognition Letters, vol. 20, no. 11–13, pp. 1361–1369, Nov. 1999.
[58] J. Kittler, M. Hatef, R.P.W. Duin, and J. Mates, “On combining classifiers,” IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 20, no. 3, pp. 226–239, 1998.
[59] M. Muhlbaier, A. Topalis, and R. Polikar, “Ensemble confidence estimates posterior probability,” 6th Int. Workshop on Multiple Classifier Sys., in Lecture Notes in Comp. Science, N. Oza, R. Polikar, J. Kittler, and F. Roli, Eds., vol. 3541, pp. 326–335, 2005.
[60] J. Grim, J. Kittler, P. Pudil, and P. Somol, “Information analysis of multiple classifier fusion,” 2nd Int. Workshop on Multiple Classifier Systems, in Lecture Notes in Computer Science, J. Kittler and F. Roli, Eds., vol. 2096, pp. 168–177, 2001.
[61] L.I. Kuncheva, “A theoretical study on six classifier fusion strategies,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no. 2, pp. 281–286, 2002.
[62] S.B. Cho and J.H. Kim, “Multiple network fusion using fuzzy logic,” IEEE Transactions on Neural Networks, vol. 6, no. 2, pp. 497–501, 1995.
[63] M. Grabisch and J.M. Nicolas, “Classification by fuzzy integral: performance and tests,” Fuzzy Sets and Systems, vol. 65, no. 2–3, pp. 255–271, 1994.
[64] Y. Lu, “Knowledge integration in a multiple classifier system,” Applied Intelligence, vol. 6, no. 2, pp. 75–86, 1996.
[65] T.K. Ho, “Random subspace method for constructing decision forests,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 20, no. 8, pp. 832–844, 1998.
[66] D.H. Wolpert, “Stacked generalization“, Neural Networks, vol. 5, pp. 241-259, 1992.
[67] R. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, “ Adaptive mixtures of local experts“, Neural Computation, vol. 3, pp. 79-87, 1991.
[68] G. Brown, “Diversity in neural network ensembles,” Ph.D. dissertation, University of Birmingham, 2004.
[69] B. Rosen , “Ensemble learning using decorrelated neural networks,” Connection Science, vol. 8, no. 3, pp. 373–384, 1996.
[70] L. Breiman, “Random Forests,” Machine Learning, vol. 45, no. 1, pp. 5–32, Oct. 2001.
[71] L. Breiman, “Pasting small votes for classifcation in large databases and on-line,” Machine Learning, vol. 36, pp. 85–103, 1999.
[72] P.J. Boland, “Majority system and the Condorcet jury theorem,” Statistician, vol. 38, no. 3, pp. 181–189, 1989.
[73] L. Shapley and B. Grofman, “Optimizing group judgmental accuracy in the presence of interdependencies,” Public Choice (Historical Archive), vol. 43, no. 3, pp. 329–343, 1984.
[74] L.I. Kuncheva, Combining Pattern Classifiers, Methods and Algorithms. New York, NY: Wiley Interscience, 2005.
[75] N. Littlestone and M.Warmuth, “Weighted majority algorithm,” Information and Computation, vol. 108, pp. 212–261, 1994.
[76] Y.S. Huang and C.Y. Suen, “Behavior-knowledge space method for combination of multiple classifiers,” Proc. of IEEE Computer Vision and Pattern Recog., pp. 347–352, 1993.
[77] Y.S. Huang and C.Y. Suen, “A method of combining multiple experts for the recognition of unconstrained handwritten numerals,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 17, no. 1, pp. 90–94, 1995.
[78] R. Struharik, “Digitalna elektronska kola za realizaciju stabala odluka”, doktorska disertacija, FTN, 2009.
[79] Levi D., HereBoy: A Fast Evolutionary Algorithm, The Second Nasa/DoD Workshop on Evolvable hardware EH’00, 2000, pp. 17-24
[80] A. Asuncion, D.J. Newman, UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science, 2007 http://www.ics.uci.edu/~mlearn/MLRepository.html

УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
Трг Доситеја Обрадовића 6, 21000 Нови Сад, Република Србија
Деканат: 021 6350-413; 021 450-810; Централа: 021 485 2000
Рачуноводство: 021 458-220; Студентска служба: 021 6350-763
Телефакс: 021 458-133; e-mail: ftndean@uns.ac.rs
ИНТЕГРИСАНИ СИСТЕМ МЕНАЏМЕНТА СЕРТИФИКОВАН ОД:

Наш број:
Ваш број:
Датум: 2012-12-07

ИЗВОД ИЗ ЗАПИСНИКА

Наставно-научног већа Факултета техничких наука у Новом Саду, на 2. редовној седници одржаној дана 28.11.2012. године, донело је следећу одлуку:

-непотребно изостављено-

Тачка 14.2.4. Питања научноистраживачког рада и међународне сарадње / верификација нових техничких решења

У циљу доношења одлуке о прихватању ***техничког решења –под називом:***

ИП ЈЕЗГРА ЗА ХАРДВЕРСКО ГЕНЕРИСАЊЕ АНАСАМБАЛА СТАБАЛА ОДЛУКА

именују се рецензенти:

- Проф. др Драган Васиљевић, Електротехнички факултет у Београду
- Проф. др Теуфик Токић. Електронски факултет у Нишу

Аутори техничког решења: доцент др Растислав Струхарик, проф. др Ладислав Новак.

-непотребно изостављено-

Записник водила:
Јасмина Димић, дипл. правник

Тачност података оверава:
Секретар
Иван Нешковић, дипл. правник

Декан
Проф. др Раде Дорословачки

Softver:
IP jezgra za hardversko generisanje ansambala stabala odluka

Rukovodilac projekta: prof. dr Ljiljana Živanov
Odgovorno lice: dr Rastislav Struharik
Autori: Rastislav Struharik, Ladislav Novak
Fakultet tehničkih nauka (FTN), Novi Sad

Razvijeno: u okviru projekta tehnološkog razvoja TR-32016
Godina: 2011 - 2012.
Primena: jun 2012.

Kratak opis

Hardverske implementacije stabala odluka mogu predstavljati jedino rešenje u slučajevima kada je potrebno izvršiti klasifikaciju instance u vrlo kratkom vremenu, ili kada je potrebno adaptivno formiranje prediktivnog modela, u toku rada sistema. Ovo su tipični zahtevi koji se sreću prilikom projektovanja savremenih a pogotovo budućih *embedded* sistema. Imajući u vidu ove činjenice, razvijena su IP jezga za hardversko generisanje ansambala stabala odluka koja u mnogome olakšavaju integraciju i rad sa stablima odluka prilikom projektovanja *embedded* sistema.

Tehničke karakteristike:
IP jezgra za hardversko generisanje ansambala stabala odluka modelovana su pomoću standardnog jezika za specifikaciju hardvera, VHDL. Razvijeni modeli su parametrizovani što omogućava jednostavno prilagođavanje arhitekture trenutnim potrebama korisnika.

Tehničke mogućnosti:
Korišćenjem konfiguracionih parametara definisanih unutar VHDL modela arhitektura za hardversko generisanje ansambala stabala odluka (broja stabala u ansamblu, broj bitova koji se koristi za predstavu vednosti atributa problema, koeficijenata razdvajajućih hiperpovrši, ciljnih klasa; broja atributa preko kojih je opisan klasifikacioni problem; maksimalnog broja čvorova u realizovanom stablu odluke, itd.) moguće je prilagoditi VHDL model potrebama tekuće aplikacije. Pilikom sinteze hardvera ovi parametri se koriste kako bi se automatski generisala optimalna hardverska implementacija za tekuću aplikaciju.

Realizator:
Fakultet tehničkih nauka – FTN

Korisnik:
Fakultet tehničkih nauka – FTN, Novi Sad

Podtip rešenja:
Softver – M85

Mišljenje

Tehničko rešenje "IP jezgra za hardversko generisanje ansambala stabala odluka" autora doc. dr Rastislava Struharika, i prof. dr Ladislava Novaka, realizovano 2011.-2012. godine, prikazano je na 34 stranice A4 formata, grupisano je u ukupno pet poglavlja:

1. Opis problema koji se rešava tehničkirn rešenjem,
2. Stanje rešenosti problema u svetu - prikaz i analiza postojećih rešenja,
3. Suština tehničkog rešenja (uključujući i prateće ilustracije i tehničke crteze),
4. Detaljan opis primene tehničkog rešenja i
5. Literatura.

Tehničko rešenje pripada polju tehničko-tehnoloških nauka i oblasti elektrotehničkog inženjerstva. Naručilac tehničkog rešenja je Fakultet tehnilkih nauka u Novom Sadu, Republika Srbija, koji je i korisnik tehničkog rešenja.

Tehničko rešenje je realizovano u okviru projekta "Nove generacije ugrađenih elektronskih komponenti i sistema u neorganskim i organskim tehnologijama za uređaje široke potrošnje" (Broj projekta TR 32016, Program istraživanja u oblasti tehnološkog razvoja za period 2011-2014., Tehnološka oblast - Elektronika, telekomunikacije i informacione tehnologije, Rukovodilac projekta: dr Ljiljana Živanov, redovni profesor).

Na osnovu analize tehničkog rešenja "IP jezgra za hardversko generisanje ansambala stabala odluka" autora doc. dr Rastislava Struharika, i prof. dr Ladislava Novaka, mogu se izvesti sledeći zaključci:

1. Dokumentacija tehničkog rešenja jasno prikazuje kompletnu strukturu tehničkog rešenja – opis problema, daje dataljniji osvrt na stanje u svetu, sadrži odgovarajući prikaz teorijskih osnova na kojima je zasnovano tahničko rešenje i posebno detaljno prikazuje strukturu i primenu realizovanog tehničkog rešenja.
2. Predloženo tehničko rešenje, "IP jezgra za hardversko generisanje ansambala stabala odluka", predstavlja efikasan alat za rešavanje problema u oblasti hardverske implementacije jedne grupe algoritama mašinskog učenja, ansambala stabala odluka.
3. Tehničko rešenje karakteriše originalan naučni doprinos koji ima izraženu praktičnu dimenziju budući da se kroz korišćenje niza konfiguracionih parametara omogućava njegova fleksibilna i univerzalnija primena.

Na osnovu prethodnog, predlažem da se "IP jezgra za hardversko generisanje ansambala stabala odluka", autora doc. Dr Rastislava Struharika, i prof. Dr Ladislava Novaka, prihvati kao novo tehničko rešenje i u skladu sa Pravilnikom o postupku i načinu vrednovanja, i kvantitativnom iskazivanju naučnoistraživačkih rezultata istraživača ("Službeni glasnik RS", broj 38/2008) klasifikuje kao rezultat "M85 Prototip, nova metoda, softver, standardizovan ili atestiran instrument, nova genska proba, mikroorganizmi".

U Nišu, 17.12.2012. god.

Prof. dr Teufik Tokić,
Univerzitet u Nišu
Elektronski fakultet

Softver:
IP jezgra za hardversko generisanje ansambala stabala odluka

Rukovodilac projekta: prof. dr Ljiljana Živanov
Odgovorno lice: dr Rastislav Struharik
Autori: Rastislav Struharik, Ladislav Novak
Fakultet tehničkih nauka (FTN), Novi Sad

Razvijeno: u okviru projekta tehnološkog razvoja TR-32016
Godina: 2011 - 2012.
Primena: jun 2012.

Kratak opis

Hardverske implementacije stabala odluka mogu predstavljati jedino rešenje u slučajevima kada je potrebno izvršiti klasifikaciju instance u vrlo kratkom vremenu, ili kada je potrebno adaptivno formiranje prediktivnog modela, u toku rada sistema. Ovo su tipični zahtevi koji se sreću prilikom projektovanja savremenih a pogotovo budućih *embedded* sistema. Imajući u vidu ove činjenice, razvijena su IP jezga za hardversko generisanje ansambala stabala odluka koja u mnogome olakšavaju integraciju i rada sa stablima odluka prilikom projektovanja *embedded* sistema.

Tehničke karakteristike:
IP jezgra za hardversko generisanje ansambala stabala odluka modelovana su pomoću standardnog jezika za specifikaciju hardvera, VHDL. Razvijeni modeli su parametrizovani što omogućava jednostavno prilagođavanje arhitekture trenutnim potrebama korisnika.

Tehničke mogućnosti:
Korišćenjem konfiguracionih parametara definisanih unutar VHDL modela arhitektura za hardversko generisanje ansambala stabala odluka (broja stabala u ansamblu, broj bitova koji se koristi za predstavu vednosti atributa problema, koeficijenata razdvajajućih hiperpovrši, ciljnih klasa; broja atributa preko kojih je opisan klasifikacioni problem; maksimalnog broja čvorova u realizovanom stablu odluke, itd.) moguće je prilagoditi VHDL model potrebama tekuće aplikacije. Pilikom sinteze hardvera ovi parametri se koriste kako bi se automatski generisala optimalna hardverska implementacija za tekuću aplikaciju.

Realizator:
Fakultet tehničkih nauka – FTN

Korisnik:
Fakultet tehničkih nauka – FTN, Novi Sad

Podtip rešenja:
Softver – M85

Mišljenje

Fakultet tehničkih nauka je razvio IP jezgra za hardversko generisanje ansambala stabala odluka. IP jezgra su opisana korišćenjem VHDL jezika za modelovanje hardvera. Razvijeni modeli jezgara se vrlo lako mogu prilagoditi trenutnim potrebama aplikacije korišćenjem konfiguracionih parametara. Na ovaj način omogućena je široka upotreba ovih jezgara u velikom broju različitih aplikacija.

U predloženom tehničkom rešenju razmatran je problem hardverskog generisanja ansambala ortogonalnih, neortogonalnih i nelinearnih stabala odluka. Analizom postojećih rešenja utvrđeno je da u dostupnoj literaturi ne postoje radovi koji se bave problematikom hardverskog generisanja ansambala stabala odluka.

Nakon uvodne glave, u kojoj su u najkraćim crtama evedeni pojmovi stabala odluka i ansambala prediktvnih modela zajedno sa najznačajnijim algoritmima za njihovo kreiranje, predložena je arhitektura za hardversko generisanje ansambala ortogonalnih, neortogonalnih i nelinearnih stabala odluka bazirane na poznatom Bagging algoritmu. H_DTE arhitektura predstavlja paralelnu implementaciju originalnog Bagging algoritma koja koristi DTS algoritam za formiranje individualnih stabala iz ansambla.

U zavisnosti od toga na koji način se realizuje modul za određivanje pozicije trening instanci u odnosu na hiperpovrš u svakom od čvorova stabla odluke, predložene su dve varijante H_DTE arhitekture:

1. H_DTE1 – arhitektura koja realizuje DTE algoritam za formiranje ansambla stabala odluka, pri čemu se pozicija instance u odnosu na hiperpovrš računa serijski
2. H_DTE2 – arhitektura koja realizuje DTE algoritam za formiranje ansambla stabala odluka, pri čemu se pozicija instance u odnosu na hiperpovrš računa paralelno

Predložene arhitekture upoređene su u terminima potrebnih hardverskih resursa za njihovu implementaciju pomoću FPGA komponenti kao i u terminima performansi izraženih u vremenu potrebnom za formiranje stabla odluke.

Nakon sprovedene teorijske analize izvršena je i ilustracija formiranja ansambala stabala odluka korišćenjem niza UCI problema. Korišćenjem ovih konkretnih problema, na eksperimentalan način potvrđene su teorijske estimacije potrebnih resursa kao i mogućih performansi.

U skladu sa gore iznetim činjenicama tehničko rešenje ispunjava uslove da bude priznato kao softver (odnosno M85 u skladu sa Oravilnikom o postupku i načinu vrendovanja i kvantitativnom iskazivanju naučnoistraživačkih rezultata istraživača, Sl. gl. RS br. 38/08).

Dr Dragan Vasiljević, dipl. el. inž.

Redovni profesor Elektrotehničkog fakulteta, Beograd

Dragan M. Vasiljević

УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА

Трг Доситеја Обрадовића 6, 21000 Нови Сад, Република Србија
Деканат: 021 6350-413; 021 450-810; Централа: 021 485 2000
Рачуноводство: 021 458-220; Студентска служба: 021 6350-763
Телефакс: 021 458-133; e-mail: ftndean@uns.ac.rs

ИНТЕГРИСАНИ СИСТЕМ МЕНАЏМЕНТА СЕРТИФИКОВАН ОД:

Наш број: 01.сл
Ваш број:
Датум: 2012-12-28

ИЗВОД ИЗ ЗАПИСНИКА

Наставно-научног већа Факултета техничких наука у Новом Саду, на 3. редовној седници одржаној дана 26.12.2012. године, донело је следећу одлуку:

-непотребно изостављено-

Тачка 14.1.8. Питања научноистраживачког рада и међународне сарадње / верификација нових техничких решења

Одлука

На основу позитивног извештаја рецензената прихвата се
техничко решење – (М85) под називом:

ИП ЈЕЗГРА ЗА ХАРДВЕРСКО ГЕНЕРИСАЊЕ АНАСАМБАЛА СТАБАЛА ОДЛУКА

Аутори техничког решења: доцент др Растислав Струхарик, проф. др Ладислав Новак.

-непотребно изостављено-

Записник водила:
Јасмина Димић, дипл. правник

Тачност података оверава:
Секретар
Иван Нешковић, дипл. правник

Декан
проф. др Раде Дорословачки