TEHNIČKO REŠENJE

IP jezgra za hardversko generisanje 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 stabala odluka koja u mnogome olakšavaju integraciju i rada sa stablima odluka prilikom projektovanja *embedded* sistema.

Tehničke karakteristike:

IP jezgra za hardversko generisanje 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 stabala odluka (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

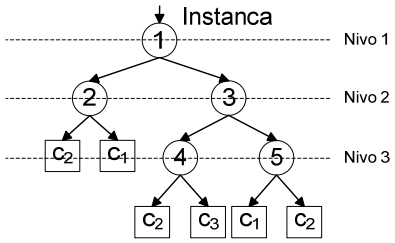
Projekat u okviru koga je realizovano tehničko rešenje:

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*)

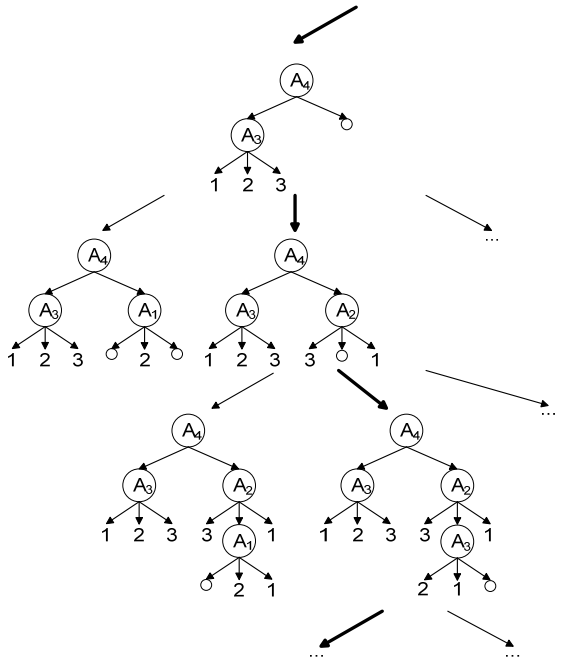
Ulazi:	
trening_skup	*Skup trening instanci. Svaka od instanci je predstavljena pomoću niza vrednosti atributa,* ***d****, praćenog brojem klase kojoj ta instanca pripada*
atributi	*Lista atributa pomoću kojih je opisan problem. Lista ima ukupno* ***d*** atributa.
Izlazi:	
koren	*Koren formiranog neortogonalnog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

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

3. IP jezgra za hardversko generisanje stabala odluka

3.1 Evolutivni dizajn stabala odluka

Glavni nedostatak stabala odluka u odnosu na druge prediktivne modele jeste da je njihova tačnost uglavnom manja od tačnosti koja se može postići ako se isti problem rešava korišćenjem drugih prediktivnih modela (veštačkih neuronskih mreža ili *support vector* mašina). Ukoliko se detaljnije analizira način particije prostora pomoću neuronskih mreža ili *support vector* mašina, može se doći do zaključka da ovi prediktivni modeli prilikom particije hiperprostora definisanog atributima problema koji se rešava koriste neortogonalne hiperravni, ili još preciznije, nelinearne hiperpovrši. Upravo zbog toga ovi prediktivni modeli ostvaruju bolje rezultate u poređenju sa stablima odluka. Stoga je i prirodno razmotriti načine na koji se mogu modifikovati klasična stabla odluke tako da i ona prilikom particije prostora problema koriste neortogonalne hiperravni ili čak i nelinearne hiperpovrši. Stabla odluka sa neortogonalnim hiperravnima ili nelinearnim hiperpovršima bi potencijalno mogla imati veću tačnost u odnosu na klasična stabla odluka koja koriste ortogonalne hiperravni.

Sa druge strane, kod većine praktičnih problema klasifikacije, krajnje je neefikasno vršiti particiju hiperprostora pomoću ortogonalnih hiperravni. Razdvajajuće površi između instanci koje pripadaju različitim klasama najčešće su nelinearne ili u najmanju ruku mogu da se aproksimiraju pomoću neortogonalnih hiperravni. Ideja o korišćenju neortogonalnih hiperravni odnosno nelinearnih hiperpovrši umesto ortogonalnih hiperravni kao testova u svakom od čvorova stabla odluke trebalo bi da rezultuje u znatno kompaktnijim stablima odluka (sa manjim brojem čvorova).

U dostupnoj literaturi postoji čitav niz algoritama za formiranje neortogonalnih stabala odluka i manji broj algoritama za formiranje nelinearnih stabala odluka. Zajedničko svim ovim algoritmima je da su bazirani na heurističkim algoritmima, jer je poznato da je problem formiranja optimalnog neortogonalnog ili nelinearnog stabla odluke NP težak.

3.1.1 Korišćena *impurity* funkcija

Prilikom određivanja optimalnog položaja hiperpovrši u svakom čvoru stabla odluke potrebno je koristiti neku funkciju koja će meriti trenutni kvalitet položaja. Ta funkcija se zove *impurity* funkcija.

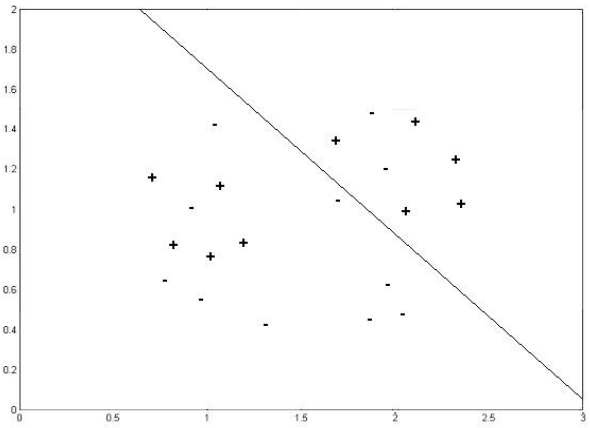
Na početku ćemo definisati *impurity* funkciju za probleme klasifikacije u samo dve klase, a zatim će biti predstavljena generalizovana forma koja se koristi kada se vrši klasifikacija u više od dve klase.

$$impurity = \frac{1}{N}\left(N_{1+} \cdot ld\frac{N_{1+}}{N_{1+}+N_{1-}} + N_{1-} \cdot ld\frac{N_{1-}}{N_{1+}+N_{1-}} + \left(N_{+}-N_{1+}\right) \cdot ld\frac{N_{+}-N_{1+}}{N-N_{1+}-N_{1-}} + \left(N_{-}-N_{1-}\right) \cdot ld\frac{N_{-}-N_{1-}}{N-N_{1+}-N_{1-}} \right) \tag{3.1}$$

Značenje pojedinih simbola u formuli je:

N	- ukupan broj trening instanci asociranih tekućem čvoru
N_{+}	- ukupan broj trening instanci koje pripadaju klasi 1 (od ukupno N instanci)
N_{-}	- ukupan broj trening instanci koje pripadaju klasi 0 (od ukupno N instanci)
N_{1+}	- ukupan broj instanci koje pripadaju klasi 1 i nalaze se iznad hiperpovrši
N_{1-}	- ukupan broj instanci koje pripadaju klasi 0 i nalaze se iznad hiperpovrši

Na primer, pretpostavimo da imamo problem klasifikacije u dve klase koji je opisan pomoću dva atributa. Neka su položaj trening instanci asociranih tekućem čvoru i trenutni položaj hiperpovrši prikazani na sledećoj slici. Kao što se sa slike može videti, pretpostavka je da koristimo neortogonalne hiperravni.



Slika 3.1 Podela trening instanci pomoću hiperravni

Pretpostavljajući da je trenutni položaj instanci kao na slici 3.1, vrednosti pojedinih simbola iz jednačine (3.1) su:

N	=	21
N_+	=	10
N_-	=	11
N_{1+}	=	5
N_{1-}	=	2

U ovom slučaju vrednost *impurity* funkcije jednaka je

$$impurity = 0.915$$

Impurity funkcija definisana na ovakav način ima sledeća svojstva, ima vrednost 0 kada hiperpovrš potpuno razdvaja instance jedne klase od instanci druge klase. Kako se kvalitet podele definisane položajem hiperpovrši smanjuje, tako i vrednost *impurity* funkcije raste od 0 ka 1.

Ukoliko je neophodno rešavati probleme koji imaju više od dve klase potrebno je koristiti sledeću, generalnu formu *impurity* funkcije.

$$impurity = -\frac{1}{N}\left(\sum_{i=1}^{k} N_{1i} \cdot ld \frac{N_{1i}}{\sum_{j=1}^{k} N_{1j}} + \sum_{i=1}^{k} (N_i - N_{1i}) \cdot ld \frac{N_i - N_{1i}}{N - \sum_{j=1}^{k} N_{1j}}\right) \tag{3.2}$$

Značenje pojedinih simbola iz gornjeg izraza je sledeće:

N	- ukupan broj trening instanci asociranih tekućem čvoru
N_i	- ukupan broj trening instanci koje pripadaju klasi *i* (od ukupno N instanci)
N_{1i}	- ukupan broj instanci koje pripadaju klasi *i*, a nalaze se iznad hiperpovrši

U narednim odeljcima biće predstavljeni algoritmi za formiranje neortogonalnih i nelinearnih stabala odluka kao i algoritam za kresanje formiranih stabala odluka sa ciljem poboljšanja generalizacije.. Kod neortogonalnih stabala odluka kao testovi u čvorovima koriste se linearne kombinacije atributa koji opisuju problem (2.2), što odgovara particiji prostora neortogonalnim hiperravnima. Kod nelinearnih stabala odluka testovi su još složeniji i odgovaraju polinomima drugog i trećeg reda, (2.4).

Za optimizaciju položaja hiperpovrši unutar svakog čvora koristiće se *HereBoy* algoritam [35]. Kao mera kvaliteta trenutnog položaja hiperpovrši biće korišćena *impurity* funkcija definisana u ovom odeljku.

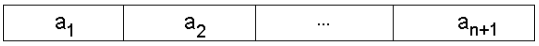
HereBoy, kao i većina evolutivnih algoritama, tokom svog rada traži rešenja problema kojima odgovara velika fitnes vrednost. *Impurity* funkcije (3.1) i (3.2) sa druge strane su definisane na takav način da je njihova vrednost minimalna kada je kvalitet podele tekućom hiperpovrši najbolji, tako da ne mogu direktno biti iskorišćene kao fitnes funkcije. Zbog toga je fitnes funkcija definisana sledećim izrazom

$$fitnes = 1 - impurity \tag{3.3}$$

HereBoy za reprezentaciju problema koristi nizove binarnih simbola. Obzirom da *HereBoy* treba da pronađe optimalni položaj hiperpovrši na osnovu skupa trening instanci koje su pridružene tekućem čvoru, problem optimizacije se svodi na nalaženje optimalnih vrednosti koeficijenata koje definišu položaj hiperpovrši. Ovaj skup koeficijenata biće predstavljen pomoću niza binarnih simbola, pri čemu će se taj niz

sastojati od odgovarajućeg broja podnizova, iste dužine *l*, od kojih će svaki predstavljati vrednost jednog koeficijenta. Pomoću parametra *l*, moguće je kontrolisati tačnost sa kojom se određuju vrednosti pojedinih koeficijenata.

Na primer ako se želi pronaći optimalan položaj hiperravni definisane jednačinom (2.2), izgled hromozoma biće kao na sledećoj slici.



Slika 3.2 Struktura hromozoma

Svaki od podnizova kodira trenutnu vrednost odgovarajućeg koeficijenta a_i iz jednačine (2.2).

3.1.2 Algoritam za formiranje neortogonalnih stabala odluka

Algoritam za formiranje neortogonalnih stabala odluka pri čemu se za optimizaciju položaja hiperravni u svakom od čvorova stabla koristi *HereBoy* algoritam, ima sledeći izgled.

koren = DTL (*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
  - Odredi dominantnu klasu unutar skupa *trening_skup* i numeriši tekući čvor sa brojem te klase
  - Koristeći *HereBoy* algoritam i fitnes funkciju (3.3), pronađi optimalnu poziciju hiperravni unutar hiperprostora definisanog atributima problema, odnosno pronađi optimalne vrednosti koeficijenata iz jednačine (2.2) za koje će *impurity* funkcija (3.2) 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, $trening\_skup_{ispod}$.
  - Kreiraj levu granu koja polazi od čvora *koren*, i na tom mestu kreiraj podstablo pozivajući *koren->levo* = DTL ($trening\_skup_{iznad}$)
  - Kreiraj desnu granu koja polazi od čvora *koren*, i na tom mestu kreiraj podstablo pozivajući *koren->desno* = DTL ($trening\_skup_{ispod}$)
- Vrati *koren*

Algoritam 3.1 Algoritam za formiranje neortogonalnog stabla odluke koji za nalaženje optimalne pozicije hiperravni u svakom čvoru koristi HereBoy *algoritam*

3.1.3 Algoritam za formiranje nelinearnih stabala odluka

Algoritam za formiranje nelinearnih stabala odluka pri čemu se za optimizaciju položaja hiperpovrši u svakom od čvorova stabla koristi *HereBoy* algoritam, ima sledeći izgled.

koren = DTN (*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 nelinearnog 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

- Odredi dominantnu klasu unutar skupa *trening_skup* i numeriši tekući čvor sa brojem te klase
- Koristeći *HereBoy* algoritam i fitnes funkciju (3.3), pronađi optimalnu poziciju hiperpovrši unutar hiperprostora definisanog atributima problema, odnosno pronađi optimalne vrednosti koeficijenata iz jednačine (2.4) za koje će *impurity* funkcija (3.2) imati minimalnu vrednost.
- Koristeći optimalnu hiperpovrš, podeli skup *trening_skup* na dva podskupa, jedan u kome će se nalaziti sve instance locirane iznad optimalne hiperpovrši, $trening\_skup_{iznad}$, i drugi, u kojem će se nalaziti sve instance koje su locirane ispod optimalne hiperpovrši, $trening\_skup_{ispod}$.
- Kreiraj levu granu koja polazi od čvora *koren*, i na tom mestu kreiraj podstablo pozivajući
  koren->levo = DTN ($trening\_skup_{iznad}$)
- Kreiraj desnu granu koja polazi od čvora *koren*, i na tom mestu kreiraj podstablo pozivajući
  koren->desno = DTN ($trening\_skup_{ispod}$)

- Vrati *koren*

Algoritam 3.2 Algoritam za formiranje nelinearnog stabla odluke koji za nalaženje optimalne pozicije hiperpovrši u svakom čvoru koristi HereBoy *algoritam*

3.1.4 Problem inicijalizacije i njegovo rešenje

Prilikom rada sa *HereBoy* algoritmom kao prvi korak neophodno je izvršiti inicijalizaciju hromozoma. Ova inicijalizacija se uglavnom vrši na slučajan način. Međutim u ovom slučaju takav način inicijalizacije ne može garantovati da će početni položaj hiperpovrši biti takav da će se barem neke od instanci trening skupa naći i sa jedne i sa druge strane hiperpovrši. Ako to nije slučaj onda će vrednost fitnes funkcije biti jednaka 0 i ostaće nula sve dok se ne naiđe na takav položaj hiperpovrši kod kojega se instance nalaze i sa jedne i sa druge strane. Obzirom na to, sve do tog trenutka *HereBoy* algoritam će se praktično svesti na slučajnu pretragu prostora rešenja koja je vrlo neefikasna i svakako je treba izbeći. Upravo zbog toga je razvijen postupak slučajne inicijalizacije hromozoma koji će pored toga što je slučajan ipak voditi računa da instance budu locirane i sa jedne i sa druge strane površi.

Posmatrajmo situaciju kada se u čvorovima koriste linearni testovi, kojima odgovaraju neortogonalne hiperravni definisane jednačinom (2.2). Ako problem koji rešavamo ima ukupno n atributa pomoću kojih je opisan, tada će se hromozom sastojati od ukupno $n+1$ koeficijenata. Prilikom inicijalizacije, n od $n+1$ koeficijenata biće inicijalizovano na slučajan način, a vrednost preostalog koeficijenta biće odabrana na takav način da se obezbedi da hiperpovrš prolazi kroz središte trening instanci pridruženih posmatranom čvoru:

$$centar_i = -\frac{1}{N_{train}} \sum_{j=1}^{N_{train}} A_{ij}, \quad i = 1..n \tag{3.4}$$

N_{train} označava ukupan broj trening instanci asociranih tekućem čvoru stabla, a n predstavlja broj atributa pomoću kojih je opisan klasifikacioni problem. A_{ij} predstavlja vrednost i-tog atributa j-te trening instance. Algoritam za slučajnu inicijalizaciju pozicije hiperravni koji vodi računa da se i sa jedne i sa druge strane hiperravni nađe barem jedna instanca iz trening skupa ima sledeći izgled.

hiperravan = Inicijalizuj_hiperravan (*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:	
hiperravan	*Koeficijenti hiperravan*

- Koristeći instance iz skupa *trening_skup* i jednačinu (3.4) odredi vrednost središta trening instanci, $centar_i$, $i=1, ..., n$.
- Na slučajan način inicijalizuj n koeficijenata hiperravni, $a_1, a_2, ..., a_n$.
- Izračunaj vrednost $n+1$-vog koeficijenta korišćenjem sledeće formule

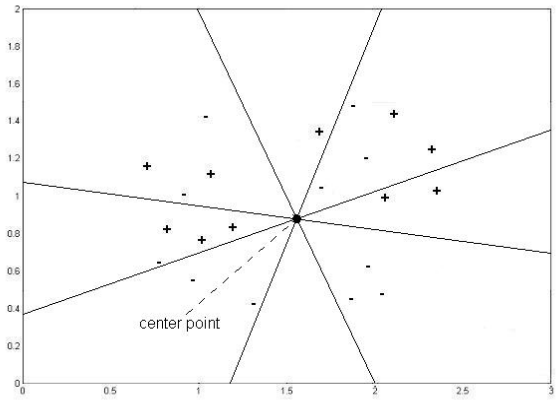
$$a_{n+1} = -\sum_{i=1}^{n} centar_i \cdot a_i \tag{3.5}$$

- *hiperravan*=$(a_1, a_2, ..., a_n, a_{n+1})$.
- Vrati *hiperravan*

Algoritam 3.3 Algoritam za slučajnu inicijalizaciju hiperravni

Algoritam za slučajnu inicijalizaciju hiperpovrši je praktično identičan sa algoritmom 3.3 te stoga neće biti prikazan.

Na sledećoj slici ilustrovan je položaj središta, kao i moguće inicijalne pozicije hiperpovrši koje su prihvatljive i mogle bi biti rezultat izvršavanja algoritma 3.3. Slika prikazuje slučaj kada se za particiju prostora koriste neortogonalne hiperravni.



Slika 3.3 Moguće inicijalne pozicije hiperravni

3.1.6 Vremenska kompleksnost predloženih algoritama

Vremenska kompleksnost algoritama za formiranje neortogonalnih i nelinearnih stabala odluke zavisi od tri parametra: veličine trening skupa *m*, broja atributa klasifikacionog problema *n* i broja generacija *HereBoy* algoritma *gen*.

Vremenska kompleksnost predloženih algoritama takođe zavisi i od veličine stabla koja će biti formirana. U opštem slučaju ova veličina nije poznata unapred, jer se stablo formira iterativno, tokom samog algoritma. Za potrebe određivanja vremenske kompleksnosti neophodno je odrediti maksimalnu veličinu stabla koje se može formirati korišćenjem zadatog trening skupa veličine *m*. Obzirom da predloženi algoritmi formiraju binarna stabla, najveće moguće stablo koje se može formirati od *m* trening instanci imaće *m-1* čvorova. Stoga će najveće stablo imati *m-1* različitih neortogonalnih ili nelinearnih hiperpovrši.

Da bismo odredili optimalnu poziciju hiperravni ili hiperpovrši u svakom od čvorova stabla, koristimo *HereBoy* algoritam. Vreme potrebno za određivanje optimalne pozicije hiperravni ili hiperpovrši zavisi od broja atributa problema *n*, broja instanci koje su asocirane tekućem čvoru m_i i od vrste hiperpovrši koju koristimo za particiju. Vreme potrebno za izvršavanje evolutivnog algoritma u tekućem čvoru može se predstaviti pomoću sledećeg izraza.

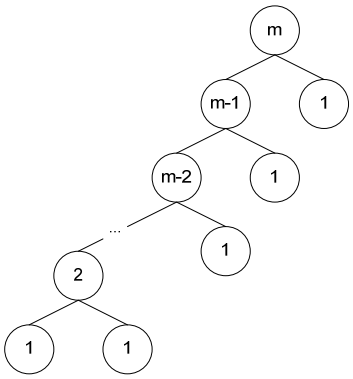
$$T_{hiperpovrš_i} = gen \cdot m_i \cdot f(n) \tag{3.6}$$

Funkcija *f(n)* predstavlja vreme potrebno da se odredi pozicija jedne instance u odnosu na hiperpovrš. Ovo vreme očigledno zavisi od tipa hiperpovrši koja se koristi, odnosno od broja atributa pomoću kojih se opisuje klasifikacioni problem. U nastavku će biti razmotren oblik funkcije *f(n)* u slučaju korišćenja neortogonalnih hiperravni odnosno nelinearnih hiperpovrši.

Obzirom, da stablo odluke u najgorem slučaju može imati *m-1* čvorova, vremenska kompleksnost predloženih algoritama za formiranje neortogonalnih odnosno nelinearnih stabala odluka može se predstaviti sledećim izrazom.

$$T_{stablo} = \sum_{i=1}^{m-1} T_{hiperpovrš_i} = \sum_{i=1}^{m-1} gen \cdot m_i \cdot f(n) \tag{3.7}$$

Vreme formiranja stabla očigledno zavisi od distribucije instanci trening skupa po čvorovima stabla odluke. Najgori slučaj ove distribucije prikazan je na slici 3.4. Na slici se vidi binarno stablo od *m-1* čvorova, pri čemu je u svakom od čvorova i listova naveden broj instanci koji im je asociran.



Slika 3.4 Najgora moguća distribucija trening instanci po čvorovima stabla odluke

Poznavanjem najgore distribucije instanci trening skupa po čvorovima stabla, moguće je odrediti vremensku kompleksnost formiranja neortogonalnog i nelinearnog stabla odluke u najgorem slučaju.

$$T_{stablo} = \sum_{i=1}^{m-1} gen \cdot m_i \cdot f(n) = gen \cdot f(n) \cdot (m-1) \cdot (\frac{m}{2} + 1) \tag{3.8}$$

Jedino što je preostalo uraditi jeste da se proceni vremenska kompleksnost funkcije *f(n)*. Kao što je rečeno, ova funkcija određuje vreme potrebno za određivanje pozicije instance u odnosu na hiperpovrš i ona zavisi od tipa hiperpovrši koja se koristi. U slučaju da se koriste linearne hiperpovrši opisane jednačinom (2.2) za određivanje pozicije instance potrebno je izvršiti *n* operacija množenja i *n* operacija sabiranja. Vremenska kompleksnost određivanja pozicije instance u ovom slučaju iznosi

$$f_{hiperravan}(n) = 2 \cdot n \tag{3.9}$$

U slučaju korišćenja nelinearnih hiperpovrši, preciznije hiperpovrši definisanih izrazima (2.4), funkcija *f(n)* ima sledeće vrednosti

$$\begin{aligned} f_{hiperpovrš\,2}(n) &= \frac{3}{2} \cdot n^2 + \frac{7}{2} \cdot n \\ f_{hiperpovrš\,3}(n) &= \frac{2}{3} \cdot n^3 + \frac{7}{2} \cdot n^2 + \frac{29}{6} \cdot n \end{aligned} \tag{3.10}$$

Poznavanjem tačnih izraza za funkciju *f(n)* u ova tri slučaja, moguće je precizno odrediti vremensku kompleksnost predloženih algoritama za formiranje stabala odluka.

$$\begin{aligned} T_{linearno\_stablo} &= O(gen \cdot m^2 \cdot n) \\ T_{nelinearno\_stablo\_2} &= O(gen \cdot m^2 \cdot n^2) \\ T_{nelinearno\_stablo\_3} &= O(gen \cdot m^2 \cdot n^3) \end{aligned} \tag{3.11}$$

Što se tiče predloženog algoritma za kresanje formiranog stabla odluke, njegovu vremensku kompleksnost je daleko lakše proceniti. Analizom rada algoritma A.8 možemo videti da se prilikom kresanja svaki čvor stabla posećuje tačno jednom. To znači da je vremenska kompleksnost predloženog algoritma za kresanje linearna u terminima broja čvorova formiranog stabla.

3.2 Arhitekture za hardversko generisanje stabala odluka

U ovoj glavi biće predstavljene arhitekture za hardversko generisanje linearnih stabala odluka,. Kao što je pokazano u glavi 2, nelinearna stabla odluka koja koriste hiperpovrši polinomijalnog tipa uvek se mogu prevesti u linearna stabla odluka koja rešavaju posmatrani problem u nekom drugom, većem prostoru atributa, uvođenjem pojma veštačkih atributa. Uzimajući ovu činjenicu u obzir, nema potrebe za razvijanjem posebnih arhitektura za hardversku implementaciju algoritama za formiranje linearnih i nelinearnih stabala. Dovoljno je razmatrati samo problem hardverskog generisanja linearnih stabala odluke. Ukoliko želimo formirati nelinearno stablo odluke, prvo je neophodno transformisati polazni problem uvodeći potreban broj veštački generisanih atributa (koji se formiraju kao proizvodi originalnih atributa problema), a zatim za tako definisani

problem formirati linearno stablo odluke. Isto rezonovanje može se primeniti i u slučaju hardverske implementacije ansambala stabala odluke.

3.2.1 *H_DTS* arhitektura za hardversku implementaciju *DTL* algoritma

Originalni *DTL* algoritam predstavljen ranije, nije pogodan za hardversku implementaciju. Razlog za to je što *DTL* algoritma koristi *depth-first* tehniku prilikom formiranja stabla odluke. Ovakav pristup formiranju stabla odluke zahtevao bi vrlo složeno manipulisanje sa skupovima trening instanci asociranim čvorovima iz stabla odluke na različitim nivoima. Situacija se znatno pojednostavljuje ukoliko bi se stablo odluke formiralo korišćenjem *breath-first* tehike. Stoga je originalni *DTL* algoritam prerađen imajući ovo u vidu i tako je dobijen *DTS* algoritam za formiranje stabla odluke baziran na *breath-first* tehnici.

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 (3.3), pronađi optimalnu poziciju hiperravni unutar hiperprostora definisanog atributima problema, odnosno pronađi optimalne vrednosti koeficijenata iz jednačine (2.2) za koje će *impurity* funkcija (3.2) 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 (3.3), pronađi optimalnu poziciju hiperravni unutar hiperprostora definisanog atributima problema, odnosno pronađi optimalne vrednosti koeficijenata iz jednačine (2.2) za koje će *impurity* funkcija (3.2) 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

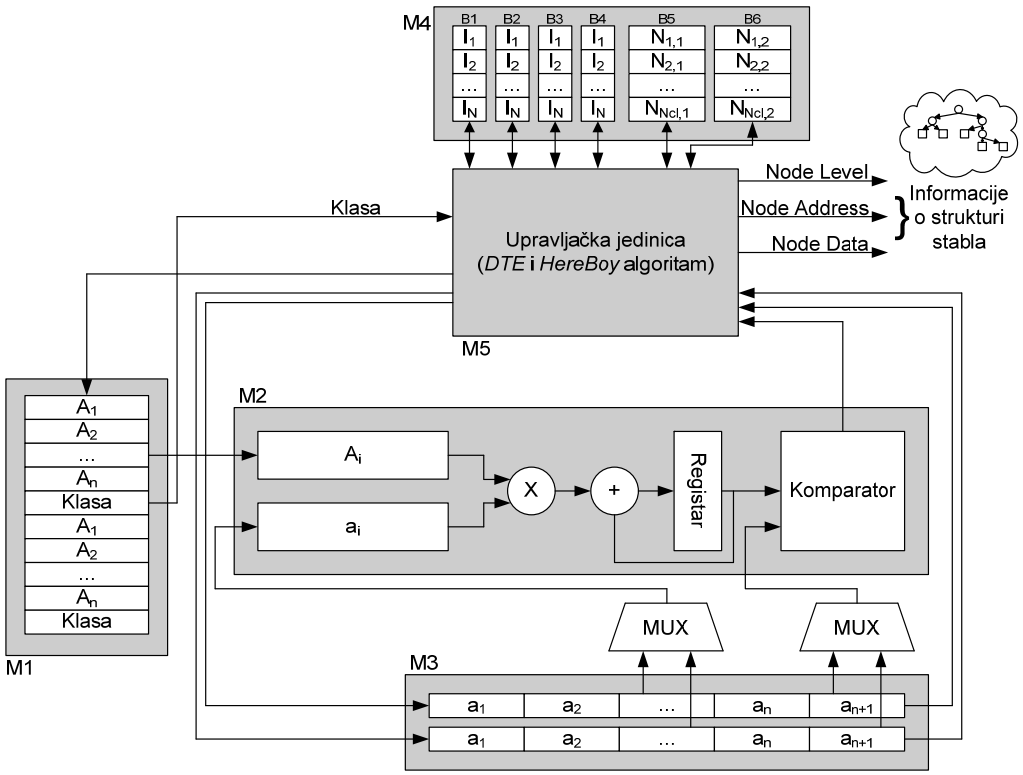
asocirane_instance = nove_asocirane_instance

- Vrati *koren*

Algoritam 3.4 DTS *Algoritam za formiranje neortogonalnog stabla odluke baziran na* breath-first *tehnici*

Kao što je ranije već rečeno, glavna karakteristika *DTS* algoritma za formiranje neortogonalnih stabala odluka jeste da se za razliku od *DTL* algoritma 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.

Na slici 3.5 prikazana je *H_DTS* arhitektura za hardversku implementaciju *DTS* algoritma. Pomoću ove arhitekture moguće je implementirati i *DTN* algoritam u slučaju kada se kao nelinearne hiperpovrši koriste polinomijalne funkcije uz prethodno generisanje veštačkih atributa.



Slika 3.5 H_DTS *arhitektura za hardversku realizaciju* DTS *algoritma*

Kao što se slike 3.5 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*).

3.2.1.1 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.

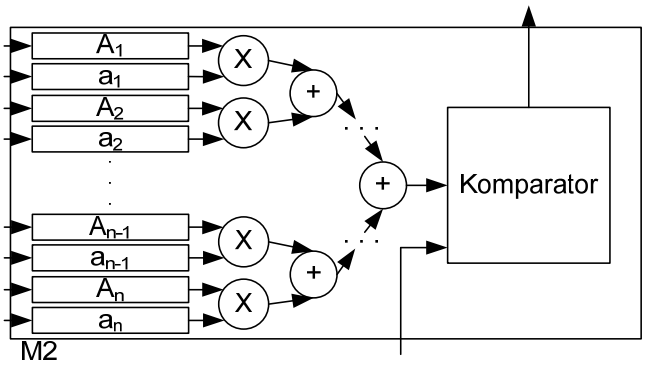
$$RAM\_Size_{M1} = N_{ts} \cdot (n+1) \tag{3.12}$$

U izrazu (3.12), 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.

4.2.1.2 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.

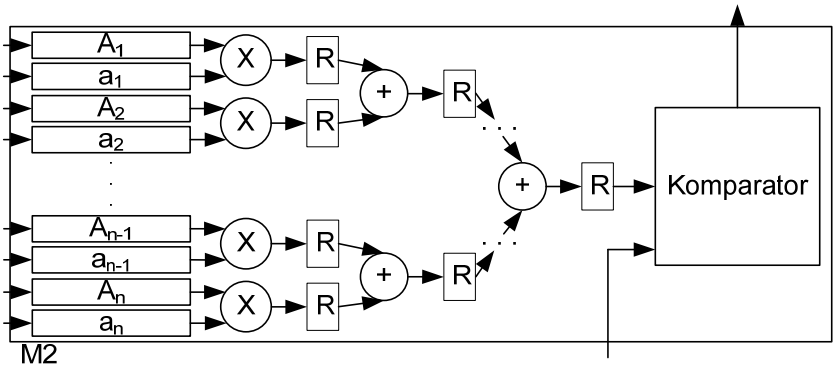
Korišćenjem ovakvog modula za evaluaciju instanci, određivanje položaja instance u odnosu na hiperravan traje ukupno *n+1* taktova. Ovo je posledica serijske evaluacije jednačine (2.2). Sa druge strane, evaluacija pozicija instanci predstavlja kritični korak u čitavom *DTS* algoritmu za formiranje neortogonalnog 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 3.6 prikazana je nova arhitektura modula *M2* koja omogućuje paralelnu evaluaciju jednačine (2.2).



Slika 3.6 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 3.7 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.

3.2.1.3 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.

$$RAM\_Size_{M3} = 2 \cdot (n+1) \tag{3.13}$$

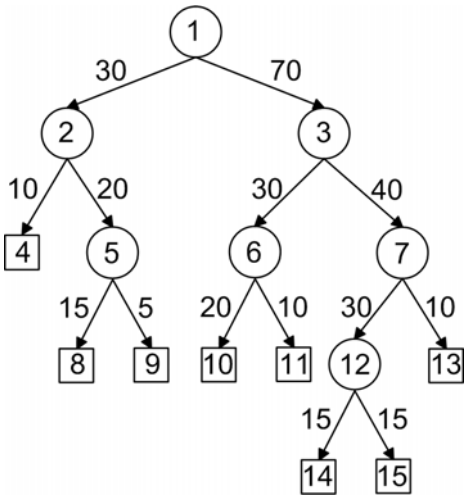
Za kodiranje pozicije svake od dve hiperravni potrebno je kodirati vrednosti *n+1* koeficijenata, pri čemu se svaki koeficijent kodira pomoću N_c bita.

3.2.1.4 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

$$RAM\_Size_{M4\_B1-B4} = N_{ts} \tag{3.14}$$

Za predstavljanje vrednosti indeksa potrebno je $\lceil ld(N_{ts}) \rceil$ bita. Smisao i način korišćenja memorija za čuvanje asociranih indeksa instanci najjednostavnije će se razumeti na ilustrativnom primeru. Slika 3.8 prikazuje strukturu jednog stabla odluke koje je moglo biti formirano korišćenjem *DTS* algoritma. Krugovi na slici označavaju čvorove stabla odluke, a kvadrati listove stabla odluke. Brojevi unutar krugova, odnosno kvadrata označavaju broj instanci trening skupa koje su asocirane datom čvoru, odnosno listu. U ovom primeru, veličina trening skupa iznosi 100 instanci. Svih 100 instanci koristi se za određivanje optimalne pozicije hiperravni u korenu stabla. Za određivanje optimalne pozicije hiperravni u svim ostalim čvorovima koristiće se progresivno manji broj instanci, kako se spuštamo na dublje nivoe stabla odluke. Na primer, korišćenjem optimalne hiperravni u korenu stabla, 100 instanci kompletnog trening skupa biće podeljene u dve grupe, grupu instanci koje se nalaze sa jedne strane hiperravni i drugu grupu instanci koje se nalaze sa druge strane hiperravni, u skladu sa *DTS* algoritmom. Pretpostavimo da se u ovom slučaju 30 instanci nalazi sa jedne strane hiperravni, a preostalih 70 instanci sa druge strane hiperravni, kao što je prikazano na slici 3.8. Imajući ovo u vidu, na slici 3.8 je prikazana jedna od mogućih distribucija instanci po čvorovima stabla odluke.

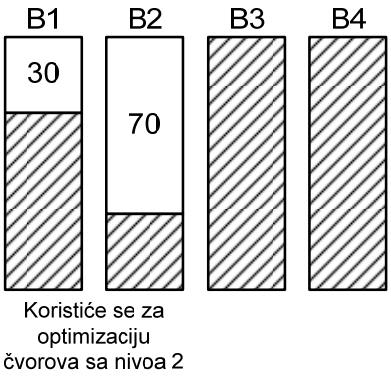


Slika 3.8 Stablo odluke sa brojem instanci trening skupa asociranim svakom čvoru odnosno listu stabla

Poznavanje distribucije instanci trening skupa po čvorovima stabla je od centralnog značaja za *DTS* algoritam, jer se samo asocirane instance posmatranom čvoru koriste za optimizaciju pozicije hiperravni u tom čvoru. Jedan od načina za asociranje instanci trening skupa svakom od čvorova stabla odluke bio bi da se koriste posebne memorije u koje bi se smeštale asocirane instance iz modula *M1*. Iako je ovo moguće rešenje, ono bi bilo vrlo neefikasno u terminima potrebne veličine memorije. Međutim, postoji i efikasnije rešenje. Umesto da se za svaki čvor čuvaju njemu asocirane instance, dovoljno je čuvati njihove indekse, odnosno pozicije na kojima su smeštene u trening skup memoriji, modulu *M1*. Na ovaj način, za posmatrani čvor je potrebno čuvati samo indekse njemu asociranih instanci, umesto samih instanci. Ovo je pristup koji je korišćen u *H_DTS* arhitekturi.

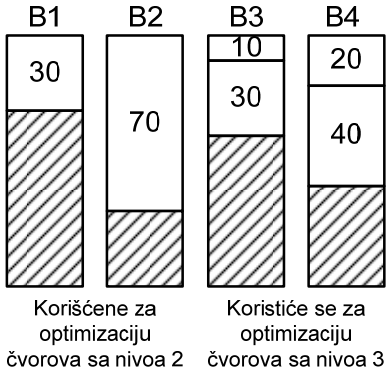
DTS algoritam formira stablo odluke koristeći *breath-first* tehniku, odnosno formira stablo tako što optimizuje sve čvorove iz tekućeg nivoa, pa tek onda prelazi na sledeći nivo. Ovakav pristup znatno olakšava manipulaciju skupovima indeksa asociranih različitim čvorovima u stablu odluke. Dve memorije koriste se za čuvanje asociranih indeksa instanci čvorovima iz tekućeg nivoa stabla odluke. Ova podaci se koriste da bi se izabrale i koristile samo asocirane instance prilikom optimizacije pozicije hiperravni za tekući čvor u stablu odluke. Nakon što se završi proces optimizacije pozicije hiperravni za tekući čvor, korišćenjem optimalne hiperravni vrši se podela asociranih instanci na dve grupe, instance koje se nalaze sa jedne strane optimalne hiperravni i na instance koje se nalaze sa druge strane hiperravni. Ove dve grupe instanci će zapravo biti asocirane čvorovima naslednicima tekućeg čvora. Informacija o asociranim instancama čvorovima naslednicima čuva se u preostale dve memorije. Nakon što se završi proces optimizacije čvorova iz tekućeg nivoa, prelazi se na optimizaciju čvorova iz sledećeg nivoa stabla odluke. Sada se koriste memorije u kojima su smešteni podaci o asociranim instancama za čvorove naslednike za pristup asociranim instancama, a u preostale dve memorije (u kojima se čuvala distribucija instanci po čvorovima iz prethodnog nivoa) biće smeštena distribucija instanci po njihovim čvorovima naslednicima. Kao što se može zaključiti, dve grupe od po dve memorije za čuvanje indeksa asociranih instanci se alternativno smenjuju, prilikom formiranja stabla odluke. Način korišćenja i alternativnog smenjivana memorija za čuvanje indeksa asociranih instanci opisaćemo na primeru formiranja stabla sa slike 3.8.

Na početku formiranja stabla sve četiri banke, *B1*-*B4*, su prazne. Za potrebe optimizacije položaja hiperravni u korenu stabla koriste se sve instance iz trening skupa. Nakon što se završi optimizacija hiperravni u korenu stabla, vrši se particija trening skupa pomoću optimalne hiperravni. Nakon particije formiraju se dva skupa instanci, jedan koji sadrži instance koje se nalaze sa jedne strane hiperravni i broji 30 instanci, i drugi koji sadrži instance koje se nalaze sa druge strane hiperravni i broji 70 instanci. Ova particija je konzistentna sa situacijom prikazanom na slici 3.8. Indeksi 30 instanci koje se nalaze sa jedne strane hiperravni smeštaju se u banku *B1*, dok se indeksi 70 instanci koje se nalaze sa druge strane hiperravni smeštaju u banku *B2*. Ovi indeksi će biti korišćeni u narednoj fazi da bi se optimizovala pozicija hiperravni lociranih u dva naslednika korena stabla. Banke *B3* i *B4* se trenutno ne koriste i prazne su. Sadržaj sve četiri banke nakon završene optimizacije korena stabla prikazan je na slici 3.9. Bitno je napomenuti da se optimizacija korena stabla odluke može smatrati i kao optimizacija svih čvorova koji se nalaze u stablu odluke na nivou 1.



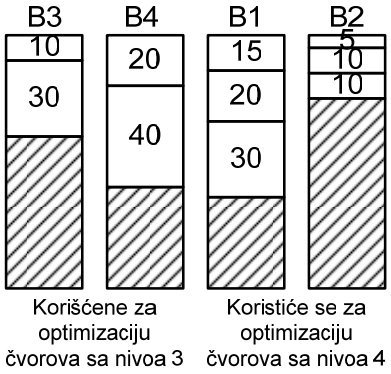
Slika 3.9 Sadržaj četiri memorijske banke za smeštanje indeksa asociranih instanci nakon optimizacije čvorova stabla iz nivoa 1

U sledećem ciklusu formiranja stabla odluke, koristeći indekse instanci iz banke *B1* optimizuje se levi naslednik korena stabla. Nakon završene optimizacije, trening skup of 30 instanci asociran levom nasledniku korena stabla deli se na dva podskupa. Prvi podskup čini 10 instanci koje se nalaze sa jedne strane hiperravni, a drugi podskup čini 20 instanci koje se nalaze sa druge strane hiperravni. Indeksi instanci iz ova dva podskupa smeštaju se u banke *B3* i *B4* respektivno. Isti postupak se zatim primenjuje za desnog naslednika korena stabla i formirani podskupovi indeksa instanci veličine 30 i 40 elemenata smeštaju se u banke *B3* i *B4*. Na ovom nivou u stablu mogu postojati samo dva čvora te je stoga proces optimizacije čvorova lociranih na drugom nivou u stablu završen, a izgled četiri memorijske banke prikazan je na slici 3.10.



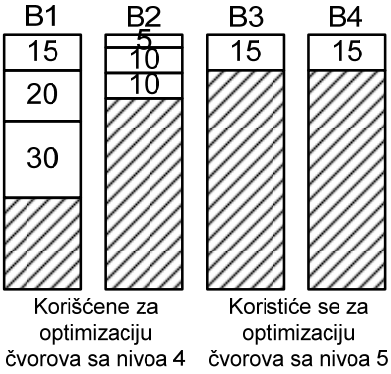
Slika 3.10 Sadržaj četiri memorijske banke za smeštanje indeksa asociranih instanci nakon optimizacije čvorova stabla iz nivoa 2

U sledećem ciklusu optimizuju se čvorovi iz trećeg nivoa stabla odluke. Za razliku od prethodnog ciklusa, sada se za optimizaciju čvorova koriste podaci smešteni u bankama *B3* i *B4*, a novi podaci o asociranim instancama čvorovima iz četvrtog nivoa smeštaju se u banke koje se sada ne koriste, a to su banke *B1* i *B2*. Prilikom smeštanja novih indeksa, prethodni sadržaj banaka *B1* i *B2* se briše, odnosno prepisuje sa novim podacima. Na ovom nivou mogu postojati maksimalno četiri čvora, a nakon što se optimizuju pozicije hiperravni u svakom od njih i izvrši particija skupova trening instanci koji su im bili pridruženi, sadržaj četiri memorijske banke biće kao na slici 3.11, u slučaju da se formira stablo sa slike 3.8. Bitno je napomenuti da stvaran broj čvorova koji postoji na tekućem nivou zavisi od strukture stabla koje se formira, odnosno od prirode problema koji se rešava. Svaki put kada se naiđe na list, on se ne optimizuje dalje, i dalji rast stabla u tom pravcu se zaustavlja. Ovo je slučaj sa listom označenim brojem 4 sa slike 3.8, kojem je asocirano ukupno 10 instanci. Tako da u slučaju stabla sa slike 3.8 u trećem nivou zapravo postoji samo 3 čvora i njihovom optimizacijom će se formirati ukupno 6 podskupova instanci koje će biti asocirane čvorovima/listovima iz četvrtog sloja stabla.



Slika 3.11 Sadržaj četiri memorijske banke za smeštanje indeksa asociranih instanci nakon optimizacije čvorova stabla iz nivoa 3

Opisani postupak se dalje ponavlja nad čvorovima iz četvrtog nivoa, pri čemu se koriste informacije o asociranim instancama iz banaka *B1* i *B2*, a nove informacije se smeštaju u banke *B3* i *B4*. Znači, još jednom je došlo do zamene banaka, kao što je ranije objašnjeno. Sadržaj banaka nakon optimizacije čvorova iz četvrtog nivoa prikazan je na slici 3.12. Bitno je napomenuti da u slučaju stabla odluke sa slike 3.8 u četvrtom sloju postoji samo jedan čvor, označen brojem 12, tako da će postojati samo dva podskupa instanci koje su asocirane čvorovima iz petog sloja stabla.



Slika 3.12 Sadržaj četiri memorijske banke za smeštanje indeksa asociranih instanci nakon optimizacije čvorova stabla iz nivoa 5

U slučaju stabla odluke sa slike 3.8, postupak formiranja bi se završio jer na petom nivou u stablu nema ni jednog novog čvora koji treba optimizovati. U opštem slučaju postupak alternativnog smenjivanja banaka bi se naravno nastavio sve dok u narednom sloju postoji barem jedan čvor koji je potrebno optimizovati, što se jasno vidi i u *DTS* algoritmu.

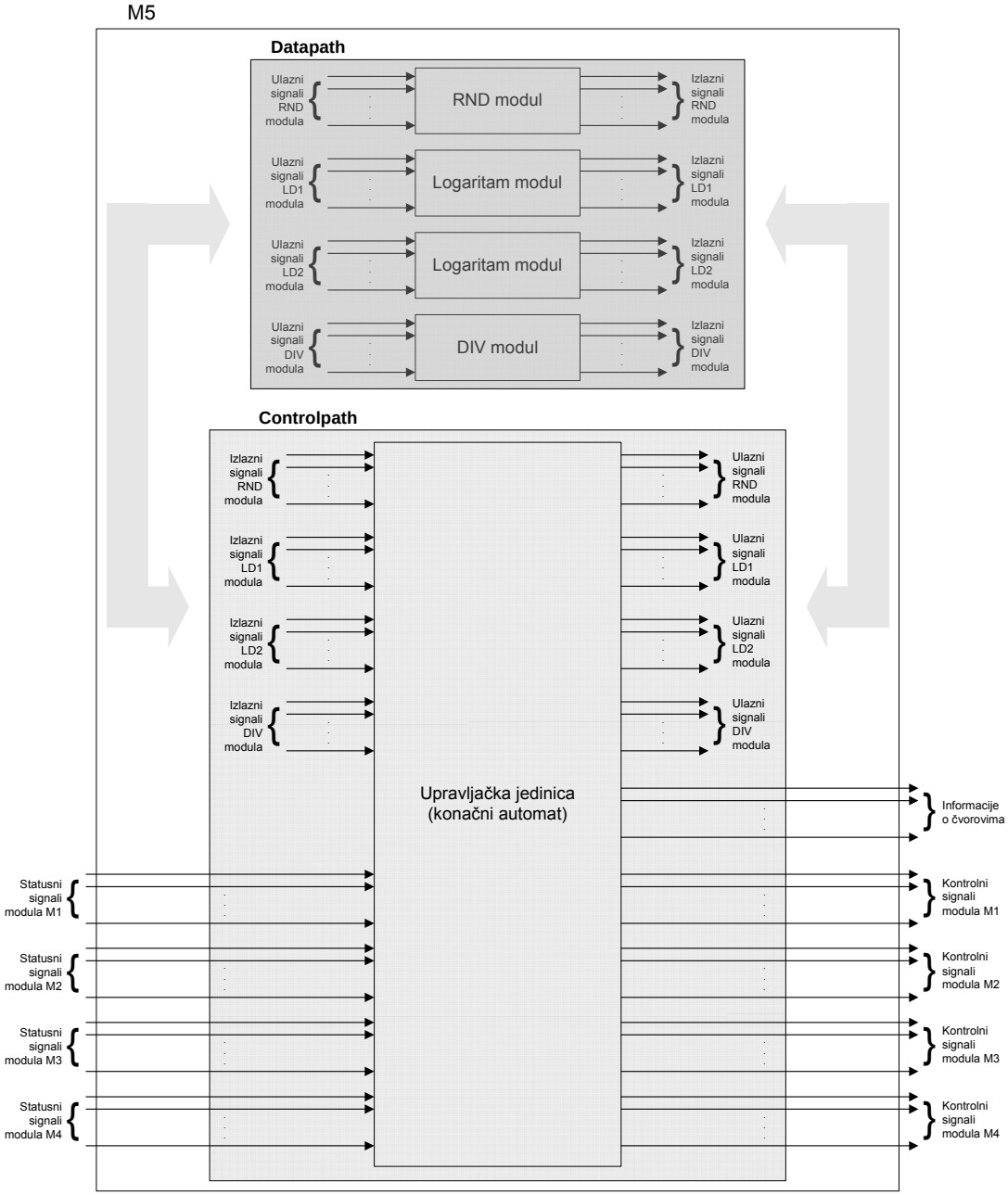
Da bi se mogla izračunati vrednost fitnes funkcije (3.3) tokom optimizacije hiperravni u svakom od čvorova, pored indeksa asociranih instanci potrebno je znati koliko instanci iz svake klase je asocirano posmatranom čvoru. Ove informacije se čuvaju u dodatne dve banke iz modula M4. Veličina ove dve dodatne banke može se izračunati korišćenjem sledeće formule.

$$RAM\_Size_{M4\_B5-B6} = N_{cl} \tag{3.15}$$

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

3.2.1.5 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 opisane u Prilogu B. 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 3.12.



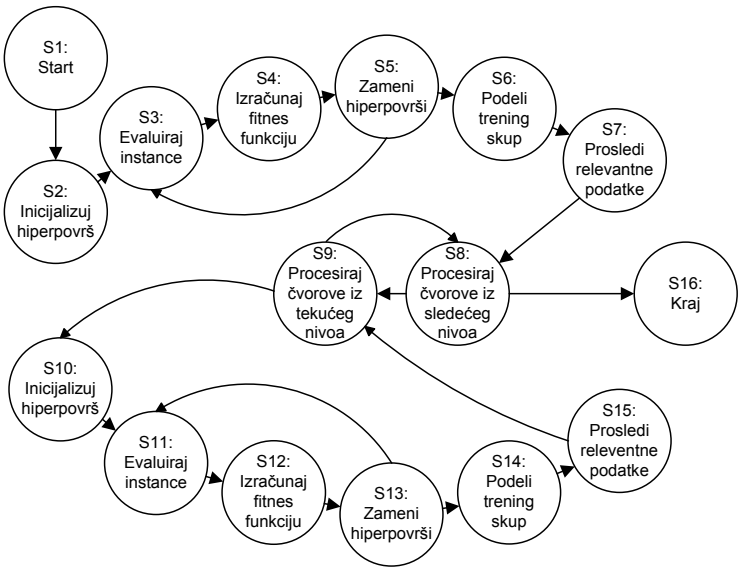
Slika 3.12 Detaljna struktura modula M5

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 3.12.

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 (3.3).

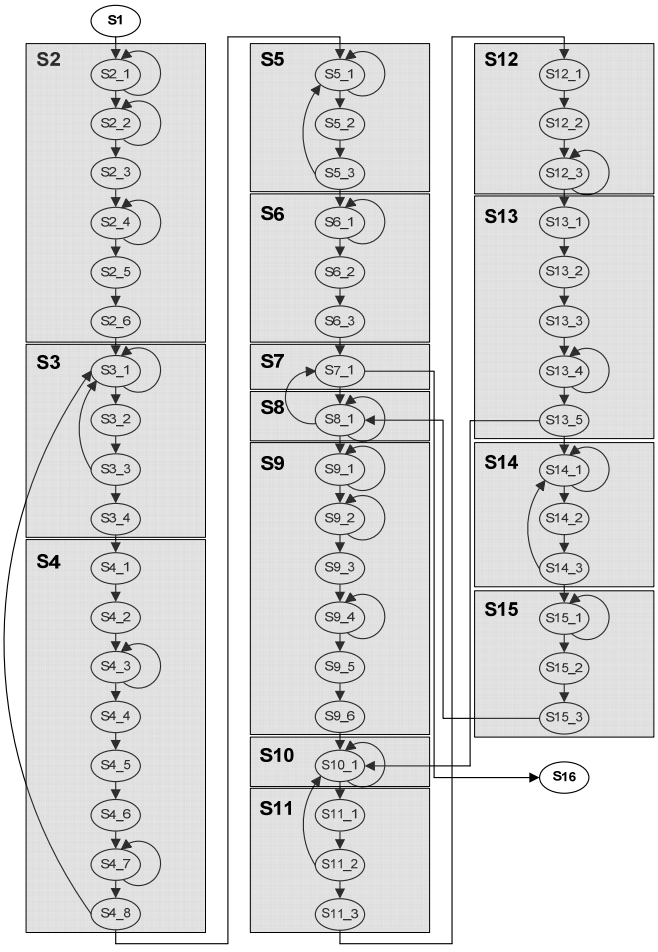
Upravljačka jedinica

Upravljačka jedinica realizovana je kao konačni automat čiji je uprošćeni dijagram stanja i prelaza prikazan na slici 3.13.



Slika 3.13 Uprošćeni dijagram stanja upravljačke jedinice

Dijagram stanja prikazan na slici 3.13 prikazuje samo najvažnije korake koje izvodi upravljačka jedinica tokom svog rada. Funkcija koja se izvodi u svakom od stanja takođe je naznačena na slici 3.13. Svako prikazano stanje zapravo se sastoji iz većeg broja stanja koja zajedno ostvaruju posmatranu funkciju. Detaljni izgled dijagrama stanja upravljačke jedinice prikazan je na slici 3.14 radi potpunosti, iako on nije neophodan za razumevanje rada upravljačke jedinice. Kao što se sa slike 3.14 može videti, upravljačka jedinica realizovana je kao konačni automat sa ukupno 52 stanja. Stanja su grupisana u celine, od kojih svaka odgovara po jednom stanju sa slike 3.13.



Slika 3.14 Potpuni dijagram stanja upravljačke jedinice

Za objašnjenje rada upravljačke jedinice dovoljno je koristiti dijagram stanja sa slike 3.13. 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 3.2.1.4. 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.

Modul za generisanje pseudoslučajnih brojeva

RND modul implementira generator pseudoslučajnih brojeva baziran na *MT19937* algoritmu [36], koji ima periodu od (219937-1) brojeva i 623-dimenzionalnu uniformnu raspodelu verovatnoće. Ove osobine čine ga posebno interesantnim za korišćenje u naučnim aplikacijama. Detalji *MT19937* algoritma prikazani su na sledećoj slici.

rnd_broj = MT19937 ()

Izlaz:

rnd_broj *Generisani pseudoslučajni broj*

- Ako je $k = 623$
  temp = (MT(k)&0x80000000) | MT(0)&0x7FFFFFFF
- Inače
  temp = (MT(k)&0x80000000) | MT(k+1)&0x7FFFFFFF

- Ako je *k+397 < 624*
  Ako je *temp&0x1=0*
    - *MT(k)=MT(k+397) ^ (temp >> 1);*

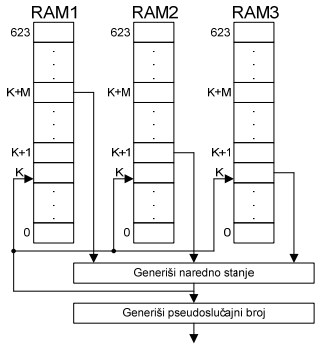
  Inače
    - *MT(k)=MT(k+397) ^ (temp >> 1) ^ 0x9908B0DF;*
- Inače
  Ako je *temp&0x1=0*
    - *MT(k)=MT(k+397-624) ^ (temp >> 1);*

  Inače
    - *MT(k)=MT(k+397-624) ^ (temp >> 1) ^ 0x9908B0DF;*
- *y=MT(k)*
- *k=k+1*
- Ako je *k=624*
  k=0
- *y=y ^ (y >> 11)*
- *y=(y ^ (y << 7)) & 0x9D2C5680*
- *y=(y ^ (y << 15)) & 0FC60000*
- *rnd_broj=y>>18*

Algoritam 3.5 MT19937 *algoritam za generisanje pseudoslučajnih brojeva*

MT19937 algoritam koristi niz *MT* od 624 elementa za čuvanje informacije o trenutnom stanju generatora pseudoslučajnih brojeva. Svaki od ovih elemenata predstavlja jedan 32-bitni broj. Pre početka korišćenja generatora ovaj niz je potrebno inicijalizovati. Algoritam se ciklično kreće kroz niz *MT* i koristeći vrednosti tri elementa ovoga niza sa indeksima *k, k+1* i *k+397* generiše novu vrednost elementa sa indeksom k. Ova vrednost se upisuje u niz *MT*, a ujedno se koristi i za generisanje narednog pseudoslučajnog broja izvođenjem odgovarajućeg broja logičkih operacija i operacija pomeranja. U gornjem algoritmu simboli &, ^, |, >> i << označavaju standardne logičke operacije i operacije pomeranja definisane u standardnom C programskom jeziku.

U slučaju hardverske implementacije *MT19937* algoritma sistem se sastoji iz tri modula, RAM memorije koja čuva trenutno stanje generatora slučajnih brojeva (niz *MT*), modula za generisanje novog stanja koji određuje novu vrednost elementa niza *MT* sa indeksom *k* i modula koji transformiše ovu vrednost zapravo generiše pseudoslučajni broj. U slučaju sekvencijalne implementacije potrebno je četiri ciklusa (tri ciklusa čitanja iz memorije i jedan ciklus upisa u memoriju) za generisanje jednog pseudoslučajnog broja. Za potrebe hardverske implementacije *HereBoy* algoritma ovo bi bilo izrazito neefikasno rešenje jer bi značajno usporilo rad. Ukoliko bi se koristila višepristupna RAM memorija sa tri porta za čitanje i jednim portom za upis, bilo bi moguće generisati jedan pseudoslučajni broj u svakom ciklusu. Projektovanje višepristupne memorije je prilično zahtevna operacija tako da je u ovom slučaju ona realizovana korišćenjem tri modula sa istim portom za upis i različitim portovima za čitanje podataka. Struktura čitavog modula za generisanje pseudoslučajnih brojeva koji je u stanju da generiše po jedan pseudoslučajni broj u svakom ciklusu prikazana je na slici 3.15.



Slika 3.15 Arhitektura modula za generisanje pseudoslučajnih brojeva

Modul sadrži tri RAM memorije jednakih veličina od 624 32-bitne lokacije. Svaka od ovih memorija čuva istu sliku trenutnog stanja generatora. Da bi ova slika bila konzistentna, tokom rada se svaki novi podatak mora istovremeno upisivati u sve tri memorije. Ovo je postignuto na taj način što su ulazni portovi sve tri memorije povezani zajedno. Sa druge strane svaka od memorija ima po jedan port za čitanje podataka koji je nezavisan od portova za čitanje druge dve memorije, obezbeđujući da se u istom ciklusu pročitaju tri različita podatka na lokacijama *k*, *k+1* i *k+397*. Preostala dva modula („*Generiši naredno stanje*“ i „*Generiši pseudoslučajni broj*“) izvršavaju korake iz *MT19937* algoritma koji su neophodni da bi se formiralo naredno stanje generatora i generisao pseudoslučajni broj.

Modul za realizaciju operacije računanja logaritma

Prilikom računanja fitnes funkcije (3.3) neophodno je izračunati vrednost logaritma osnove dva nekog celog broja. Vrednost logaritamske funkcije se u digitalnim sistemima tipično računa razvojem logaritma u Tejlorov red ili primenom nekog sličnog iterativnog postupka. Ovakav pristup izračunavanju vrednosti logaritamske funkcije pogodan je u slučaju korišćenja mikroprocesora ali je prilično neefikasan ukoliko se planira direktna hardverska realizacija. Takođe ovaj pristup nije pogodan ukoliko je potrebno brzo izračunati vrednost logaritamske funkcije. Pod ovakvim uslovima znatno efikasniji pristup, kako u pogledu potrebnih hardverskih resursa i brzine računanja, bazira se na korišćenju *look-up* tabela.

Pretpostavimo da želimo izračunati vrednost logaritma osnove dva nekog pozitivnog ograničenog celog broja, *x*, *ld(x)*. Prvi korak jeste da se odredi broj *k* takav da važi

$$0 < x < 2^k \tag{3.16}$$

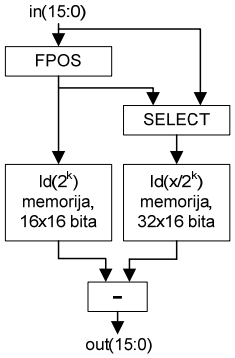
Sada logaritam osnove dva broja *x* možemo odrediti i kao

$$ld\left(x\right) = ld\left(2^k\right) + ld\left(\frac{x}{2^k}\right) = k + ld\left(\frac{x}{2^k}\right) \tag{3.17}$$

Bitno je naglasiti da je argument drugog logaritma iz izraza (4.20) uvek iz intervala

$$0 < \frac{x}{2^k} < 1 \tag{3.18}$$

Te se može izračunati unapred i smestiti u odgovarajuću tabelu. Veličina potrebne tabele direktno zavisi od zahtevane greške aproksimacije, ali se i sa relativno malom tabelom mogu postići jako dobri rezultati. Bitno je takođe naglasiti da će drugi član u izrazu (3.17) uvek biti negativan zbog uslova (3.18). Struktura hardverskog modula koji implementira gore opisani postupak računanja logaritamske funkcije prikazana je na slici 3.16.

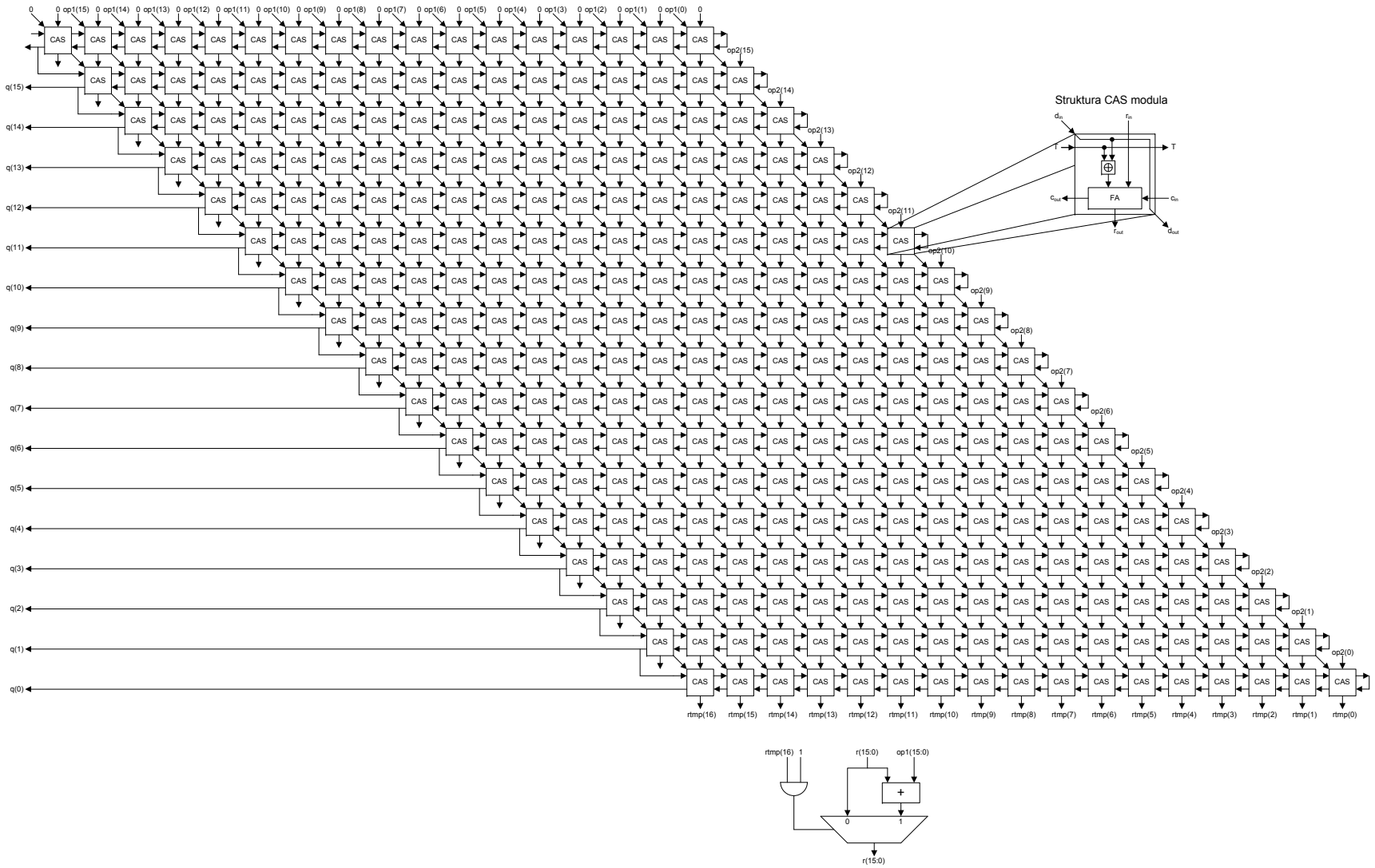


Slika 3.16 Arhitektura modula za računanje logaritma

Modul *FPOS* zadužen je za određivanje vrednosti broja *k* na osnovu vrednosti ulaznog broja *in* čiji logaritam želimo izračunati. *SELECT* modul, koristeći broj *k* iz ulaznog broja *in* izdvaja 5 bita sa odgovarajuće pozicije. Ovih 5 bita koristi se kao adresa za adresiranje memorije u kojoj se nalaze smeštene vrednosti funkcije *ld(x)* kada je *0<x<1*. Korišćenje samo 5 bita za formiranje adrese za *look-up* tabelu dovoljno je da se greška aproksimacije održi dovoljno malom, a da sa druge strane omogući da veličina *look-up* tabele bude prihvatljiva.

Modul za realizaciju operacije deljenja dva broja

Prilikom računanja vrednosti fitnes funkcije definisane izrazom (3.3) neophodno je izvoditi i operacije deljenja brojeva. Stoga je u okviru M5 modula implementiran i delitelj brojeva, *DIV* modul. U dostupnoj literaturi postoji veliki broj različitih arhitektura za hardversku implementaciju operacije deljenja kao što su arhitekture za sekvencijalnu realizaciju deljenja [37, str. 39], neobnavljajuće (*nonrestoring*) arhitekture, *SRT* arhitekture, i mrežne (*array*) arhitekture. Mrežne arhitekture su pogotovo pogodne u slučaju da se želi postići velika brzina rada, ali zahtevaju veći utrošak resursa. Obzirom da je brzina rada sistema bila jedan od kritičnih parametara prilikom razvoja *H_DTS* arhitekture, za realizaciju operacije deljenja odabrana je mrežna arhitektura. Mrežna arhitektura za deljenje brojeva bazirana je na činjenici da se deljenje dva *n*-bitna broja može realizovati izvođenjem *n* operacija oduzimanja ili sabiranja. Ukoliko se ove operacije realizuju u paraleli korišćenjem *n* sabirača/oduzimača, pri čemu svaki naredni sabirač/oduzimač obrađuje izlaz prethodnog modula, dobija se mrežna arhitektura za izvođenje operacije deljenja. Za potrebe izračunavanja vrednosti fitnes funkcije (3.3) bilo je neophodno realizovati delenje dva neoznačena 16-bitna broja. Struktura mrežnog delitelja dva neoznačena 16-bitna broja koji je implementiran prikazana je na slici 3.17.



Slika 3.17 Arhitektura modula za deljenje dva neoznačena 16-bitna broja

Kao što se sa slike 3.17 može videti, mrežna arhitektura sastoji se iz odgovarajućeg broja identičnih slojeva, pri čemu svaki od njih predstavlja kontrolisani sabirač/oduzimač. Svaki od ovih modula formiran je od odgovarajućeg broja *CAS* (*Controlled Add Subtract*) ćelija, koje predstavljaju 1-bitni kontrolisani sabirač/oduzimač. Pomoću ulaza T može se kontrolisati funkcija svakog sabirača/oduzimača. Ukoliko je T=0 izvodi se operacija sabiranja, a ukoliko je T=1 izvodi se operacija oduzimanja. Koju operaciju je potrebno izvršiti u sledećem nivou određuje c_{out} izlaz modula iz tekućeg nivoa. Nakon propagacije signala kroz sistem, količnik q, deljenja dva 16-bitna neoznačena broja dobija se kombinovanjem c_{out} izlaza iz svakog nivoa, a ostatak r, kao izlaz kontrolisanog sabirača/oduzimača iz poslednjeg sloja.

3.2.2 Primer formiranja stabla odluke pomoću *H_DTS* arhitekture

U ovom odeljku ilustrovaćemo način formiranja stabla odluke korišćenjem *H_DTS* arhitekture na jednom konkretnom primeru, rešavanju poznatog problema dve spirale. Ovaj problem sastoji se od 80 instanci koje pridaju jednoj od dve moguće klase, pri čemu po 40 instanci pripada svakoj od klasa. Svaka od instanci je opisana pomoću dva atributa. Instance su tako raspoređene u ravni da predstavljaju dve isprepletene spirale. Potrebno je formirati stablo odluke koje će razdvojiti instance koje pripadaju jednoj klasi od instanci koje pripadaju drugoj klasi. U ovom primeru pomoću *H_DTS* arhitekture formiraćemo neortogonalno stablo odluke. Prikazani tok formiranja stabla odluke koji sledi odnosi se na situaciju kada je *H_DTS* arhitektura konfigurisana sa sledećim parametrima:

- maksimalni broj generacija *HereBoy* algoritma, G = 10000,

- maskimalni broj mutiranih bitova, P_M = 0.6,

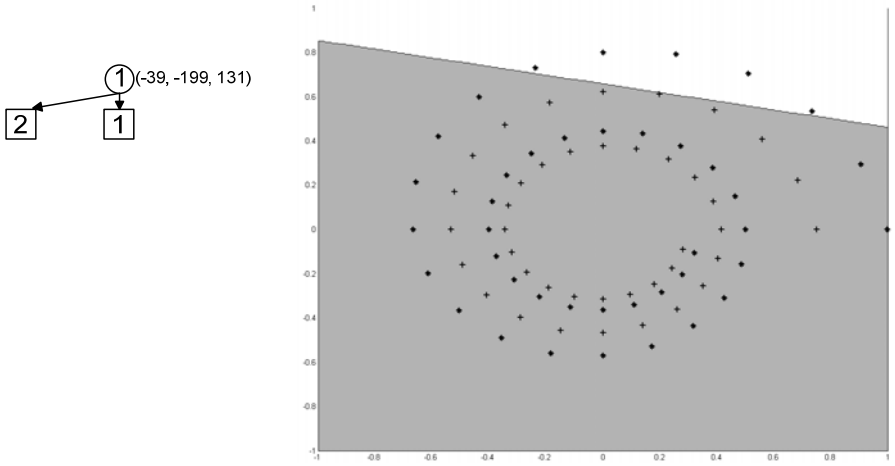
- maksimalna verovatnoća prihvatanja, P_P = 0.2

Prvi korak prilikom formiranja stabla odluke jeste da se optimizuje položaj hiperravni u korenu stabla, koristeći sve raspoložive instance iz trening skupa. Nakon 1000 iteracija *HereBoy* algoritma optimalni položaj hiperravni u korenu stabla odluke kao i izgled stabla odluke formiranog do sada prikazani su na slici 3.18.

Na slici 3.18 prikazana je particija prostora atributa u slučaju kada bi se formiranje stabla odluke zaustavilo nakon optimizacije korena stabla odluke. Upravo zbog toga su oba naslednika korena stabla proglašeni listovima kojima su dodeljene klase kojima pripada većina instanci asociranih svakom od listova. U ovom slučaju, kao što se sa slike može videti, stablo nije u stanju da u potpunosti razdvoji instance jedne klase od instanci druge klase, što je bilo i očekivano. Stoga će rast stabla morati da se nastavi, i da se trenutni listovi transformišu u čvorove stabla.

Pored korena stabla odluke, označenog sa krugom sa brojem jedan, ispisane su vrednosti koeficijenata optimalne hiperravni. Kako je reč o klasifikacionom problemu sa dva atributa, hiperravan je u ovom slučaju definisana pomoću tri koeficijenta. Vrednosti koeficijenata su na slici prikazane kao celi brojevi, iako je zapravo reč o realnim brojevima. Ovo je urađeno prvenstveno radi preglednosti. Stvarne vrednosti koeficijenata hiperravni mogu se dobiti tako što se prikazane celobrojne vrednosti podele sa 512. Tako bi u slučaju koeficijenata hiperravni smeštene u korenu stabla stvarne vrednosti koeficijenata bile

$$(-0.076171875, -0.388671875, 0.255859375) \tag{3.19}$$

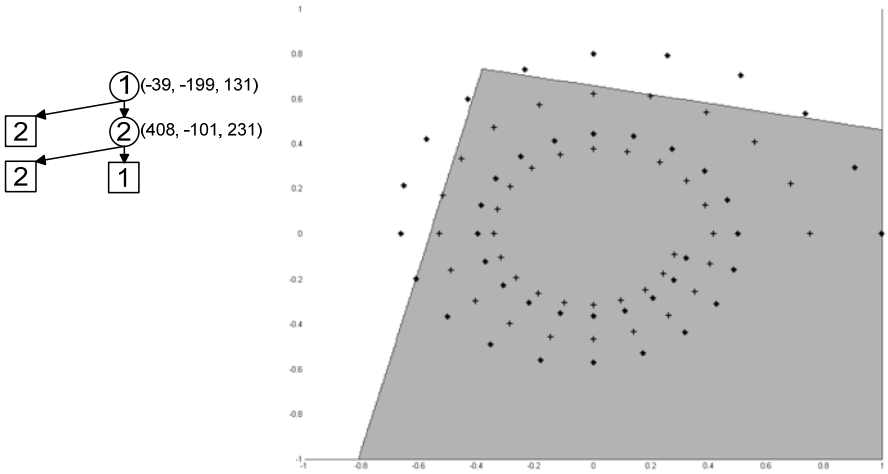


Slika 3.18 Izgled stabla odluke i trenutna particija prostora atributa nakon optimizacije korena stabla

Nakon što se završila optimizacija korena stabla odluke, upravljačka jedinica emituje relevantne podatke o korenu stabla (koeficijente optimalne pozicije hiperravni, kao i informacije o čvorovima naslednicima), a

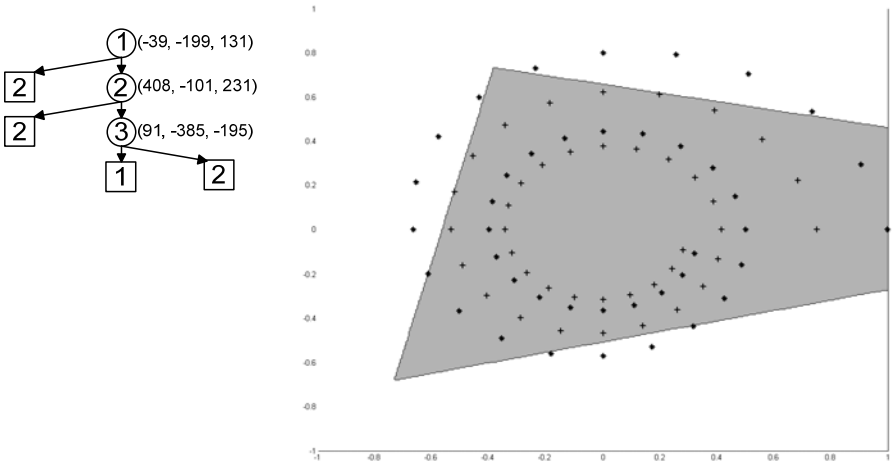
zatim proverava da li postoji barem jedan čvor u narednom sloju stabla. U ovom trenutku upravljačka jedinica se nalazi u stanju S8, prikazanom na slici 3.13. U ovom primeru, na drugom nivou postoji jedan čvor, obzirom da su levom nasledniku asocirane samo instance koje pripadaju klasi 2 te se on može proglasiti listom. Upravljačka jedinica prelazi u stanje S9, a zatim za svaki čvor iz tekućeg nivoa (u ovom primeru ima samo jedan takav čvor, to je desni naslednik korena stabla) vrši optimizaciju pozicije hiperravni, stanja S10-S15. Nakon što se završila optimizacija položaja hiperravni za desnog naslednika korena stabla, upravljačka jedinica emituje relevantne podatke o optimizovanom čvoru i ponovo se vraća u stanje S9, a obzirom da je to i jedini čvor koji je neophodno optimizovati iz tekućeg sloja, odmah prelazi u stanje S8.

Izgled stabla odluke formiranog do sada prikazan je na slici 3.19. Kao što se može videti sa slike, stablo odluke je sada uvećano i vrši particiju prostora atributa korišćenjem dve hiperravni. Kao i u prethodnom slučaju, oba naslednika čvora 2 su proglašeni listovima kojima su pridružene dominantne klase kojima pripada većina asociranih instanci. Sa slike se može videti da i ovo stablo nije u stanju da u potpunosti razdvoji instance jedne klase od instanci druge klase, te je stoga neophodan dalji rast stabla.



Slika 3.19 Izgled stabla odluke i trenutna particija prostora atributa nakon optimizacije čvorova sa drugog nivoa u stablu

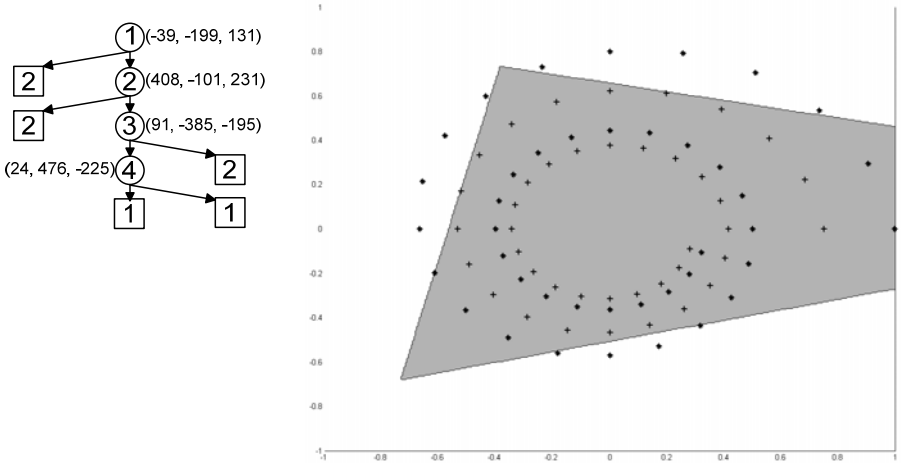
Na trećem nivou u stablu postoji samo jedan čvor koji je potrebno optimizovati, to je desni naslednik čvora 2. Nakon optimizacije položaja hiperravni u ovom čvoru upravljačka jedinica ponovo prelazi u stanje S8. Particija prostora atributa nakon dodavanja novog čvora na trećem nivou prikazana je na slici 3.20. Na slici je prikazan i trenutni izgled formiranog stabla. Pošto i ovo stablo nije u stanju da u potpunosti razdvoji instance jedne klase od instanci druge klase, postupak rasta stabla se mora dalje nastaviti.



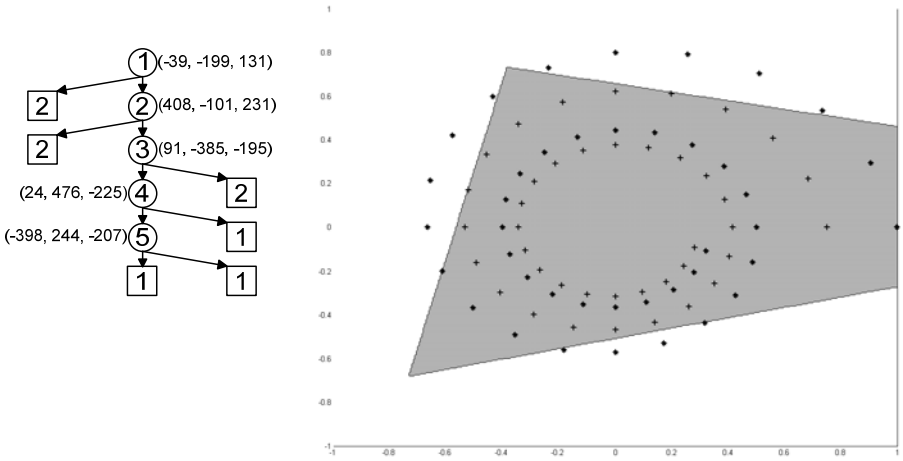
Slika 3.20 Izgled stabla odluke i trenutna particija prostora atributa nakon optimizacije čvorova sa trećeg nivoa u stablu

Na četvrtom nivou stabla takođe je neophodno optimizovati jedan čvor. Ovoga puta reč je o levom nasledniku čvora 3. Slika 3.21 prikazuje strukturu stabla odluke i particiju prostora atributa nakon završene optimizacije čvorova iz četvrtog nivoa u stablu odluke. I ovo stablo nije u stanju da u potpunosti razdvoji instance jedne klase od instanci druge klase, pa se rast stabla nastavlja. Obzirom da se sve do osmog nivoa stabla odluke situacija ponavlja, odnosno u svakom od slojeva optimizuje se po samo jedan čvor, slike 3.22 do

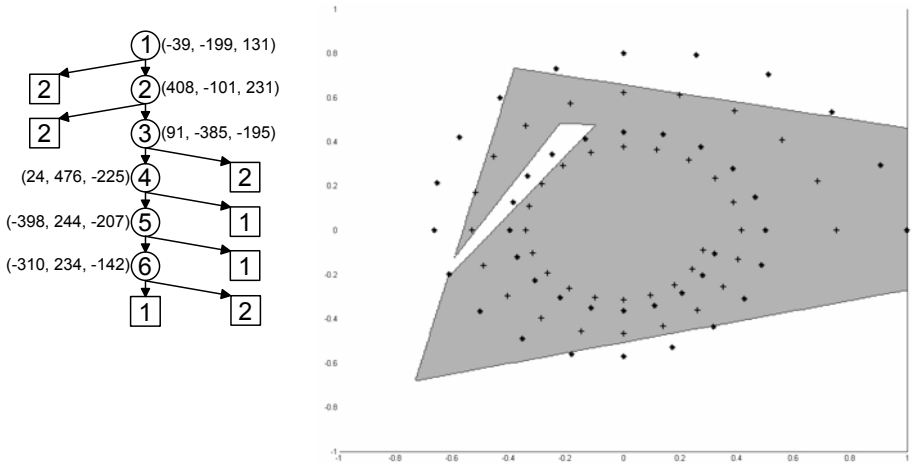
3.24 prikazuju rast stabla odluke i promenu načina na koji se vrši particija prostora atributa bez dodatnih komentara.



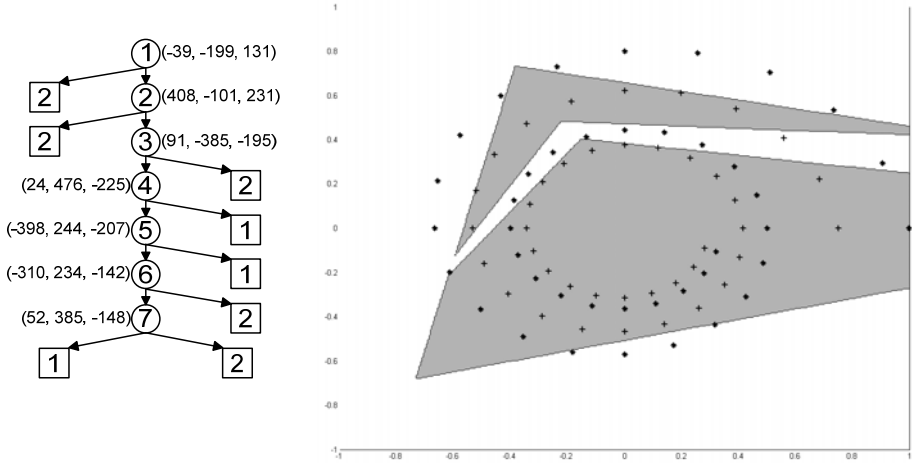
Slika 3.21 Izgled stabla odluke i trenutna particija prostora atributa nakon optimizacije čvorova sa četvrtog nivoa u stablu



Slika 3.22 Izgled stabla odluke i trenutna particija prostora atributa nakon optimizacije sa petog nivoa u stablu

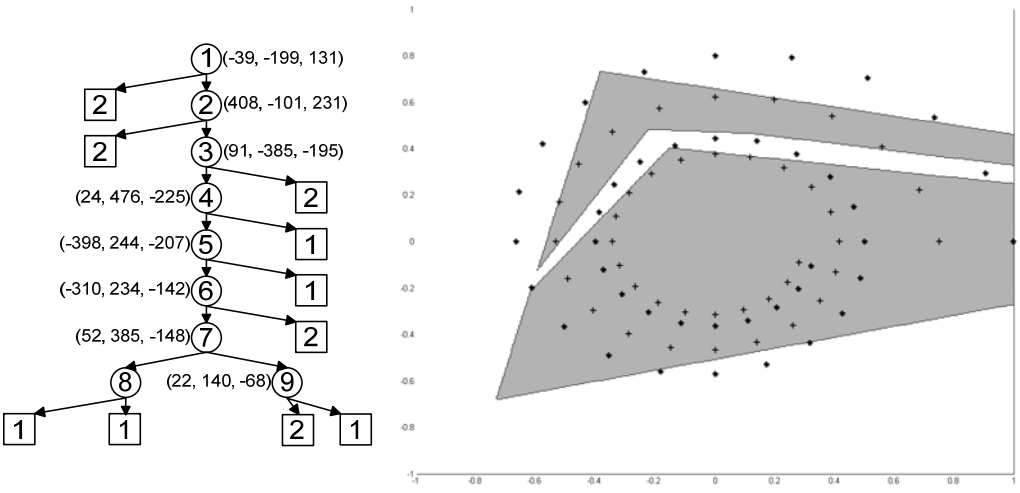


Slika 3.23 Izgled stabla odluke i trenutna particija prostora atributa nakon optimizacije sa šestog nivoa u stablu

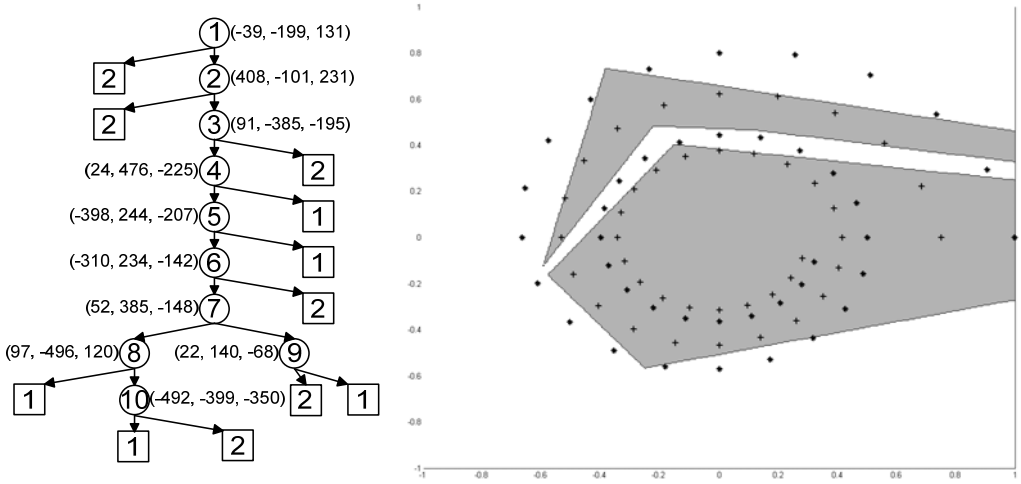


Slika 3.24 Izgled stabla odluke i trenutna particija prostora atributa nakon optimizacije sa sedmog nivoa u stablu

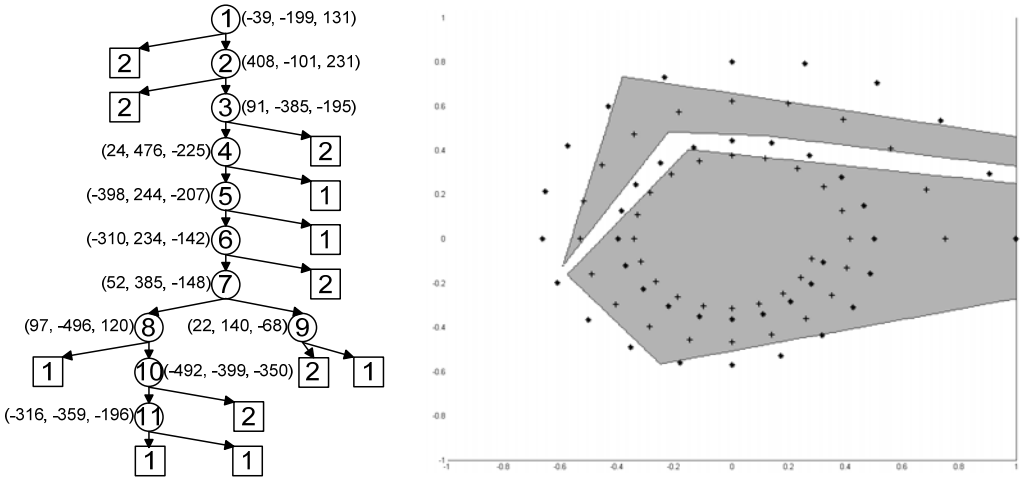
Do nove situacije dolazi se prilikom optimizacije čvorova koji se nalaze na osmom nivou u stablu odluke. Po prvi put potrebno je optimizovati dva čvora u jednom te istom nivou. Postupak optimizacije se prvo primenjuje na levog naslednika čvora 7, a nakon toga i na desnog naslednika čvora 7. U slučaju da je neophodno optimizovati još veći broj čvorova u lociranih u istom nivou postupak bi se nastavio, krećući se sa leva u desno u posmatranom stablu. Slika 3.25 prikazuje strukturu do sada formiranog stabla i trenutnu particiju prostora atributa. Pošto ni ovo stablo nije u stanju da u potpunosti razdvoji instance jedne klase od instanci druge klase, postupak dodavanja novih čvorova nastavlja se i dalje i prikazan je na slikama 3.26 do 3.33.



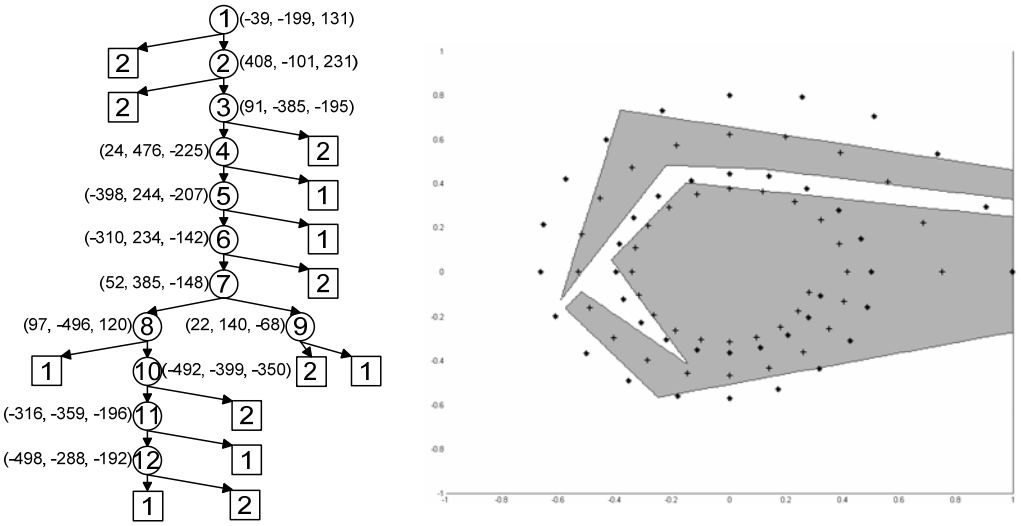
Slika 3.25 Izgled stabla odluke i trenutna particija prostora atributa nakon optimizacije sa osmog nivoa u stablu



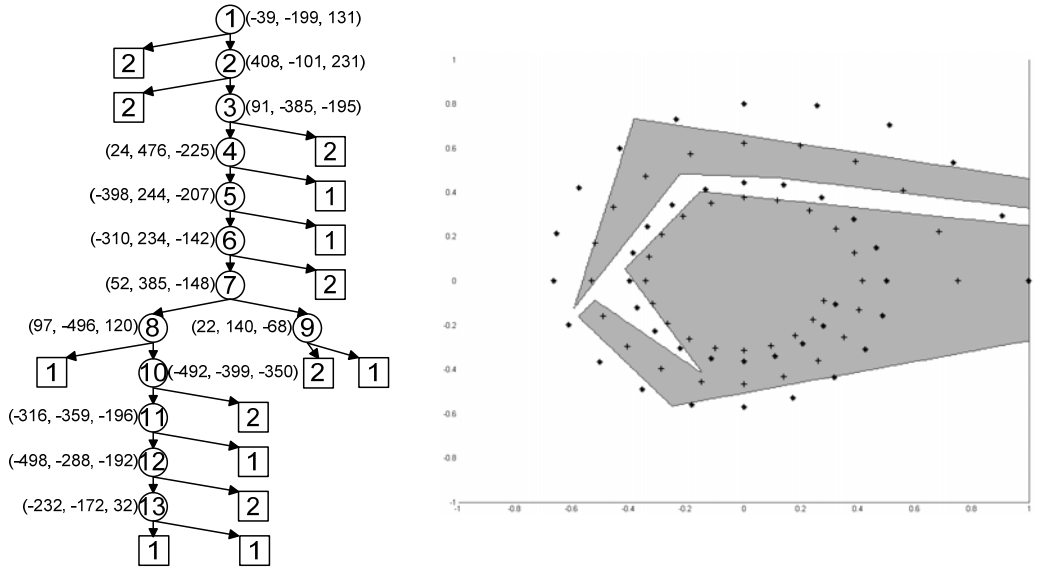
Slika 3.26 Izgled stabla odluke i trenutna particija prostora atributa nakon optimizacije sa devetog nivoa u stablu



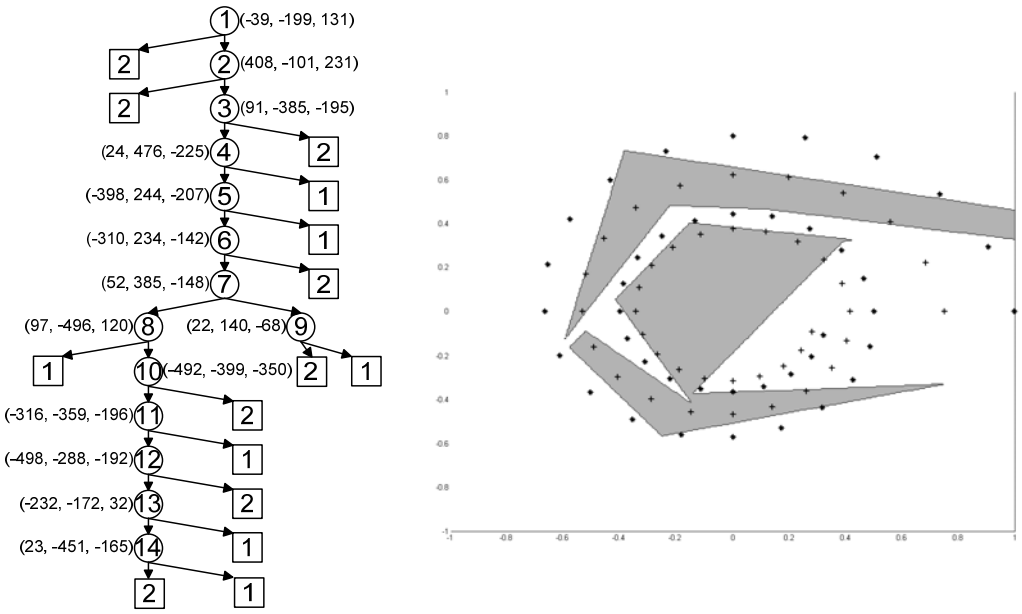
Slika 3.27 Izgled stabla odluke i trenutna particija prostora atributa nakon optimizacije sa desetog nivoa u stablu



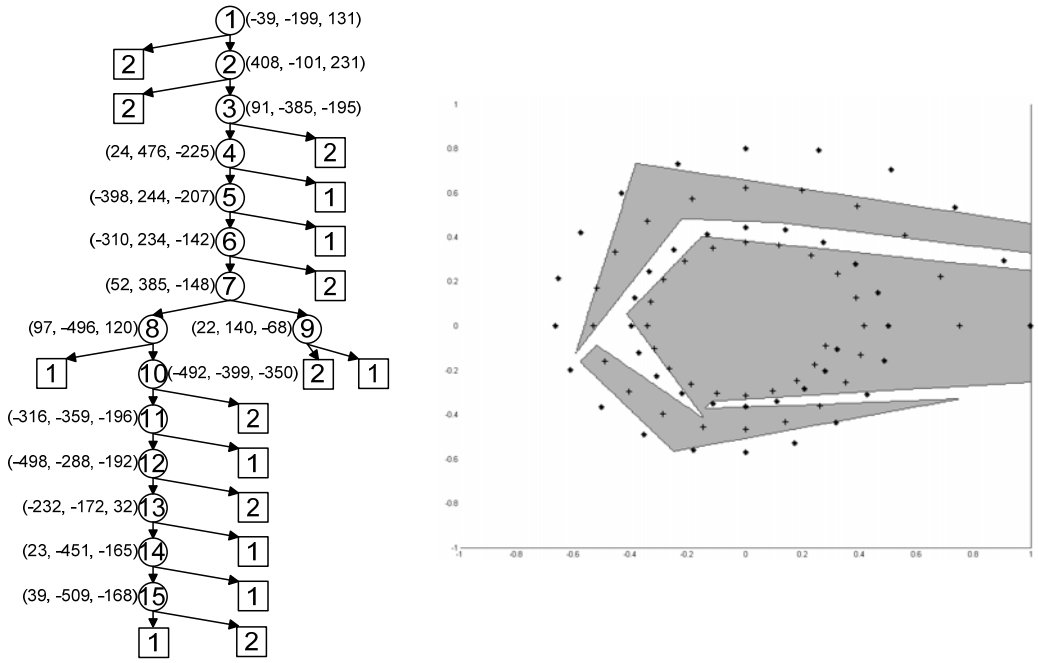
Slika 3.28 Izgled stabla odluke i trenutna particija prostora atributa nakon optimizacije sa jedanaestog nivoa u stablu



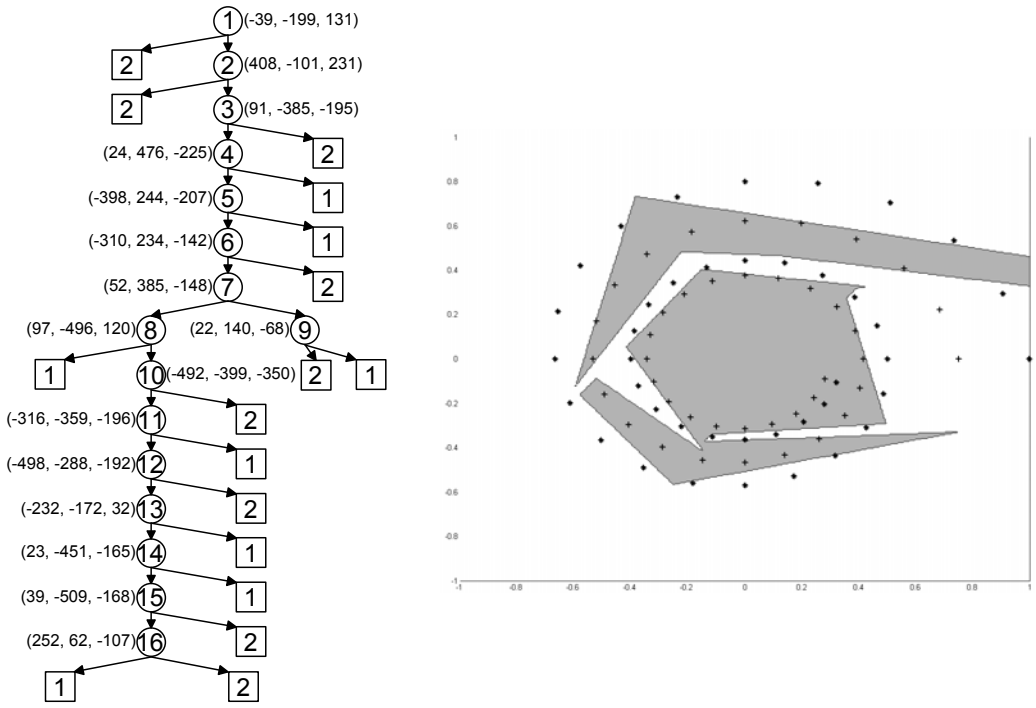
Slika 3.29 Izgled stabla odluke i trenutna particija prostora atributa nakon optimizacije sa dvanaestog nivoa u stablu



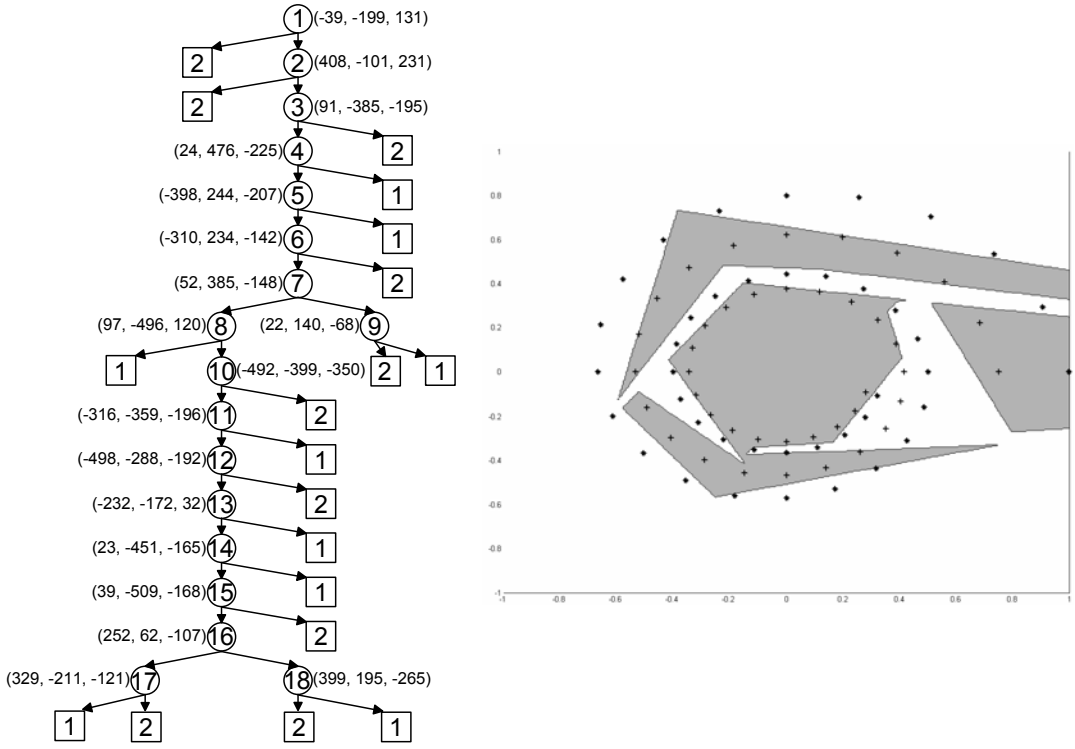
Slika 3.30 Izgled stabla odluke i trenutna particija prostora atributa nakon optimizacije sa trinaestog nivoa u stablu



Slika 3.31 Izgled stabla odluke i trenutna particija prostora atributa nakon optimizacije sa četrnaestog nivoa u stablu



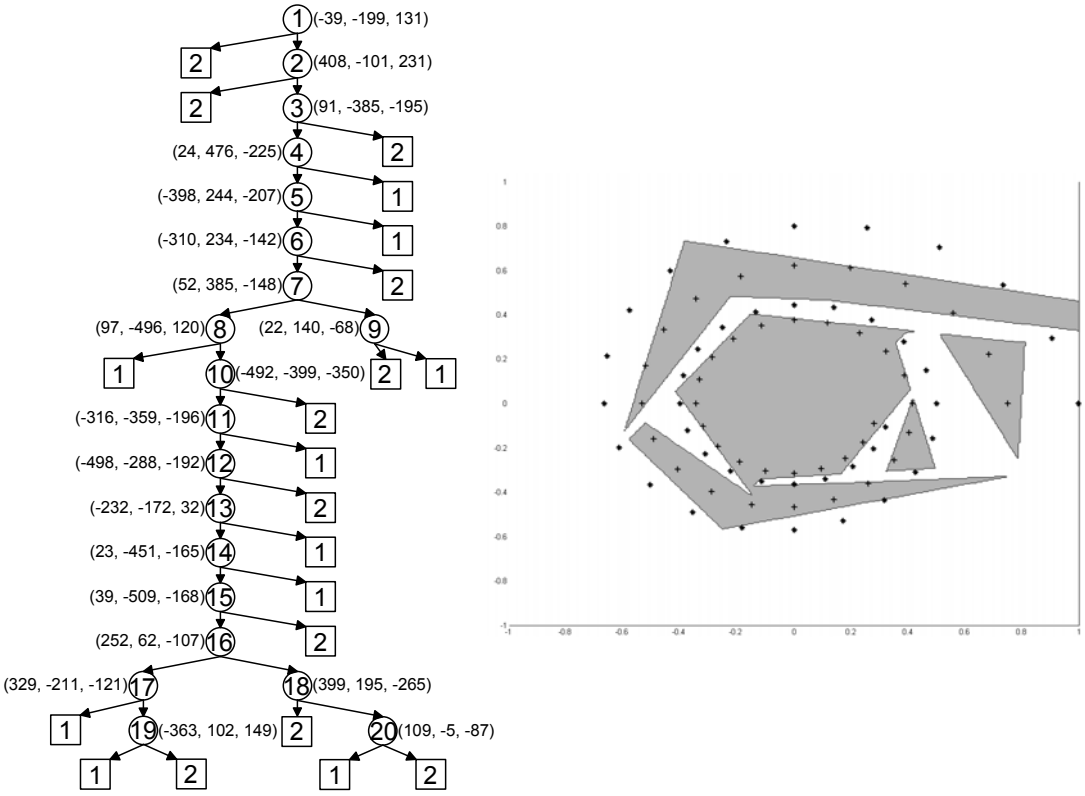
Slika 3.32 Izgled stabla odluke i trenutna particija prostora atributa nakon optimizacije sa petnaestog nivoa u stablu



Slika 3.33 Izgled stabla odluke i trenutna particija prostora atributa nakon optimizacije sa šesnaestog nivoa u stablu

Nakon optimizacije čvorova iz šesnaestog nivoa u stablu odluke njegov izgled prikazan je na slici 3.33. U sedamnaestom čvoru postoji ukupno četiri čvora kandidata za dalju optimizaciju. Od ta četiri samo dva čvora je zapravo neophodno dalje optimizovati dok se preostala dva mogu zameniti listovima. Nakon optimizacije dva čvora iz sedamnaestog nivoa izgled stabla odluke i particija prostora atributa pomoću njega prikazani su na slici 3.34. Kao što se sa slike može videti ovo stablo odluke u potpunosti razdvaja instance jedne klase od instanci druge klase. Ovo je indikacija da je postupak formiranja stabla odluke za posmatrani problem dve spirale završen i da dalji rast stabla nije neophodan. Nakon završetka optimizacije čvorova iz sedamnaestog

nivoa upravljačka jedinica se vraća u stanje S8, a pošto nema čvorova iz osamnaestog nivoa koje je potrebno optimizovati upravljačka jedinica prelazi u stanje S16 i završava rad.



Slika 3.34 Konačni izgled stabla odluke i particija prostora atributa

3.3 Potrebni hardverski resursi za implementaciju i performanse *H_DTS* 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_DTS* 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_DTS* 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_DTS1* – arhitektura koja realizuje *DTS* algoritam za formiranje stabla odluke, pri čemu se pozicija instance u odnosu na hiperpovrš računa serijski
2. *H_DTS2* – arhitektura koja realizuje *DTS* algoritam za formiranje stabla odluke, 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}
- 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_DTS* 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_DTS* arhitektura

Arhitektura	Memorija	Sabirači	Množači
H_DTS1	$N_{ts} \cdot (n+1) x N_a$, $2 \cdot (n+1) x N_c$, $(4N_{ts} + 2N_{cl}) x \lceil ld(N_{ts}) \rceil$	1	1
H_DTS2	$N_{ts} \cdot (n+1) x N_a$, $2 \cdot (n+1) x N_c$, $(4N_{ts} + 2N_{cl}) x \lceil ld(N_{ts}) \rceil$	$n-1$	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_DTS* arhitektura može se zaključiti da *H_DTS1* i *H_DTS2* arhitekture imaju identične memorijske zahteve, ali da se broj neophodnih sabirača i množača drastično razlikuje. Kao što se moglo i očekivati, *H_DTS2* arhitektura koja koristi paralelni način evaluacije pozicije instance u odnosu na hiperravan zahteva značajno veći broj množača i sabirača u poređenju sa *H_DTS1* arhitekturom. *H_DTS1* arhitektura ima izuzetno dobru osobinu da je broj neophodnih sabirača i množača nezavisan od bilo kojeg parametra problema. Prilikom korišćenja *H_DTS1* 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_DTS* 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_DTS* arhitektura

Arhitektura	Vreme formiranja (najgori slučaj)
H_DTS1	$\left[\sum_{i=1}^{N_{dt}} \left[(2n+2) \cdot N_{ts_i} + 3n + 8 + N_{HB} \left[(n+2) \cdot N_{ts_i} + N_{cl} + n + 6 \right] \right] + M \right] \cdot CP$
H_DTS2	$\left[\sum_{i=1}^{N_{dt}} \left[2N_{ts_i} + 3n + 12 + ld(n) + N_{HB} \left[2N_{ts_i} + N_{cl} + n + 7 + ld(n) \right] \right] + M \right] \cdot CP$

U tabeli 3.2 prikazani su egzaktni izrazi za vreme potrebno za formiranje stabla odluke ili ansambla stabala odluke. Interesantno je uporediti ove izraze sa izrazima za vremensku kompleksnost *DTL* algoritma izloženim u glavi 2. U slučaju *H_DTS1* arhitekture izraz iz tabele 3.2 može se transformisati na sledeći način.

$$
\begin{aligned}
T_{H\_DTS1} &= O\left(\left[\sum_{i=1}^{N_{dt}}\left[(2n+2)\cdot N_{ts_i}+3n+8+N_{HB}\left[(n+2)\cdot N_{ts_i}+N_{cl}+n+6\right]\right]+M\right]\cdot CP\right)= \\
&= O\left(\sum_{i=1}^{N_{ts}}\left[(2n+2)\cdot i+3n+8+N_{HB}\left[(n+2)\cdot i+N_{cl}+n+6\right]\right]\right)= \\
&= O\left(N_{HB}(n+2)\cdot\frac{N_{ts}\cdot(N_{ts}+1)}{2}\right)= \\
&= O\left(N_{HB}\cdot N_{ts}^2\cdot n\right)
\end{aligned}
\tag{3.20}
$$

Izraz (3.20) izveden je pod pretpostavkom da razmatramo najgori mogući slučaj formiranja stabla odluke, koji je bio prikazan na slici 3.8. 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. Ukoliko izraz (3.20) uporedimo sa prvim izrazom iz (3.11) možemo videti da su oni jednaki.

U slučaju *H_DTS2* arhitekture sličnim transformacijama može se doći do sledeće procene vremenske kompleksnosti.

$$
\begin{aligned}
T_{H\_DTS2} &= O\left(\left[\sum_{i=1}^{N_{dt}}\begin{bmatrix}2N_{ts_i}+3n+12+ld(n)+\\+N_{HB}\left[2N_{ts_i}+N_{cl}+n+7+ld(n)\right]\end{bmatrix}+M\right]\cdot CP\right)= \\
&= O\left(\sum_{i=1}^{N_{ts}}\left[2i+3n+12+ld(n)+N_{HB}\left[2i+N_{cl}+n+7+ld(n)\right]\right]\right)= \\
&= O\left(N_{HB}\cdot\frac{N_{ts}\cdot(N_{ts}+1)}{2}\right)= \\
&= O\left(N_{HB}\cdot N_{ts}^2\right)
\end{aligned}
\tag{3.21}
$$

Vremenska kompleksnost *H_DTS2* arhitekture ne zavisi od broja atributa klasifikacionog problema koji se rešava, *n*, za razliku od izraza (3.11). Do ovoga poboljšanja vremenske kompleksnosti došlo je stoga što se u slučaju *H_DTS2* arhitekture evaluacija pozicije instance u odnosu na hiperpovrš izračunava u paraleli, a ne sekvencijalno, kako je bilo pretpostavljeno u 3.1.6. 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*“, [38]. 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* 9.1.03 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_DTS1* arhitekturu ima sledeći izgled.

```
entity hdts1 is
   generic (
      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 (7 downto 0);                              -- next node level in the decision
                                                                           tree
    node_coef_adr_o:
        out std_logic_vector (node_coef_mem_bus_size_g - 1 downto 0); -- address of the next coeff
    node_coef_o:
        out std_logic_vector (coef_res_g - 1 downto 0);                 -- next coeff value
    node_rln_adr_o:
        out std_logic_vector (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 (2*node_rln_mem_bus_size_g - 1 downto 0); -- next node address
    node_class_adr_o:
        out std_logic_vector (node_rln_mem_bus_size_g - 1 downto 0);    -- address of the class memory
    node_class_o:
        out std_logic_vector (2*class_res_g - 1 downto 0)               -- possible output class
      );
end entity hdts1;
```

Izgled entity *deklaracije u slučaju* H_DTS1 *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 četiri tabele sadrži informacije o neophodnim hardverskim resursima za jednu *H_DTS* arhitekturu.

Rezultati prikazani u tabelama 4.3-4.6 takođe su prikazani grafički, na slikama 4.35-4-37.

Tabela **Error! No text of specified style in document.**.2 Neophodni hardverski resursi u slučaju FPGA implementacije *H_DTS1* arhitekture za 29 odabranih UCI test problema

Problem	*M1 (kbits)*	*M2 (kbits)*	*M3 (kbits)*	*Sabirači*	*Množači*
ausc	101.07	0.29	26.99	1.00	1.00
bc	30.52	0.10	24.47	1.00	1.00
bsc	68.26	0.20	27.34	1.00	1.00
bcw	25.78	0.20	9.32	1.00	1.00
car	118.13	0.14	74.34	1.00	1.00
cmc	130.18	0.20	57.34	1.00	1.00
ger	244.14	0.49	39.10	1.00	1.00
gls	20.90	0.20	6.78	1.00	1.00
hep	30.27	0.39	4.88	1.00	1.00
hrtc	40.61	0.27	10.53	1.00	1.00
hrts	36.91	0.27	9.53	1.00	1.00
Ion	119.97	0.68	12.38	1.00	1.00
irs	7.32	0.10	4.73	1.00	1.00
liv	23.58	0.14	12.16	1.00	1.00
lym	27.46	0.37	4.69	1.00	1.00
page	582.98	0.21	275.72	1.00	1.00
pid	67.50	0.18	30.04	1.00	1.00
son	123.91	1.19	6.53	1.00	1.00

thy	12.60	0.12	6.77	1.00	1.00
ttt	93.55	0.20	37.46	1.00	1.00
veh	156.97	0.37	33.13	1.00	1.00
vote	38.52	0.33	7.28	1.00	1.00
vow	135.35	0.27	38.89	1.00	1.00
w21	1074.22	0.43	253.98	1.00	1.00
w40	2001.95	0.80	253.98	1.00	1.00
wdbc	172.26	0.61	22.27	1.00	1.00
wine	24.34	0.27	5.61	1.00	1.00
wpbc	64.41	0.66	6.09	1.00	1.00
zoo	17.75	0.35	2.86	1.00	1.00

Tabela **Error! No text of specified style in document.**.3 Neophodni hardverski resursi u slučaju FPGA implementacije *H_DTS2* arhitekture za 29 odabranih UCI test problema

Problem	***M1* (kbits)**	***M2* (kbits)**	***M3* (kbits)**	***Sabirači***	***Množači***
ausc	101.07	0.29	26.99	13.00	14.00
bc	30.52	0.10	24.47	3.00	4.00
bsc	68.26	0.20	27.34	8.00	9.00
bcw	25.78	0.20	9.32	8.00	9.00
car	118.13	0.14	74.34	5.00	6.00
cmc	130.18	0.20	57.34	8.00	9.00
ger	244.14	0.49	39.10	23.00	24.00
gls	20.90	0.20	6.78	8.00	9.00
hep	30.27	0.39	4.88	18.00	19.00
hrtc	40.61	0.27	10.53	12.00	13.00
hrts	36.91	0.27	9.53	12.00	13.00
ion	119.97	0.68	12.38	33.00	34.00
irs	7.32	0.10	4.73	3.00	4.00
liv	23.58	0.14	12.16	5.00	6.00
lym	27.46	0.37	4.69	17.00	18.00
page	582.98	0.21	275.72	9.00	10.00
pid	67.50	0.18	30.04	7.00	8.00
son	123.91	1.19	6.53	59.00	60.00
thy	12.60	0.12	6.77	4.00	5.00
ttt	93.55	0.20	37.46	8.00	9.00
veh	156.97	0.37	33.13	17.00	18.00
vote	38.52	0.33	7.28	15.00	16.00
vow	135.35	0.27	38.89	12.00	13.00
w21	1074.22	0.43	253.98	20.00	21.00
w40	2001.95	0.80	253.98	39.00	40.00
wdbc	172.26	0.61	22.27	29.00	30.00
wine	24.34	0.27	5.61	12.00	13.00
wpbc	64.41	0.66	6.09	32.00	33.00
zoo	17.75	0.35	2.86	16.00	17.00

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_DTS* arhitekture za svaki od 29 razmatranih UCI test problema.

Ukoliko analiziramo količinu neophodne veličine memorijskih resursa najzahtevnije su *H_DTS1* i *H_DTS2* arhitekture za *w40* test problem, koje zahtevaju po 2256.73 kilobita. Ovaj broj je daleko manji od ukupno 18576 kilobita memorije koje nam stoje na raspolaganju u *XC5VSX240T* komponenti.

Naviše sabirača zahteva *H_DTS2* arhitektura za *son* test problem, 59. *XC5VSX240T* komponenta na raspolaganju ima ukupno 1056 hardverskih sabiračkih modula što je više nego dovoljno.

Najveći broj množača potrebno je implementirati u slučaju *H_DTS2* arhitekture za *son* test problem, ukupno 60. Situacija je slična kao i u slučaju sabirača, jer *XC5VSX240T* komponenta na raspolaganju ima ukupno 1056 hardverskih sabiračkih modula.

Već prilikom teorijske analize nagovešteno je da paralelne arhitektura *H_DTS2* zahteva daleko više sabirača i množača od serijskih. Takođe pokazano je da što se tiče potrebnih memorijskih resursa, obe *H_DTS* 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 i *H_DTS1* i *H_DTS2* arhitekture mogu koristiti bez većih problema, čak i u slučaju FPGA komponenti sa skromnijim mogućnostima.

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 *DTS* 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_DTS1* i *H_DTS2,* 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 **Error! No text of specified style in document.**.4 Vreme potrebno za formiranje stabla odluke u slučaju softverske odnosno hardverske implementacije *DTS* algoritma

Test skup	**Individualno stablo odluke**		
	PC ***[s]***	***H_DTS1*** ***[s]***	***H_DTS2*** ***[s]***
ausc	4.13	2.18	0.19
bc	1.58	0.65	0.14
bcw	1.20	0.81	0.09
bsc	1.64	0.53	0.07
car	3.70	2.39	0.35
cmc	17.70	5.83	0.83
ger	10.67	6.12	0.35
gls	2.16	0.55	0.09
hep	0.46	0.11	0.01
hrtc	4.00	0.99	0.13
hrts	1.50	0.59	0.07
ion	2.49	1.64	0.07
irs	0.30	0.10	0.02
liv	2.17	0.68	0.13
lym	1.31	0.36	0.03
page	22.95	19.29	1.87
pid	4.76	2.00	0.27
son	3.32	1.46	0.06
thy	0.28	0.16	0.03
ttt	1.87	1.52	0.18
veh	9.25	4.70	0.35
vote	0.67	0.36	0.03
vow	10.40	4.78	0.53
w21	42.97	35.06	1.84
w40	65.99	62.77	2.05
wdbc	1.87	1.74	0.07
wine	0.43	0.23	0.02

wpbc	2.54	0.88	0.05
zoo	1.32	0.27	0.03

Već na osnovu rezultata prikazanih u tabeli 4.4 može se zaključiti da hardverske implementacije *DTS* 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_DTS1* i *H_DTS2*) u odnosu na softversku implementaciju (PC).

Tabela **Error! No text of specified style in document.**.5 Ubrzanje hardverske u odnosu na softversku implementaciju

Test skup	**Individualno stablo odluke**	
	PC	
	H_DTS1 [s]	***H_DTS2 [s]***
ausc	1.90	22.32
bc	2.44	11.70
bcw	1.48	13.33
bsc	3.09	22.47
car	1.55	10.51
cmc	3.04	21.22
ger	1.74	30.57
gls	3.91	24.83
hep	4.18	46.00
hrtc	4.04	31.01
hrts	2.56	22.06
ion	1.52	35.57
irs	3.03	14.29
liv	3.19	17.22
lym	3.61	39.70
page	1.19	12.29
pid	2.38	17.76
son	2.28	59.29
thy	1.79	10.37
ttt	1.23	10.69
veh	1.97	26.81
vote	1.88	26.80
vow	2.18	19.55
w21	1.23	23.39
w40	1.05	32.24
wdbc	1.08	25.97
wine	1.85	21.50
wpbc	2.89	47.04
zoo	4.89	44.00
AVG	**2.39**	**25.53**

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_DTS1* arhitektura, 122 MHz
    - *H_DTS2* arhitektura, 113 MHz
- učestanost *Pentium D 820* mikroprocesora iznosila je 2.8 GHz,

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:

- 2.39 puta u slučaju korišćenja *H_DTS1* arhitekture,
- 25.53 puta u slučaju korišćenja *H_DTS2* 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_DTS* 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] Levi D., HereBoy: A Fast Evolutionary Algorithm, The Second Nasa/DoD Workshop on Evolvable hardware EH'00, 2000, pp. 17-24
[36] M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator", ACM Trans. Model. Comput. Simul., vol. 8, no. 1, pp. 3-30, 1998.
[37] I.Koren, Computer Arithmetic Algorithms, A K Peters, Ltd., 2002.
[38] 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 14.2.3. 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 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 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 stabala odluka koja u mnogome olakšavaju integraciju i rad sa stablima odluka prilikom projektovanja *embedded* sistema.

Tehničke karakteristike:
IP jezgra za hardversko generisanje 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 stabala odluka (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 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 generisanje ortogonalnih, neortogonalnih i nelinearnih stabala odluka. Analizom postojećih rešenja utvrđeno je da u dostupnoj literaturi ne postoji ni jedan rad koji predlaže rešenje navedenog problema.

Predloženo rešenje bazira se na hardverskoj implementaciji DTS algoritma za evolutivno generisanje ortogonalnih, neortogonalnih i nelinearnih stabala odluka koji se bazira na korišćenju HereBoy evlutivnog algoritma u svojoj osnovi. DTS algoritam takođe je rezultat rada autora predloženog tehničkog rešenja. Glavna karakteristika DTS algoritma za formiranje neortogonalnih stabala odluka jeste da se 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 prilikom hardverske implementacije i rezultuje u znatno efikasnijoj arhitekturi u terminima potrebnih hardverskih resursa.

H_DTS arhitektura se sastoji od pet glavnih modula: memorije za smeštanje instanci trening skupa, modula za evaluaciju instance, memorije za smeštanje hromozoma, memorija za čuvanje indeksa instanci i brojača kao i kontrolne jedinice.

Tehničko rešenje predlaže dve arhitekture za hardversko generisanje stabala odluka:

1. H_DTS1 arhitekturu koja realizuje DTS algoritam za formiranje stabla odluke, pri čemu se pozicija instance u odnosu na hiperpovrš računa serijski i
2. H_DTS2 arhitekturu koja realizuje DTS algoritam za formiranje stabla odluke, 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 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

Softver:
IP jezgra za hardversko generisanje 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 stabala odluka koja u mnogome olakšavaju integraciju i rad sa stablima odluka prilikom projektovanja *embedded* sistema.

Tehničke karakteristike:
IP jezgra za hardversko generisanje 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 stabala odluka (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 stabala odluka" autora doc. dr Rastislava Struharika, i prof. dr Ladislava Novaka, realizovano 2011.-2012. godine, prikazano je na 45 stranica 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 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 stabala odluka", predstavlja efikasan alat za rešavanje problema u oblasti hardverske implementacije jednog algoritma za formiranje ortogonalnih, neortogonalnih i nelinearnih 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 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

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