SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 6296

Primjena sustava LCS na klasifikacijske probleme

Matija Bertović

Zagreb, lipanj 2019.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Kako biste uklonili ovu stranicu, obrišite naredbu `\izvornik`.

SADRŽAJ

1. Uvod

Ljudi često prilikom zaključivanja i rješavanja problema koriste znanje već stečeno susrećući se s jednostavnijim problemima unutar sličnog područja. Zašto ne pokušati napraviti sustav temeljen na tehnikama umjetne inteligencije koji ima iste sposobnosti? U tom slučaju, sustav ne bi svaki problem morao učiti ispočetka, nego bi već sadržavao neko znanje sakupljeno tijekom učenja na manjim problemima. Na taj bismo način mogli ubrzati vrijeme potrebno za učenje većih problema. Potrebno je pronaći sustav koji bi mogao imati tu sposobnost. Sustav bi trebao moći prikupiti i spremiti znanje naučeno na nekom problemu. Prikupljeno znanje kasnije bi trebalo moći biti iskorišteno prilikom učenja zahtjevnijih problema unutar slične domene.

Sustav koji može omogućiti navedenu funkcionalnost i koji je detaljnije analiziran u ovom radu naziva se sustav LCS (engl. *Learning Classifier System*). Sustav se sastoji od većeg broja pravila, u kojima je sadržano znanje koje on posjeduje. To znanje kasnije će biti izvučeno i korišteno pri stvaranju novih pravila.

Klasični sustav LCS nema mogućnost iskorištavanja već prikupljenog znanja, nego svaki problem mora učiti od nule, stoga je potrebna promjena koja će to omogućiti. Sustav je nadograđen, te umjesto klasične ternarne abecede sadrži binarna stabla, slična onima generiranima tehnikama *genetskog programiranja*.

Sustav je testiran na 4 različita problema booleovih funkcija: problem multipleksora, problem većinskog bita, problem parnog pariteta i problem bita prijenosa. Prilikom implementacije, korištena je ideja Willsonovog sustava XCS temeljenog na preciznosti (Wilson, 1995).

U poglavlju 2 dan je kratak pregled područja. Poglavlje 3 daje detaljniji opis gore navedenih problema koji su rješavani u ovom radu. Način rada sustava i detaljan opis korištenih algoritama opisan je u poglavlju 4. Rezultati dobiveni primjenom razvijenog sustava opširno su prikazani grafovima i komentirani u poglavlju 5. Rad je zaključen i dan je prijedlog budućeg razvoja u poglavlju 6.

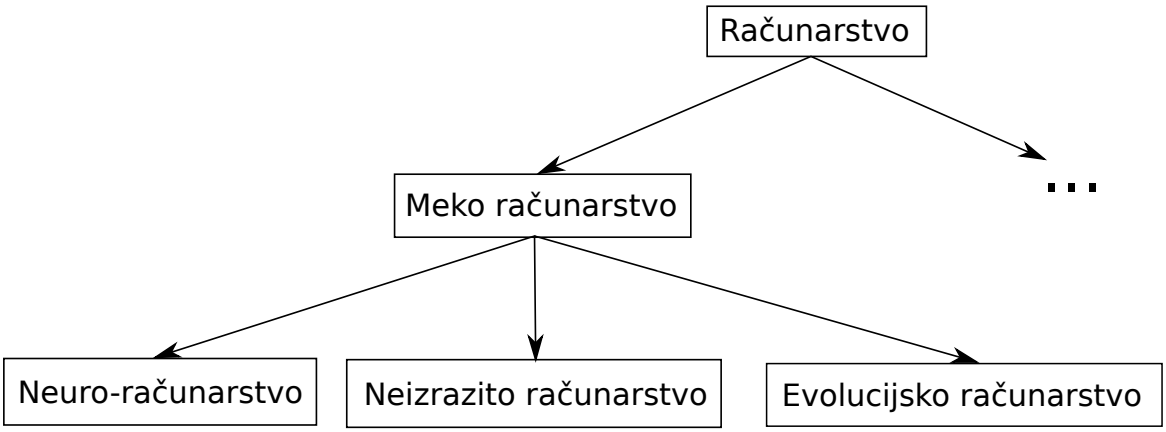
2. Pregled područja

Sustavi LCS su sustavi temeljeni na pravilima, koji povezuju *evolucijsko računarstvo* i *strojno učenje* kako bi se što bolje prilagodili rješavanju danog problema. Sustav je u međudjelovanju s nepoznatom okolinom (engl. *environment*), od koje pomoću osjetila (engl. *sensors*) dobiva ulaz, a preko djelovatelja (engl. *effectors*) nad njom izvršava određenu akciju (Iqbal et al., 2014). Sustav se sastoji od skupa određenog broja pravila koja zajedno rješavaju neki problem. Bitno je naglasiti da poanta učenja nije pronaći jedno pravilo koje na kraju rješava dani problem, nego razviti cijelu populaciju pravila koja ga međusobno rješavaju. Pravila su najčešće u obliku "AKO *uvjet* ONDA *akcija*". Uvjet pravila većinom je tvoren od znakova ternarne abecede 0, 1, #. Znakovi 0 i 1 moraju se podudarati s odgovarajućim znakovima ulaza, dok se znak # podudara s bilo kojim znakom. Znak # zovemo simbolom *don't care* (engl. *don't care symbol*)[1] i on omogućava generaliziranje. Problemi su najčešće takvi da je prostor pretraživanja jako velik i nije moguće doslovno naučiti svaki primjer, nego je potrebna sposobnost generaliziranja. Prilikom istraživanja novih pravila koriste se tehnike *evolucijskog računarstva* (engl. *evolutionary computation*).

Položaj *evolucijskog računarstva* unutar *računarstva* prikazan je slikom 2.1. *Evolucijsko računarstvo* jedna je od grana *računarstva* iz skupine *mekog računarstva* (engl. *soft computing*), zajedno s *neuro-računarstvom* i *neizrazitim računarstvom*. Ono se bavi algoritmima pretraživanja temeljenima na prirodnoj selekciji. Cilj je unapređenje početne populacije u kojoj svaka jedinka predstavlja rješenje ili dio rješenja. Jedinke se trebaju moći vrednovati s obzirom na rješavani problem kako bismo ih mogli uspoređivati. Za unapređenje populacije jedinki koriste se tehnike selekcije (engl. *selection*), križanja (engl. *crossover*) i mutacije (engl. *mutation*). Selekcija je postupak odabira roditeljskih jedinki iz populacije. Prilikom odabira roditelja, većinom se u obzir uzima njihova dobrota (jedinke koje imaju veću dobrotu, imaju i veću vjerojatnost odabira za roditeljsku jedinku). Nakon odabira roditelja, dolazi do njihovog križanja te tako nastaju potomci (engl. *offspring*). Križanje je kombiniranje genetskog materijala roditelja čime se stvaraju potomci koji dio svog genetskog materijala naslijede od jednog, a dio od drugog roditelja. Nakon križanja, javlja se mutacija gena potomaka. Svaki gen ima određenu vjerojatnost mutacije. Mutacijom gen može promijeniti

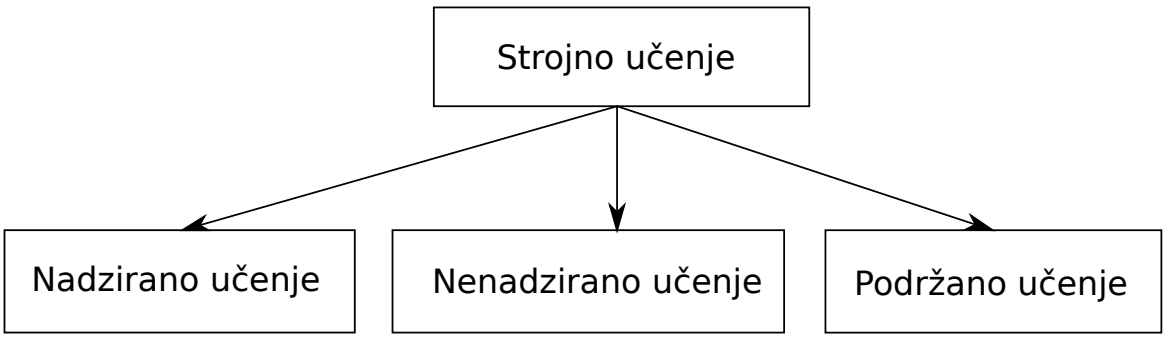
[1] U literaturi se još može naći naziv *wildcard*.

trenutnu vrijednost u neku novu koja ovisi o trenutnoj ili se može generirati potpuno novi gen. Više informacija o *evolucijskom računarstvu* i *genetskim algoritmima* čitatelj može pronaći u (Eiben i Smith, 2015).



Slika 2.1: Položaj *evolucijskog računarstva* unutar *računarstva*.

Prilikom rada sustava LCS, pravila djeluju zajedno, ali neka su *bolja* i imaju veću sposobnost generaliziranja od drugih. Kako bi razlikovali različita pravila i u kojoj mjeri ona utječu na konačni ishod sustava, pravilima se dodjeljuju razni parametri. Ažuriranje tih parametara obavlja se tehnikama *podržanog učenja* (engl. *reinforcement learning*), koje usmjerava potragu za boljim pravilima (Bull, 2004).



Slika 2.2: Položaj *podržanog učenja* unutar *strojnog učenja*.

Podržano učenje grana je *strojnog učenja* (engl. *machine learning*) zajedno za *nadziranim* (engl. *supervised*) i *nenadziranim učenjem* (engl. *unsupervised learning*), kao što je prikazano na slici 2.2. *Podržano učenje* temeljeno je na pokušajima, nakon kojih sustav dobije određenu brojčanu nagradu. Cilj sustava temeljenog na podržanom učenju je maksimizirati nagradu koju sustav dobije izvršavanjem neke akcije. U ovisnosti o dobivenoj nagradi, sustav podešava svoje parametre kako bi s vremenom dobivao sve veću i veću nagradu. Značajka *podržanog učenja* je da sustav ne uči na poznatim točnim ishodima, nego on sam odabire svoju akciju na temelju koje dobiva određenu brojčanu nagradu. Na taj način

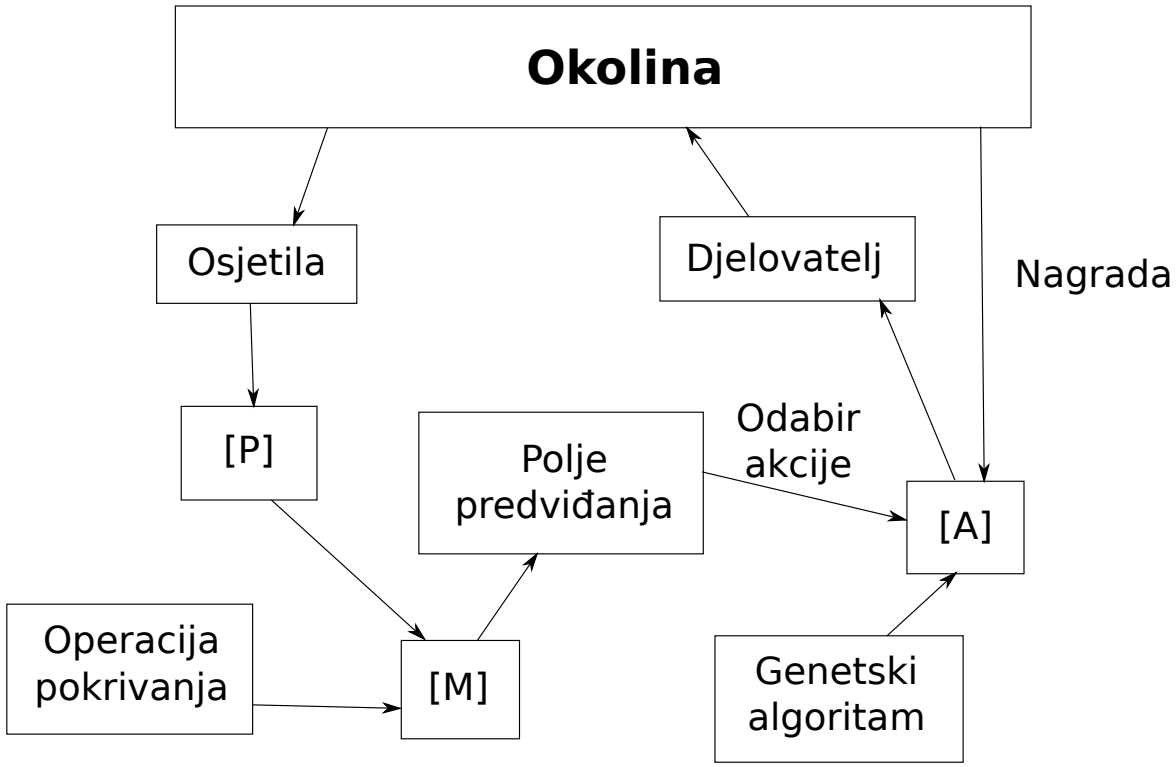
je sustav primoran učiti samo iz svog prethodnog iskustva. Detaljniju analizu *podržanog učenja* čitatelj može pronaći u (Sutton i Barto, 1998).

Dva glavna pristupa u implementaciji sustava LCS su *michiganski stil* (engl. *Michigan-Style*) i *pittsburgski stil Pittsburgh-Style*. *Pittsburgski stil* sustava LCS sadrži skup jedinki od kojih je svaka moguće konačno rješenje sustava. Jedna jedinka je cijeli skup pravila, što znači da je karakteristika *pittsburgskog stila* da sadrži više neovisnih skupova pravila. Iako pravila moraju biti jednake duljine, svaki skup pravila može sadržavati različit broj pravila. *Genetski algoritam* pokreće se nad cijelim skupom jedinki, odnosno na skupom skupova pravila.

U okviru ovog rada obrađen je *michiganski stil* sustava LCS. *Michiganski stil* sustava LCS okarakteriziran je jednim skupom pravila. Svako pravilo predstavlja jednu jedinku. Jedinka u *michiganskom stilu* sustava LCS ne predstavlja konačno rješenje sustava, nego je konačno rješenje cijeli skup jedinki, odnosno jedinke djeluju zajedno pri rješavanju problema. *Michiganski stil* sustava LCS prvi je formalizirao John Holland 1976. godine, a u suradnji s Judith Reitman 1978. dao njegovu implementaciju. S obzirom na složenost originalnog LCS sustava, malo jednostavniju i razumljiviju inačicu dao je Stewart W. Wilson pod nazivom ZCS (engl. *"zeroth-level" classifier system*). Nakon toga, Wilson je uveo još jednu inačicu sustava LCS pod nazivom XCS, u kojemu je promijenio način na koji se računa $dobrota$ pojedinih pravila. U sklopu ovog rada obrađen je sustav XCS, koji je detaljno objašnjen u poglavljima koja slijede, a za detaljniji opis prvotnog sustava LCS i sustava ZCS čitatelj se upućuje na (Bull, 2004).

Ilustracija međudjelovanja komponenti sustava XCS prikazana je na slici 2.3. Prilikom rada sustava izmjenjuju se načini rada *istraži* (engl. *explore*) i *iskoristi* (engl. *exploit*). U načinu rada *istraži*, sustav se prilagođava zadanom problemu mijenjanjem svoje konfiguracije, a u načinu rada *iskoristi* sustav se ne mijenja, nego se na temelju trenutne konfiguracije predviđa akcija koju treba izvršiti.

Prilikom načina rada *istraži* sustav najprije od nepoznate okoline preko osjetila dobije ulazni podatak. Po primitku podatka, skeniraju se sva pravila unutar populacije pravila [P]. Od svih pravila iz [P], od onih koja odgovaraju dobivenom ulaznom podatku tvori se podudarni skup (engl. *match set*) [M]. Nakon formiranja podudarnog skupa [M], u slučaju da on ne sadrži barem jedno pravilo za svaku moguću akciju, pokreće se operacija pokrivanja (engl. *covering operation*). Operacija pokrivanja generira proizvoljno pravilo koje odgovara zadanom ulazu i traženoj akciji te ga dodaje u populaciju. Nakon što je formiran konačni podudarni skup, tvori se polje predviđanja (engl. *prediction array*). Polje predviđanja za svaku moguću akciju sadrži podatak o tome kolika se prosječna nagrada očekuje izvršavanjem te akcije. Na temelju polja predviđanja izabire se konačna akcija koju će sustav izvršiti za dani ulazni podatak. Odabirom konačne akcije, skenira se podudarni skup i tvori se akcijski skup



Slika 2.3: Ilustracija sustava XCS.

(engl. *action set*) [*A*]. Akcijski skup sastoji se od svih pravila sadržanih u podudarnom skupu koja zagovaraju odabranu konačnu akciju. Po formiranju akcijskog skupa, nad okolinom se, preko djelovatelja, izvršava odabrana akcija. Nakon toga, okolina u ovisnosti o izvršenoj akciji sustavu dodjeljuje određenu numeričku nagradu (engl. *reward*). Što je sustav izvršio bolju akciju u ovisnosti o ulaznom podatku, to je nagrada veća i obrnuto. Cilj sustava je maksimizirati nagradu koju dobije od okoline. Na temelju dobivene nagrade, sustav ažurira određene parametre svih pravila unutar akcijskog skupa, u svrhu povećanja buduće nagrade. Nakon ažuriranja parametara, ako je zadovoljen određeni uvjet, uključuje se istraživačka komponenta koja primjenom *genetskog algoritma* istražuje nova pravila. Operacija istraživanja novih pravila djeluje isključivo nad akcijskim skupom.

Prilikom načina rada *iskoristi* isključuju se sve istraživačke komponente. Sustav preko osjetila od okoline dobije ulazni podatak. Po primitku podatka tvori se podudarni skup sa svim pravilima iz populacije koja odgovaraju dobivenom ulaznom podatku. Nakon toga, tvori se polje predviđanja, kao i u načinu rada *istraži*. Sada se na temelju polja predviđanja izabire ona akcija čijim izvršavanjem se očekuje najveća nagrada okoline. Trenutna iteracija ovdje završava te se čeka sljedeći podatak na ulazu.

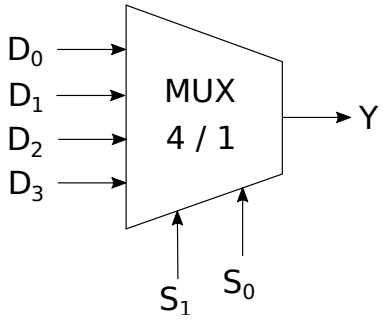
Ovaj cijeli postupak detaljno je opisan u poglavlju 4 gdje su točno opisani i korišteni pripadni algoritmi.

3. Opis problema

Problemi testirani u ovom radu su:

1. multipleksor (engl. *multiplexer*),
2. paritetni bit (engl. *parity*),
3. najzastupljeniji bit (engl. *majority-on*),
4. bit prijenosa binarnog zbrajala (engl. *carry*).

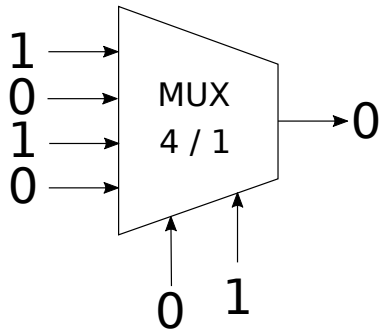
Najviše pažnje posvećeno je rješavanju multipleksora. Na slici 3.1 prikazan je multipleksor 4/1. Multipleksor 4/1 ima 2 upravljačka bita i 4 ulazna bita te jedan izlazni bit. Upravljački bitovi određuju točno jedan ulazni bit koji se dalje prosljeđuje na izlaz, kako je prikazano u tablici 3.1. Općenito, multipleksor koji ima k upravljačkih bitova funkcionira



Slika 3.1: Multipleksor 4/1.

na isti način. k upravljačkih bitova može adresirati 2^k ulaznih bitova, stoga je multipleksor s k upravljačkih bitova funkcija od $k + 2^k$ varijabli. Izlaz iz multipleksora je uvijek točno jedan bit, koji ima vrijednost 0 ili 1. Na slici 3.2 dan je konkretan primjer multipleksora 4/1. Upravljači bitovi ovdje su 01, a ulazni 1010. Upravljački bitovi adresiraju drugi ulaz u multipleksor, stoga se on propušta na izlaz. Na izlazu vidimo vrijednost 0, koja odgovara drugom ulazu. U implementaciji, ovaj primjer bi na ulaz bio dan kao niz bitova 011010. Ulaz se prvo sastoji od upravljačih bitova, a zatim ulaznih. Očekivani izlaz bi u ovom slučaju bio 0.

Problem paritetnog bita svodi se na ispitivanje broja pojavljivanja bita 1 unutar ulaza. Sklop za određivanje paritetnog bita za parni paritet na ulaz prima n-bitni binarni broj te



Slika 3.2: Konkretan primjer 4/1 multipleksora.

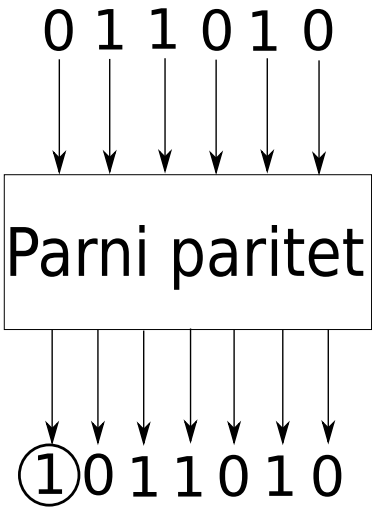
Tablica 3.1: Opis rada multipleksora 4/1.

S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

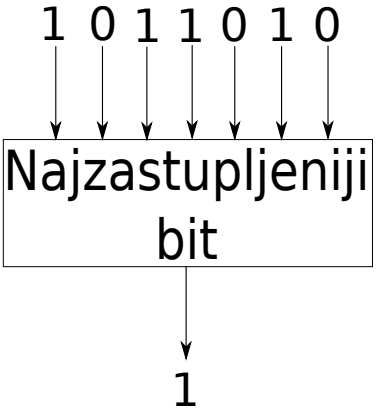
vraća broj s ulaza proširen jednim dodatnim bitom, koji osigurava da je broj pojavljivanja bita 1 paran, odnosno osigurava parni paritet. Dodani bit naziva se paritetni bit. Ako je broj pojavljivanja bita 1 na ulazu paran, broj se treba proširiti bitom 0, a u suprotnom bitom 1. Primjer jednog takvog sklopa koji radi za 6-bitne binarne brojeve prikazan je na slici 3.3. Prikazani sklop na ulaz prima 6-bitni binarni broj 011010. Ulazni podatak ima neparan broj (3) jedinica, dakle treba ga proširiti bitom 1. Sklop na izlazu vraća 7-bitni binarni broj **1**011010 koji ima paran broj (4) jedinica. Prilikom rješavanja ovog problema, razvijeni sustav određuje samo prikazani paritetni bit za parni paritet, a ne cijeli novi $(n + 1)$-bitni broj.

Prilikom problema najzastupljenijeg bita, ispituje se frekvencija pojavljivanja pojedinih bitova. Sklop za određivanje najzastupljenijeg bita na ulaz prima n-bitni binarni broj i na izlaz vraća samo jedan bit, bit koji se najviše puta pojavio u ulaznom binarnom broju. Primjer jednog takvog sklopa koji na ulaz prima 7-bitni binarni broj prikazan je na slici 3.4. Ulaz u sklop je binarni broj 1011010. Ulazni broj ima 4 bita 1 i 3 bita 0. S obzirom da je broj pojavljivanja bita 1 veći od broja pojavljivanja bita 0, na izlaz se vraća bit 1. Prilikom rješavanja problema najzastupljenijeg bita, ulazni brojevi uvijek su neparne duljine, kako bi se izbjeglo pojavljivanje jednakog broja bitova 0 i bitova 1.

Na slici 3.5 dan je primjer rada 7-bitnog binarnog zbrajala. Višebitno binarno zbrajalo na ulaze prima bitove oba pribrojnika. Bitovi pribrojnika šalju se u grupama prema težini. Iz slike je vidljivo da se 3-bitno binarno zbrajalo sastoji od 3 komponente. Općenito, n-bitno binarno zbrajalo sastoji se od n komponenti. Komponenta označena slovom H je

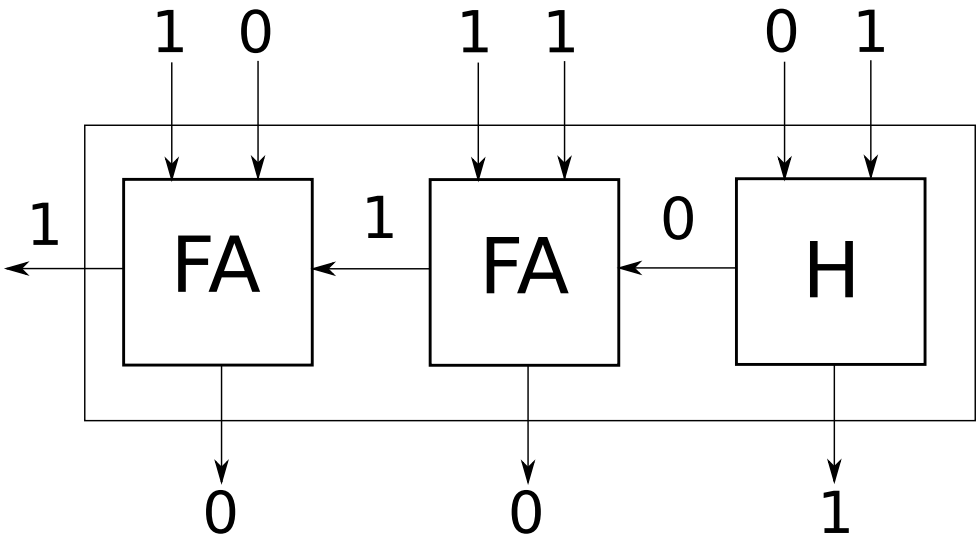


Slika 3.3: Primjer sklopa za određivanje paritetnog bita na 6-bitnom binarnom broju.



Slika 3.4: Primjer sklopa za određivanje najzastupljenijeg bita na 7-bitnom binarnom broju.

poluzbrajalo (engl. *half-adder*). Poluzbrajalo na ulaze prima dva bita, na jedan izlaz vraća njihov zbroj (označen strelicom prema dolje), a na drugi izlaz vraća bit prijenosa (označen strelicom prema lijevo). Komponente označene slovima FA su potpuna zbrajala (engl. *full-adder*). Potpuno zbrajalo na ulaze prima 3 bita, a kao i poluzbrajalo, na izlaze vraća zbroj tih bitova i bit prijenosa. U prikazanom primjeru, prvi binarni broj je 110, a drugi binarni broj 011. Rezultat zbrajanja je binarni broj 001, a konačni bit prijenosa, koji predstavlja izlaz iz 3-bitnog zbrajala je 1. U implementaciji je ulaz u n-bitno binarno zbrajalo prikazan redom bitovima prvog pa drugog pribrojnika i on je veličine $2n$, a izlaz je konačni bit prijenosa. Testirajući ovaj primjer, podatak dobiven iz okoline bio bi 110011, a očekivani izlaz bio bi 1. Ulazni podatak uvijek će biti parne duljine $2n$, kako bi se on mogao interpretirati kao 2 n-bitna binarna broja.



Slika 3.5: Primjer višebitnog binarnog zbrajala za zbrajanje 3-bitnih binarnih brojeva.

4. Opis algoritama

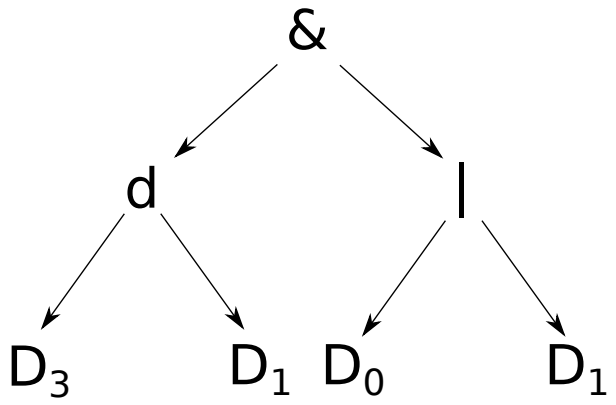
U sklopu razrade ovog sustava, potrebno je ostvariti iskorištavanje znanja već naučenog na jednostavnijim problemima. Klasični bitovi uvjeta u ranije opisanim pravilima nam to onemogućavaju. Iz tog razloga, svaki bit uvjeta zamijenjen je programskim isječkom, koji ovisno o ulazu, vraća 0 ili 1. Takav sustav XCS, koji koristi programske isječke umjesto uvjetnih bitova, naziva se XCSCFC[1]. Na taj način, prilikom stvaranja novih pravila, ona mogu sadržavati programske isječke izvučene iz pravila koja su rješavala jednostavniji problem u istoj domeni. Prilikom preuzimanja programskih isječaka, u obzir dolaze samo precizna i iskusna pravila čija dobrota (engl. *fitness*) je veća od prosječne unutar te populacije pravila (Iqbal et al., 2014).

U ovom radu, isječci koda modelirani su binarnim stablima dubine do najviše 2, što znači da je moguće imati najviše 7 čvorova. Skup funkcija koje čvorovi mogu obavljati je $\{AND, OR, NAND, NOR, NOT\}$. U primjerima, te su funkcije redom označene s $\&, |, d, r, \sim$. Skup mogućih završnih (terminalnih) čvorova pojedinog binarnog stabla je $\{D_0, D_1, ..., D_{n-1}\}$, gdje *n* predstavlja duljinu ulaza dobivenog od okoline. Svaki završni čvor predstavlja točno jedan bit ulaza. Svaki programski isječak na ulaz dobiva cijeli ulaz dobiven od okoline, a na izlazu vraća rezultat operacija koje se nalaze u čvorovima stabla. Na slici 4.1 prikazan je primjer jednog takvog binarnog stabla koje vraća rezultat operacije $D_3D_1dD_0D_1|\&$. Operacija je zbog jednostavnosti prikazana u *postfiks* obliku. Iz slike je vidljivo da se u binarnom stablu ne moraju pojavljivati svi bitovi ulaza, a također i da se pojedini bitovi mogu pojavljivati više puta.

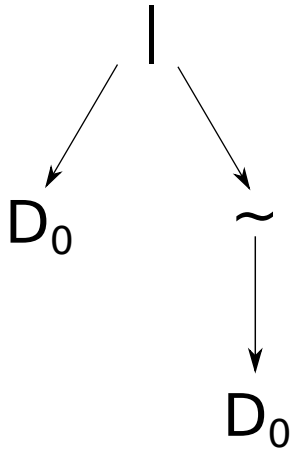
Potreban je i programski isječak koji označava simbol *don't care*, a koji za svaki niz bitova koje dobije na ulazu, na izlazu vraća 1. On je prikazan na slici 4.2 i označava operaciju $D_0D_0 \sim |$, a preuzet je iz (Iqbal et al., 2014).

Prije detaljnog objašnjenja korištenih algoritama, potrebno je naglasiti da su u implementaciji korišteni makroklasifikatori (engl. *macroclassifiers*) (Wilson, 1995). Makroklasifikatori su pravila koji imaju dodatan parametar brojnosti (engl. *numerosity*). Razlog korištenju makroklasifikatora je bolja vremenska i prostorna složenost sustava. Naime, pravila koja imaju jednaki uvjet i akciju sadržana su u istom pravilu, ali je njegova brojnost u tom slučaju

[1] Kratica dolazi od engl. *XCS with code-fragment conditions*.



Slika 4.1: Primjer programskog isječka prikazanog binarnim stablom.



Slika 4.2: Isječak koda korišten kao simbol *don't care*.

veća od 1. Brojnost pravila cl sadržana je u varijabli $cl.n$. Ako u populaciji postoji n pravila koji imaju jednak uvjet i akciju, to je zapisano u jednom pravilu čija je brojnost u tom slučaju postavljena na n. Kod korištenja makroklasifikatora, prilikom svih izračuna potrebno je u obzir uzeti i parametar brojnosti.

U načinu rada *istraži*, na početku se od okoline dobije ulazni podatak s. U ovisnosti o s, formira se *podudarni skup* [*M*]. [*M*] se sastoji od svih pravila iz populacije [*P*] koji odgovaraju ulazu s.

Za pravilo cl[2] kažemo da odgovara ulazu s, ako svaki programski isječak za zadani ulaz s na izlazu daje 1. S obzirom da svaki programski isječak unutar uvjeta pravila na ulaz dobiva cijeli s, poredak programskih isječaka unutar pravila uopće nije bitan. Algoritam 1 prikazuje postupak evaluacije pravila cl u odnosu na ulaz s. Pri tome cf označava *i*-ti programski isječak unutar zadanog pravila cl, a val rezultat programskog isječka cf s obzirom na zadani ulaz s. n je duljina uvjeta unutar pravila, odnosno broj programskih isječaka koji se nalaze

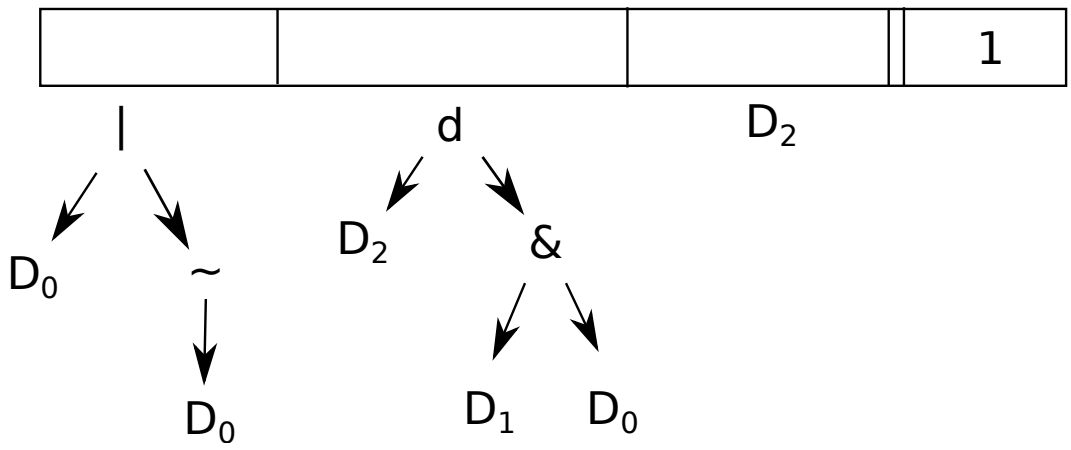
[2]Kratica dolazi od engl. *classifier*. Naime, u ovom je slučaju cijeli sustav jedan klasifikator koji raspoređuje podatke u razrede, ali i za svako pravilo se može reći da je jedan *mali* klasifikator. S obzirom da samo pravilo pojedinačno ne radi ispravnu klasifikaciju, ovdje ga ispravine zovemo *pravilo*, a ne *klasifikator*.

u pravilu. $cl.cond$ je polje svih programskih isječaka. Algoritam vraća vrijednost $true$ ako svaki programski isječak za zadani ulaz s vraća vrijednost 1, a $false$ inače.

Algoritam 1 Evaluiranje pravila cl u odnosu na ulaz s

Ulaz: cl – pravilo, s – stanje okoline.
Izlaz: odgovara li pravilo cl stanju s
for $(i := 0; i < n; i := i + 1)$ **do**
  $cf := cl.cond[i]$
  $val := evaluiraj(cf, s)$
  if $val \neq 1$ **then**
    return $false$
  end if
end for
return $true$

Primjer izgleda jednog opisanog pravila prikazan je na slici 4.3. Vidimo da zadano pravilo ima 3 uvjetna bita i jedan simbol *don't care*. Prikazano pravilo određuje akciju 1. Pretpostavimo da iz okoline dobijemo ulaz $D_0D_1D_2 = 011$. Ulaz se za zadano pravilo evaluira za svaki uvjetni bit pravila. Da bi se ulaz i pravilo mogli podudarati, svaki programski isječak uvjeta pravila mora vratiti 1, u suprotnom se ulaz i pravilo ne podudaraju. Prvi programski isječak izvodi funkciju $D_0D_0 \sim |$, što bi se na danom primjeru pretvorilo u $00 \sim |$, a to se evaluira u 1. Ovo je ujedno i simbol *don't care* koji se uvijek evaluira u 1. Drugi programski isječak izvodi funkciju $D_2D_1D_0\&d$, što se pretvara u $110\&d$, a evaluira u 1. Posljednji programski isječak izvodi funkciju D_2, koja se pretvara u 1 i evaluira u 1. Pokazano je da se sva tri programska isječka za ovaj ulaz evaluiraju u 1, stoga bismo mogli reći da se ovo pravilo i ulaz podudaraju.



Slika 4.3: Primjer izgleda jednog konkretnog pravila s 3 uvjetna bita.

Nakon formiranja [*M*], provjerava se sadrži li [*M*] sve moguće akcije a. U slučaju da

za neku akciju ne postoji pripadno pravilo, pokreće se operacija pokrivanja (engl. *covering operation*), prikazana algoritmom 2. U operaciji pokrivanja stvara se novo pravilo čiji je svaki programski isječak simbol *don't care* s vjerojatnošću $P_{don'tCare}$, a s vjerojatnošću $1 - P_{don'tCare}$ proizvoljno generirani programski isječak, koji s obzirom na stanje s mora vraćati 1. U prikazanom algoritmu, $cl.action$ sadrži akciju koju zagovara pravilo cl. Operacija pokrivanja pokreće se za svaku akciju koja nedostaje u [*M*], a novo pravilo se dodaje u [*P*] i [*M*].

Algoritam 2 Operacija pokrivanja

Ulaz: s – ulaz dobiven iz okoline, a – akcija koju se pokriva.
Izlaz: Generirano novo pravilo cl
cl := inicijaliziraj novo pravilo
for $(i := 0; i < n; i := i + 1)$ **do**
  r := proizvoljan decimalni broj iz intervala [0, 1)
  if $r < P_{don'tCare}$ **then**
    $cl.cond[i]$:= *don't care* simbol
  else
    repeat
      cf := generiraj proizvoljni programski isječak
      $val := evaluiraj(cf, s)$
    until $val \neq 1$
    $cl.cond[i] := cf$
  end if
end for
$cl.action := a$
return cl

Nakon svakog novog dodavanja pravila u [*P*], pokreće se operacija brisanja (Butz, 2002). S obzirom da je potrebno zadržati maksimalnu veličinu populacije N, ako je trenutna veličina populacije veća od N, izabiru se pravila koja je potrebno izbrisati. Operacija brisanja prikazana je algoritmom 3. Pravilo se za brisanje odabire *proporcionalnom selekcijom* (engl. *Roulette-Wheel selection*), na temelju glasova koje daje svako pravilo. Postupak izračuna glasova prikazan je algoritmom 4. Glas svakog pravila temelji se na prosječnoj veličini akcijskog skupa u kojemu se ono nalazilo. Razlog tomu je pokušaj ostvarenja približno jednakih veličina akcijskih skupova. Također, ako je pravilo dovoljno iskusno, a dobrota pravila je znatno manja od prosječne dobrote unutar populacije, vjerojatnost njegovog izbacivanja se dodatno povećava. Time je osigurano izbacivanje lošijih pravila. Iskustvo pravila cl određeno je brojem pojavljivanja tog pravila unutar akcijskog skupa i pamti se u varijabli

Algoritam 3 Operacija brisanja

Ulaz: $[P]$ – populacija.
Izlaz: -
$velicinaPopulacije := \sum_{cl \in [P]} cl.n$
if $velicinaPopulacije \leq N$ **then**
  return
end if
$prosjecnaDobrota := (\sum_{cl \in [P]} cl.F \cdot cl.n)/velicinaPopulacije$
$sumaGlasova := 0$
for (pravilo cl iz $[P]$) **do**
  $sumaGlasova := sumaGlasova + glas(cl, prosjecnaDobrota)$
end for
r := proizvoljan decimalni broj iz intervala [0, 1)
$odabir := r \cdot sumaGlasova$
$sumaGlasova := 0$
for (pravilo cl iz $[P]$) **do**
  $sumaGlasova := sumaGlasova + glas(cl, prosjecnaDobrota)$
  if $odabir < sumaGlasova$ **then**
    if $cl.n > 1$ **then**
      $cl.n := cl.n - 1$
    else
      izbaci pravilo cl iz populacije $[P]$
    end if
    return
  end if
end for

$cl.exp$. Konstanta θ_{del} određuje granicu iskustva nakon koje se može reći da je pravilo dovoljno iskusno za brisanje. Konstanta δ ($0 < \delta \leq 1$) određuje minimalni postotak prosječne dobrote populacije koju pravilo mora imati da se njegov glas ne bi dodatno povećao. Vjerojatnost odabira svakog pravila prilikom *proporcionalne selekcije* jednaka je postotku glasa tog pravila u odnosu na ukupnu sumu glasova. Nakon što je pravilo cl izabrano za brisanje, provjerava se njegova brojnost sadržana u $cl.n$. Ako je njegova brojnost veća od 1, brojnost se umanjuje za 1 i pravilo se ne izbacuje iz populacije. U suprotnom, ako je brojnost 1, pravilo se izbacuje iz populacije.

Algoritam 4 Glas

Ulaz: cl – pravilo čiji glas računamo, $prosjecnaDobrota$ – prosječna dobrota populacije.

Izlaz: $glas$ – glas pravila cl
$vote := cl.as \cdot cl.n$
if $cl.exp > \theta_{del}$ and $cl.F/cl.n < \delta \cdot prosjecnaDobrota$ **then**
  $vote := vote \cdot prosjecnaDobrota/(cl.F/cl.n)$
end if
return $glas$

Nakon formiranja podudarnog skupa [*M*], potrebno je odrediti akciju koju će sustav izvršiti. Akcija se određuje na temelju pravila sadržanih u [*M*]. Za svaku akciju, potrebno je izračunati srednju vrijednost nagrade koju sustav očekuje izvršavanjem te akcije. Ona se označava funkcijom $P(a)$ i računa po formuli (4.1).

$$P(a) = \frac{\sum_{cl \in [M] \wedge cl.a=a} cl.p \cdot cl.F \cdot cl.n}{\sum_{cl \in [M] \wedge cl.a=a} cl.F \cdot cl.n} \tag{4.1}$$

Vrijednosti $P(a)$ za svaku moguću akciju tvore *polje predviđanja*. Ovisno o vrijednostima $P(a)$, *proporcionalnom selekcijom* se izabire konačna akcija koju sustav izvršava[3]. Vjerojatnost da će akcija a biti izabrana proporcionalna je vrijednosti $P(a)$.

Nakon odabira akcije a, formira se akcijski skup [*A*]. [*A*] se sastoji od svih pravila iz [*M*] koja zagovaraju a. Nakon formiranja akcijskog seta izvršava se odabrana akcija a i u ovisnosti o izvršenoj akciji od okoline stiže nagrada R.

Po dobitku nagrade, dolazi do ažuriranja parametara svih pravila sadržanih u [*A*]. Redom se ažuriraju iskustvo pravila (engl. *experience*) exp, njegovo predviđanje (engl. *prediction*)

[3] Akcija može biti izabrana i nekim drugim postupkom, npr. proizvoljno (engl. *pure exploration*) ili se može odabrati akcija s najvećom $P(a)$ vrijednosti (engl. *pure exploitation*). Također, može se koristiti proizvoljna akcija s određenom vjerojatnošću, a u suprotnom najbolja. Takav postupak odgovarao bi ϵ-*pohlepnom* (engl. ϵ-*greedy*) postupku odabira u podržanom učenju, gdje bi ϵ vrijednost odgovarala vjerojatnosti odabira proizvoljne akcije, kako je navedeno u (Butz, 2002).

p, pogreška u predviđanju (engl. *rediction error*) ϵ, preciznost (engl. *accuracy*) κ, relativna preciznost (engl. *relative accuracy*) κ', dobrota F i prosječna veličina akcijskih skupova koji su sadržavali to pravilo (engl. *action set size*) as. Iskustvo pravila cl, $cl.exp$, je broj pojavljivanja cl u akcijskom skupu i ažurira se po izrazu (4.2).

$$cl.exp := cl.exp + 1 \tag{4.2}$$

Predviđanje pravila, $cl.p$ procjenjuje nagradu koju sustav očekuje podudaranjem pravila i izvođenjem akcije koju ono zagovara. Ažurira se po uzoru na Q-učenje, u ovisnosti o dobivenoj nagradi R, po izrazu (4.3), gdje je β realan broj iz intervala $(0, 1]$ i naziva se stopa učenja (engl. *learning rate*).

$$cl.p := cl.p + \beta \cdot (R - cl.p) \tag{4.3}$$

Pogreška u predviđanju ažurira se u ovisnosti o nagradi R i predviđanju pravila p po izrazu (4.4).

$$cl.\epsilon := cl.\epsilon + \beta \cdot (|R - cl.p| - cl.\epsilon) \tag{4.4}$$

Prije ažuriranja dobrote F pravila, najprije je potrebno izračunati njegovu preciznost κ, te ju zatim normalizirati s obzirom na preciznosti ostalih pravila unutar akcijskog skupa, odnosno izračunati relativnu preciznost κ'.

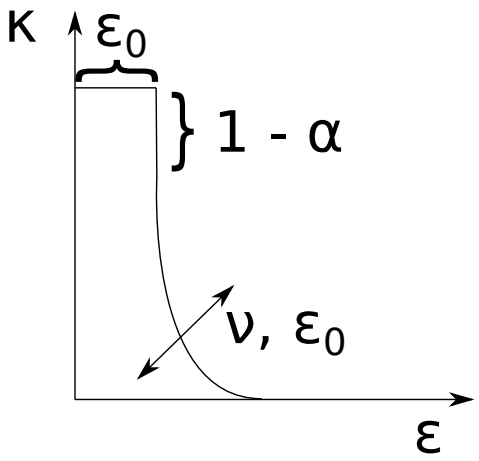
$$cl.\kappa := \begin{cases} 1 & \text{, ako je } \epsilon < \epsilon_0 \\ \alpha \cdot \left(\frac{\epsilon}{\epsilon_0}\right)^{-\nu} & \text{, inače} \end{cases} \tag{4.5}$$

$$cl.\kappa' := \frac{\kappa}{\sum_{c \in [A]} c.\kappa} \tag{4.6}$$

Pri tome, parametri α $(0 < \alpha < 1)$ i ν $(\nu > 0)$ kontroliraju brzinu propadanja preciznosti (Butz et al., 2004). Parametar ϵ_0 određuje granicu do koje najviše može doći pogreška u predviđanju ϵ, a da bi se za pravilo moglo reći da je potpuno precizno. Ako za pravilo cl vrijedi da je $cl.\epsilon < \epsilon_0$, njegova preciznost postaje 1 ($cl.\kappa = 1$), a u suprotnom preciznost ekponencijalno opada ovisno o parametrima α i ν, što je prikazano izrazom (4.5). Nakon što su izračunate preciznosti svih pravila unutar akcijskog seta [A], svaka preciznost se normira ukupnom sumom preciznosti, kao što je prikazano izrazom (4.6). Kada je izračunata relativna preciznost, dobrota se ažurira prema izrazu (4.7).

$$cl.F := cl.F + \beta \cdot (\kappa' - cl.F) \tag{4.7}$$

Dobrota pravila je procjena njegove preciznosti s obzirom na ostala pravila iz [A]. Na slici 4.4 vidi se utjecaj pojedinih parametara prilikom računanja preciznosti κ. Parametar ϵ_0 određuje do koje granice će pravila imati jednaku, maksimalnu, preciznost, parametar α uvodi



Slika 4.4: Funkcija ovisnosti preciznosti κ o pogrešci u predviđanju ϵ. Slika preuzeta iz (Butz et al., 2004)

značajnu razliku između preciznih i manje preciznih pravila, a parametri ν i ponovno ϵ_0 određuju brzinu opadanja preciznosti.

Nakon postavljanja parametara, postoji mogućnost za pokretanjem operacije istraživanja novih pravila (engl. *discovery component*). Svako pravilo dodatno sadrži i parametar koji pamti kada je zadnji put ono sudjelovalo u akcijskom skupu nad kojim se provela operacija istraživanja novih pravila. Operacija otkrivanja novih pravila provodi se ukoliko je prosječno vrijeme proteklo od prošlog pokretanja operacije otkrivanja nad pravilima unutar akcijskog skupa veće od vremena određenog konstantom θ_{GA}. Ukoliko to nije zadovoljeno, ovaj se korak preskače.

Prilikom operacije otkrivanja, najprije se iz akcijskog skupa izabiru dva roditeljska pravila. Odabir roditeljskih pravila ostvaren je turnirskom selekcijom (engl. *tournament selection*) u ovisnosti o parametru dobrote. Prilikom turnirske selekcije, nasumično se odabire unaprijed zadani broj pravila, te se kao pobjednika odabire ono koje od odabranih ima najveću vrijednost dobrote. Od izabranih roditeljskih pravila stvaraju se dva potomka, od kojih prvotno svaki ima iste programske isječke uvjeta kao jedan od roditelja.

Nakon toga, s vjerojatnošću χ provodi se križanje potomaka. Križanje je ostvarenom operacijom križanja u dvije točke (engl. *two-point crossover*) kako je prikazano algoritmom 5. Prilikom križanja u dvije točke, proizvoljno se odabiru dva mjesta unutar uvjeta pravila koja se križaju, te se zamjene svi programski isječci između njih. Unutar algoritma, varijabla n sadrži duljinu uvjeta, odnosno broj programskih isječaka. Programski isječci se ovdje ne mijenjaju, samo se razmjenjuju između potomaka. Također, prilikom operacije križanja, akcije koje pravila zagovaraju se ne mijenjaju.

Nakon toga, nad potomcima se provodi mutacija, u kojoj svaki programski isječak uvjeta ima vjerojatnost mutacije μ. Mutacija, za razliku od križanja, djeluje i na uvjete pravila i na akcije. Tijekom mutacije, svaki simbol *don't care* zamjenjuje se proizvoljno generira-

Algoritam 5 Križanje u dvije točke

Ulaz: cl_1 – prvo pravilo, cl_2 – drugo pravilo.
Izlaz: -
$x :=$ proizvoljan decimalni broj iz intervala [0, n)
$y :=$ proizvoljan decimalni broj iz intervala [0, n)
if $x > y$ **then**
  zamijeni x i y
end if
for $(i = x; i <= y; i = i + 1)$ **do**
  zamijeni $cl_1.cond[i]$ i $cl_2.cond[i]$
end for

nim programskim isječkom koji odgovara stanju s dobivenom iz okoline, a svaki drugi programski isječak zamjenjuje se simbolom *don't care*. Na poslijetku, akcije potomaka također bivaju mutirane s vjerojatnošću μ, pri čemu se akcija mijenja u bilo koju drugu akciju (u ovom radu su jedine moguće akcije 0 ili 1, stoga 0 postaje 1, a 1 postaje 0). Nakon mutacije, mutirani potomak još uvijek odgovara stanju s. Operacija mutacije prikazana je algoritmom 6.

Na kraju operacije istraživanja, predviđanje novonastalih potomaka postavlja se na srednju vrijednost predviđanja roditelja, pogreška u predviđanju postavlja se na srednju vrijednost pogreške u predviđanju roditelja pomnoženu faktorom $predictionErrorReduction$, a dobrota na srednju vrijednost dobrote roditelja pomnoženu faktorom $fitnessReduction$, kao što je navedeno u (Iqbal et al., 2014).

Prije dodavanja nastalih potomaka u populaciju, pokreće se operacija provjere obuhvaća li neki od roditelja potomke (engl. *GA subsumption*). Roditelj obuhvaća potomka, ako uvjet roditelja logički obuhvaća uvjet potomka. Razlog ovoj operaciji je taj što u slučaju da roditelj obuhvaća potomka, dodavanjem potomka u populaciju ne bi se poboljšala sposobnost sustava, jer roditelj sadrži sve informacije koje sadrži i potomak (Butz, 2002). Da bi se uopće mogla pokrenuti provjera, roditelj mora biti precizan i dovoljno iskusan. Konstanta θ_{sub} sadrži donju granicu iskustva pravila da bi se za njega moglo reći da je dovoljno iskusno za ovu operaciju, dok konstanta ϵ_0 sadrži gornju granicu pogreške u predviđanju pravila da bi oni bilo dovoljno precizno. Roditelj može obuhvatiti potomka ako oba pravila zagovaraju istu akciju, ako je roditelj precizan i dovoljno iskusan i ako je roditelj općenitiji od potomka. Uvođenjem programskih isječaka umjesto simbola ternarne abecede u uvjete pravila, maknuta je važnost poretka programskih isječaka unutar uvjeta. Iz tog razloga, prilikom ispitivanja je li roditelj općenitiji od djeteta u obzir su uzeti skupovi programskih odsječaka (Iqbal et al., 2014). Operacija provjere je li jedno pravilo općenitije od drugog prikazana je

Algoritam 6 Mutacija

Ulaz: cl – pravilo nad kojim se provodi mutacija, s – stanje okoline.
Izlaz: -
for $(i = 1; i <= n; i = i + 1)$ **do**
  $r :=$ proizvoljan decimalni broj iz intervala [0, 1)
  if $r < \mu$ **then**
    if $cl.cond[i] =$ simbol *don't care* **then**
      repeat
        cf := generiraj proizvoljni programski isječak
        $val := evaluiraj(cf, s)$
      until $val \neq 1$
      $cl.cond[i] := cf$
    else
      $cl.cond[i]$:= simbol *don't care*
    end if
  end if
end for
$r :=$ proizvoljan decimalni broj iz intervala [0, 1)
if $r < \mu$ **then**
  $cl.action :=$ proizvoljna akcija različita od $cl.action$
end if

algoritmom 7. Da bi pravilo cl_1 bilo općenitije od pravila cl_2, cl_1 mora imati više simbola *don't care* od pravila cl_2, a svaki ostali programski isječak pravila cl_1 mora biti sadržan u cl_2. Ako se pokaže da roditelj obuhvaća potomka, umjesto dodavanja tog potomka u populaciju, roditelju se parametar brojnosti povećava za 1.

Nakon provjere obuhvaća li roditelj potomka, pokreće se provjera obuhvaćanja unutar cijelog akcijskog skupa [*A*]. Pretražuje se akcijski skup i pronalazi se pravilo koje je precizno i dovoljno iskusno, a ima najveći udio simbola *don't care*. Da bi pravilo bilo precizno i dovoljno iskusno za obuhvaćanje drugih pravila, ono mora ispunjavati isti uvjet kao i u prethodnom koraku. Nakon pronalaženja takvog pravila cl, ponovno se prolazi kroz akcijski skup i za svako pravilo c od kojega je cl općenitije, pravilu cl se brojnost povećava za brojnost pravila c, a klasifikator c se briše iz populacije.

Nadalje, prilikom dodavanja novog pravila cl u populaciju, potrebno je proći kroz ostala pravila u populaciji i provjeriti postoji li već pravilo koje je jednako pravilu cl. Ako takvo pravilo postoji, cl se ne dodaje u populaciju, nego se pronađenom pravilu brojnost poveća za 1. Postupak provjere jednakosti pravila također je drugačiji u odnosu na klasični sustav

Algoritam 7 Općenitije pravilo

Ulaz: cl_1 – općenitije pravilo, cl_2 – specifičnije pravilo.
Izlaz: – je li pravilo cl_1 općenitije od cl_2
$x :=$ broj simbola *don't care* u cl_1
$y :=$ broj simbola *don't care* u cl_2
if $x \leq y$ **then**
  return $false$
end if
$X :=$ skup svih "ne-*don't care*" isječaka u cl_1
$Y :=$ skup svih "ne-*don't care*" isječaka u cl_2
if $X \not\subseteq Y$ **then**
  return $false$
end if
return $true$

XCS. Ponovno nije bitno da poredak programskih isječaka u pravilima bude jednak, nego da pravila sadrže jednake isječke, neovisno o poziciji. Postupak provjere jednakosti pravila prikazan je algoritmom 8. Da bi pravila bila jednaka, ona moraju zagovarati istu akciju, moraju imati jednak broj simbola *don't care*, a skupovi ostalih programskih isječaka tih pravila moraju biti jednaki.

Prilikom generiranja proizvoljnih programskih isječaka, kao listovi unutar binarnog stabla, osim terminalnih čvorova, koriste se i programski isječci naučeni na jednostavnijim problemima unutar iste domene (Iqbal et al., 2014). Po završetku učenja određenog problema, sakupljeni su programski isječci sadržani unutar preciznih i iskusnih pravila konačne populacije. Pravilo je precizno i dovoljno iskusno za korištenje pri učenju težeg problema ako je njegovo iskustvo veće od konstante θ_{re} i ako je njegova dobrota veća od prosječne dobrote konačne populacije. Primjer ponovnog korištenja naučenih programskih isječaka prikazan je u tablici 4.1. U tablici je različitim programskim isječcima dano ime zbog jednostavnijeg referenciranja. Vidi se da su isječci naučeni na najjednostavnijem multipleksoru 4/1 korišteni kao listovi unutar isječaka multipleksora 8/1. Jednako tako, u multipleksoru 16/1, korišteni su isječci naučeni i na multipleksoru 4/1 i 8/1.

U načinu rada *iskoristi*, sustav XCS se ne mijenja, nego izvršava najbolju moguću akciju. Početak je isti kao i u načinu rada *istraži*. Formira se podudarni skup koji se sastoji od pravila koja odgovaraju ulazu s. U ovisnosti o pravilima sadržanim u [*M*], formira se *polje predviđanja*, na isti način kako je opisano u načinu rada *istraži*. Na temelju formiranog *polja predviđanja*, odabire se akcija a s najvećom vrijednosti $P(a)$. Ovaj odabir je drugačiji od onog opisanog u načinu rada *istraži*, u kojem se akcija a odabire *proporcionalnom selekcijom*

u ovisnosti o vrijednosti $P(a)$.

Načini rada *istraži* i *iskoristi* se međusobno izmjenjuju, pri čemu način rada *iskoristi* služi testiranju sposobnosti sustava.

Algoritam 8 Jednakost pravila

Ulaz: cl_1 – prvo pravilo, cl_2 – drugo pravilo.
Izlaz: – je li pravilo cl_1 općenitije od cl_2
if $cl_1.action \neq cl_2.action$ **then**
  return $false$
end if
$x :=$ broj simbola *don't care* u cl_1
$y :=$ broj simbola *don't care* u cl_2
if $x \neq y$ **then**
  return $false$
end if
$X :=$ skup svih "ne-*don't care*" isječaka u cl_1
$Y :=$ skup svih "ne-*don't care*" isječaka u cl_2
if $X \neq Y$ **then**
  return $false$
end if
return $true$

Tablica 4.1: Ponovno korištenje naučenog znanja. Tablica preuzeta iz (Iqbal et al., 2014).

Multipleksor	Programski isječak	
	Ime	Izraz
MUX 4/1	L1_0	$D_1 D_0 D_4 dr$
	L1_1	$D_5 \sim D_1 D_0 \&\&$
	...	...
MUX 8/1	L2_0	$L1\_15 D_2 L1\_4 r\&$
	L2_1	$L1\_5 D_7 \mid L1\_11 D_3 \& r$
	...	...
MUX 16/1	L3_0	$L2\_9 L1\_7 D_{11} \mid r$
	L3_1	$L1\_10 D_{17} \mid L2\_1 D_0 r\&$
	...	...

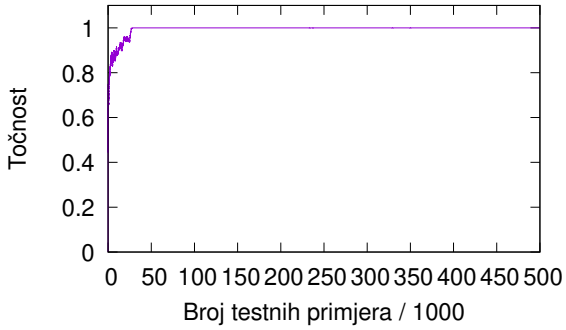
5. Rezultati

Prilikom testiranja rada sustava, korištene vrijednosti parametara sustava su podešene kao u (Iqbal et al., 2014). Stopa učenja $\beta = 0.2$, stopa propadanja dobrote $\alpha = 0.1$, granica pogreške predviđanja $\epsilon_0 = 10$, eksponent propadanja dobrote $\nu = 5$, granica prosječnog proteklog vremena prilikom istraživanja novih pravila $\theta_{GA} = 25$, vjerojatnost križanja u dvije točke $\chi = 0.8$, vjerojatnost mutacije μ, granica iskustva pri brisanju pravila $\theta_{del} = 20$, postotak srednje vrijednosti dobrote prilikom brisanja $\delta = 0.1$, granica iskustva prilikom obuhvaćanja $\theta_{sub} = 20$, vjerojatnost pojavljivanja simbola *don't care* $P_{don'tCare} = 0.33$, propadanje pogreške predviđanja $predictionErrorReduction = 0.25$, propadanje dobrote $fitnessReduction = 0.1$. Veličina turnira prilikom turnirske selekcije je 40% veličine akcijskog skupa. Nagrada okoline iznosi 1000 za ispravnu klasifikaciju, a 0 za neispravnu. Prilikom generiranja novih programskih isječaka, terminalni čvor ima 50% vjerojatnosti poprimanja programskog isječka naučenog na nižim razinama. Na osi apscisa nalazi se broj do tad korištenih testnih primjera, a na osi ordinata pomični udio točno klasificiranih primjera prilikom prethodnih 1000 iteracija *iskoristi*.

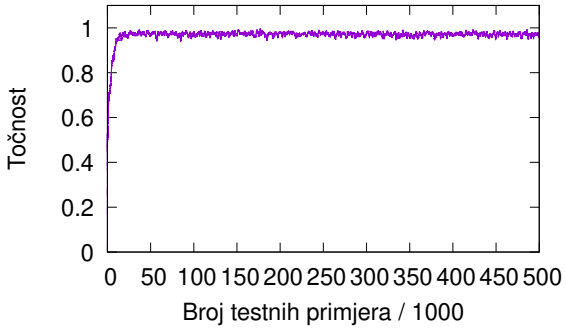
Sva testiranja odrađena su na AMD A10-8700p četverojezgrenom procesoru frekvencije 1.8 GHz i radnoj memoriji od 12 GB.

Na slici 5.1 prikazani su grafovi ponašanja sustava na problemu multipleksora. Veličina populacije N redom je 500, 1000, 2000, 5000 za multipleksore 4/1, 8/1, 16/1 i 32/1. Multipleksori 4/1, 8/1 i 16/1 trenirani su na $5 \cdot 10^5$ testnih primjera, a 32/1 na 10^6. Treniranje za multipleksore 4/1, 8/1 i 16/1 trajalo je ukupno oko 7 minuta, dok je treniranje multipleksora 32/1 trajalo otprilike 50 minuta. Broj mogućih različitih primjera multipleksora 32/1 je $2^{37} \approx 10^{11}$. Iz slike 5.1d se vidi da je već nakon otprilike $3 \cdot 10^5$ testnih primjera točnost sustava oko 90% iako je prostor pretraživanja jako velik, XCSCFC se na njemu ponaša poprilično dobro.

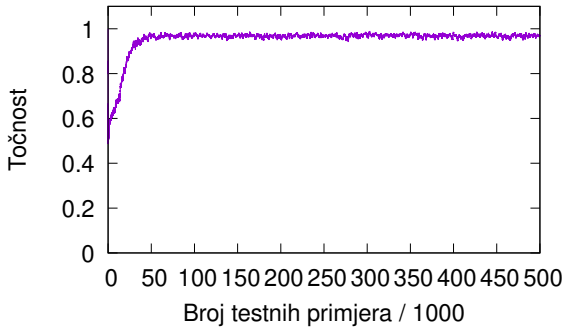
S obzirom da je, između ostalog, cilj rada bio prikazati kako radi ponovno iskorištavanje naučenog znanja na manjim problemima, za usporedbu, na slici 5.2 prikazan je graf ponašanja sustava na problemu multipleksora, ali bez korištenja prethodno stečenog znanja. Ovdje je svaki multipleksor učen od nule, bez znanja o manjim multipleksorima. Posebno je zanimljivo pogledati sliku 5.2d iz koje se vidi da u ovom slučaju, na primjeru multiplek-



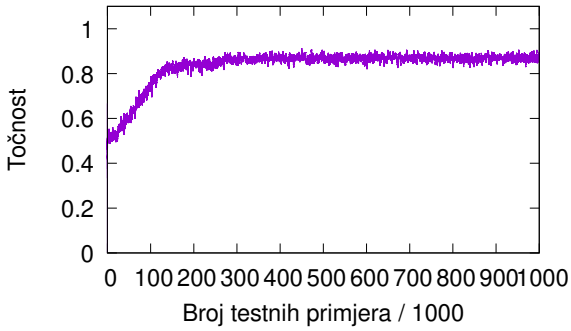
(a) Multipleksor 4/1.



(b) Multipleksor 8/1.

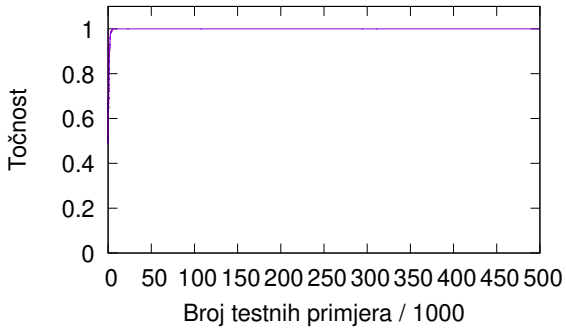


(c) Multipleksor 16/1.

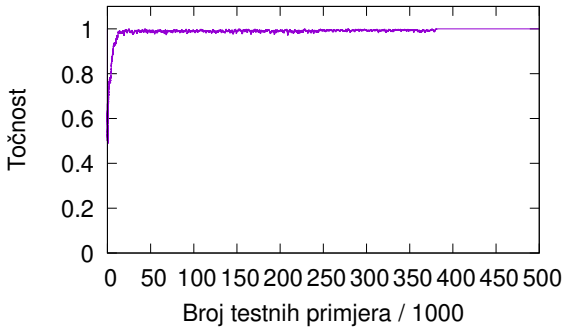


(d) Multipleksor 32/1.

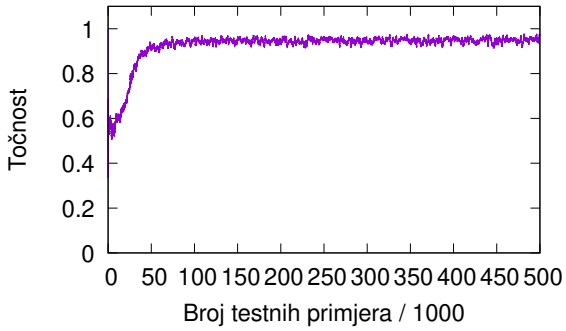
Slika 5.1: Ponašanje sustava na problemu multipleksora s iskorištavanjem već naučenog znanja.



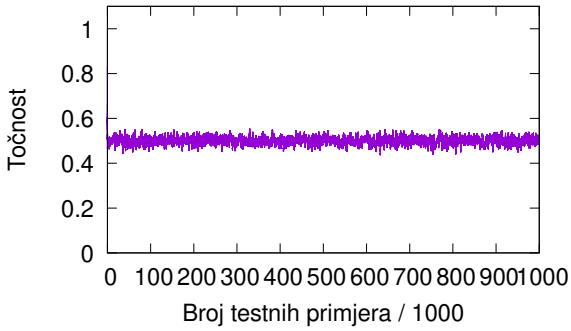
(a) Multipleksor 4/1.



(b) Multipleksor 8/1.



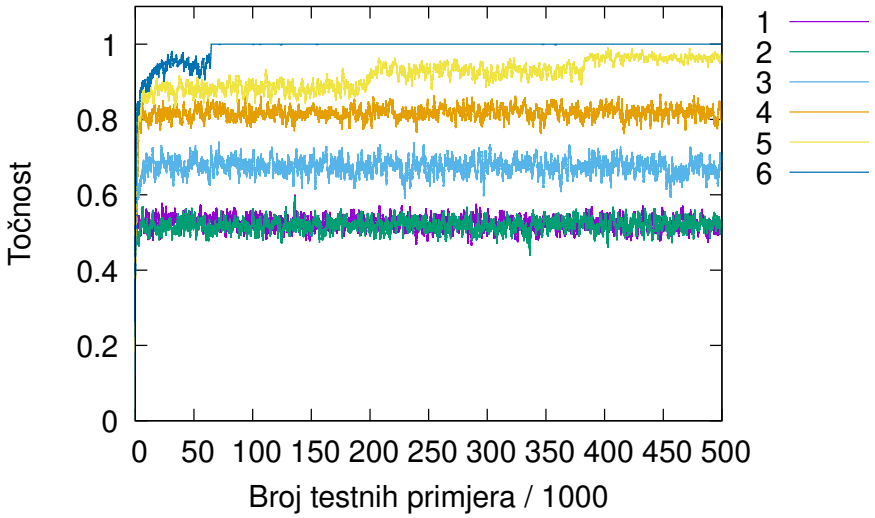
(c) Multipleksor 16/1.



(d) Multipleksor 32/1.

Slika 5.2: Ponašanje sustava na problemu multipleksora bez iskorištavanja već naučenog znanja.

sora 32/1, sustav nije naučio gotovo ništa, nasuprot vrlo dobrom rezultatu prilikom učenja multipleksora 32/1 uz korištenje prethodno stečenog znanja.



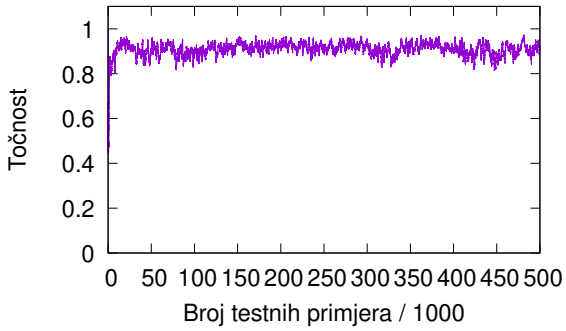
Slika 5.3: Prikaz učenja 6 bitnog multipleksora u ovisnosti o broju programskih isječaka koje klasifikatori sadržavaju.

S obzirom da je, uvođenjem programskih isječaka umjesto uvjetnih bitova, maknuta ovisnost duljine ulaza dobivenog od okoline i broja uvjetnih bitova unutar pravila, prilikom korištenja XCSCFC sustava moguće je imati broj programskih isječaka različit od veličine ulaza. U svrhu testiranja ovisnosti broja programskih isječaka unutar pojedinog pravila i kvalitete XCSCFC sustava, na slici 5.3 prikazan je graf ponašanja sustava na multipleksoru 4/1 s različitim brojevima programskih isječaka. Rezultat je očekivan, a vidi se i da između korištenja jednog programskog isječka i korištenja dva programska isječka u ovom primjeru nema razlike.

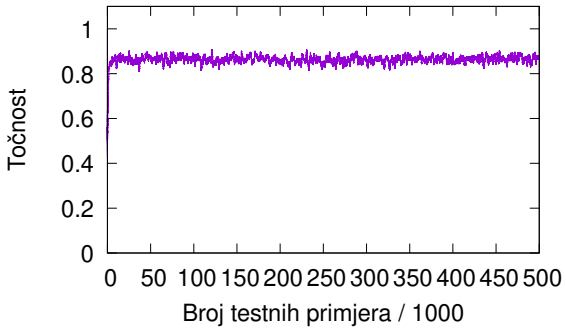
Rezultati dobiveni na problemu najzastupljenijeg bita prikazani su na slici 5.4. Veličina populacije N ovdje je redom 500, 1000 i 2000 za 3-, 5- i 7-bitni problem. Broj testnih primjera za svaku veličinu je $5 \cdot 10^5$. Prostor pretraživanja je ovdje znatno manji od onoga u problemu s multipleksorom, rezultati nisu znatno bolji. Zanimljivo je primjetiti da je sustav ovdje bolje naučio 7-bitni (slika 5.4c) problem nego 3-bitni (slika 5.4a), a prostor pretraživanja u 7-bitnom problemu je 16 puta veći.

Rezultati na problemu parnog pariteta prikazani su na slici 5.5. Veličina populacije N redom je 200, 300, 400, 500 za 2-, 3-, 4- i 5-bitni problem. Broj testnih primjera za svaku veličinu je $5 \cdot 10^5$. Iz slike se vidi da je problem s 2 bita (slika 5.5a) riješen s jako velikom točnošću, ali zbog toga što je vjerojatno naučen na pamet. Iz slike 5.5b se vidi da se već prilikom učenja za 3 bita javljaju problemi.

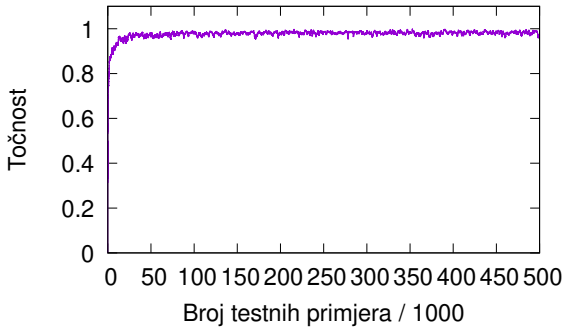
Rezultati na problemu određivanja bita prijenosa prilikom zbrajanja binarnih brojeva prikazani su grafom na slici 5.6.



(a) Problem s 3 bita.

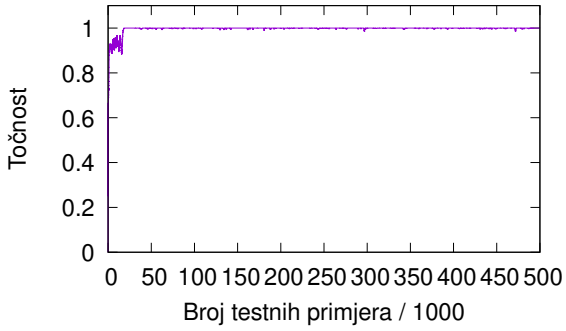


(b) Problem s 5 bita.

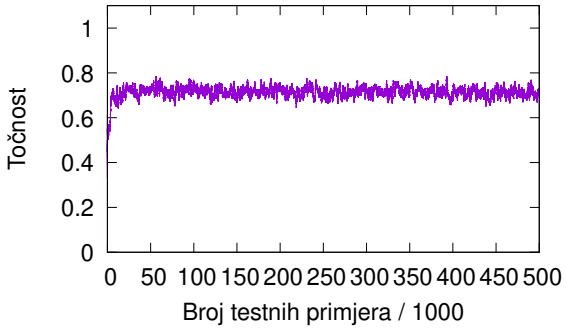


(c) Problem sa 7 bita.

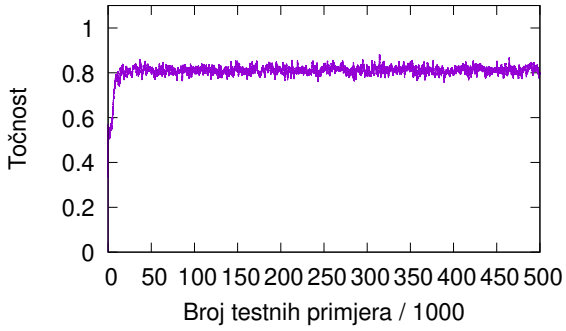
Slika 5.4: Ponašanje sustava na problemu pronalaska bita s najvećom frekvencijom pojavljivanja.



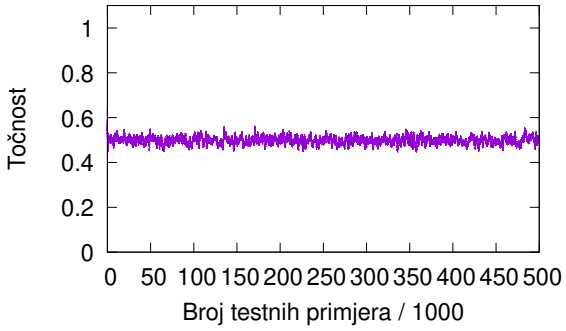
(a) Problem s 2 bita.



(b) Problem s 3 bita.

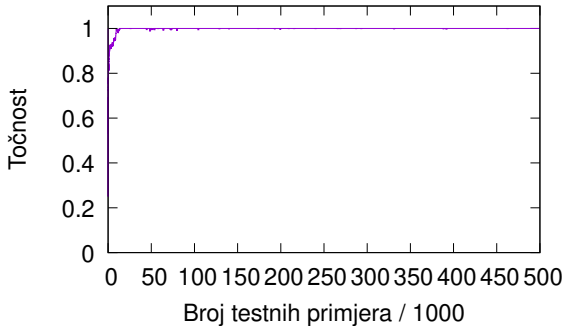


(c) Problem s 4 bita.

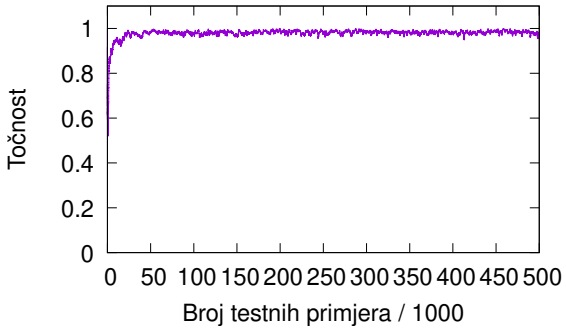


(d) Problem s 5 bita.

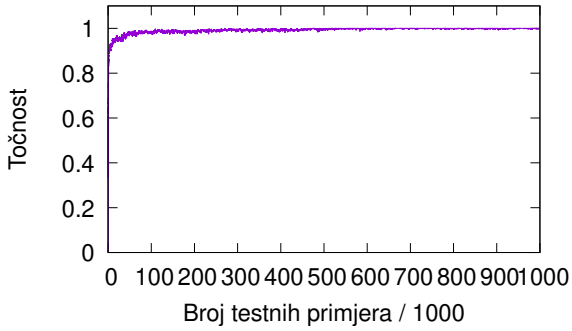
Slika 5.5: Ponašanje sustava na problemu parnog pariteta.



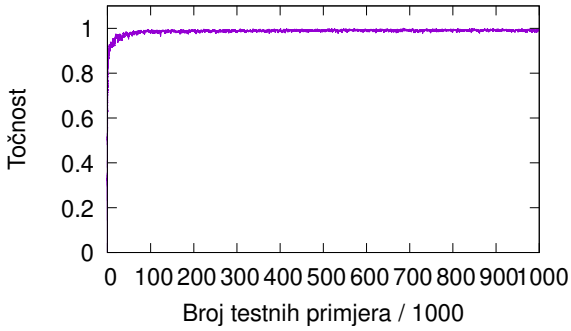
(a) Zbrajanje 2-bitnih brojeva.



(b) Zbrajanje 3-bitnih brojeva.



(c) Zbrajanje 4-bitnih brojeva.



(d) Zbrajanje 5-bitnih brojeva.

Slika 5.6: Ponašanje sustava na problemu bita prijenosa prilikom binarnog zbrajanja.

6. Zaključak

Susta XCS pokazao se kao vrlo složen, koji za neke probleme (npr. multipleksor) radi iznimno dobro, a za neke (npr. paritet) poprilično loše.

Zbog podjele sustava u više dijelova koji slijedno djeluju, otvara se puno prostora za isprobavanje. Sustav ima brojne parametre s kojima se može eksperimentirati i prilagoditi ga problemu koji se rješava.

U okviru ovog rada isproban je rad sustava samo na booleovim funkcijama, ali one predstavljaju temelj za nastavak rada na složenijim problemima. Budući rad na ovom sustavu može uključivati prilagodbu sustava za klasifikaciju (npr. jednostavnih slika) u više razreda.

Ovaj rad je motiviran radom prikazanim u (Iqbal et al., 2014), no ovdje su rezultati nešto lošiji nego što je prikazano u navedenom radu. Razlog je najvjerojatnije taj što neki dijelovi nisu precizno objašnjeni, ali glavna poanta rada je usvojena.

Na primjeru multipleksora pokazano je da korištenje znanja prethodno naučenog na jednostavnijim primjerima unutar iste domene poboljšava rezultate sustava. Znanje s nižih razina uspješno je sakupljeno i proslijeđeno prilikom učenja na težim problemima u domeni.

LITERATURA

Larry Bull. *Learning Classifier Systems: A Brief Introduction*, stranice 1–12. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. ISBN 978-3-540-39925-4. doi: 10.1007/978-3-540-39925-4_1. URL https://doi.org/10.1007/978-3-540-39925-4_1.

M. V. Butz, T. Kovacs, P. L. Lanzi, i S. W. Wilson. Toward a theory of generalization and learning in xcs. *IEEE Transactions on Evolutionary Computation*, 8(1):28–46, Feb 2004. ISSN 1089-778X. doi: 10.1109/TEVC.2003.818194.

S. W. Butz, M. V.and Wilson. An algorithmic description of xcs. *Soft Computing*, 6(3): 144–153, Jun 2002. ISSN 1432-7643. doi: 10.1007/s005000100111. URL https://doi.org/10.1007/s005000100111.

A. E. Eiben i James E. Smith. *Introduction to Evolutionary Computing*. Springer Publishing Company, Incorporated, 2nd izdanju, 2015. ISBN 3662448734, 9783662448731.

M. Iqbal, W. N. Browne, i M. Zhang. Reusing building blocks of extracted knowledge to solve complex, large-scale boolean problems. *IEEE Transactions on Evolutionary Computation*, 18(4):465–480, Aug 2014. ISSN 1089-778X. doi: 10.1109/TEVC.2013.2281537.

Richard S. Sutton i Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st izdanju, 1998. ISBN 0262193981.

Stewart W. Wilson. Classifier fitness based on accuracy. *Evol. Comput.*, 3(2):149–175, Lipanj 1995. ISSN 1063-6560. doi: 10.1162/evco.1995.3.2.149. URL http://dx.doi.org/10.1162/evco.1995.3.2.149.

Primjena sustava LCS na klasifikacijske probleme

Sažetak

Za rješavanje problema sa jako velikim prostorom pretraživanja potrebno je pronaći novo, drugačije rješenje, zbog trenutne velike vremenske i prostorne složenosti. U ovom radu, ostvareno je sakupljanje znanja sa jednostavnijih problema te njihovo ponovno korištenje u složenijim problemima. Navedena funkcionalnost ostvarena je korištenjem sustava LCS. Sustav je testiran na problemima multipleksora, većinskog bita, bita prijenosa i parnog pariteta. Pokazano je da sustav koristeći prethodno znanje probleme rješava bolje nego učenjem novih problema ispočetka.

Ključne riječi: Učeni klasifikacijski sustavi, strojno učenje, podržano učenje, genetski algoritmi, ponovno iskorištavanje znanja.

Application of LCS on Classification Problems

Abstract

To succesfully solve a large-scale problems there has been a need for a new, different approach, due to the large time and space complexity. In this work, knowledge extraction from smaller problems and its reuse in more complex problems has been achieved. The proposed functionality has been achieved on the LCS system. The system is tested on multiplexer problems, majority-on problems, carry problems and even-parity problems. It is shown that better results are achieved while using the extracted knowledge than when every problem is learned from scratch.

Keywords: Learning classifier systems, machine learning, reinforcement learning, genetic algorithms, knowledge reuse.