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 sa jednostavnijim problemima unutar sličnog područja.

2. Pregled područja

LCS (engl. *Learning Classifier Systems*) sustavi 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 sa nepoznatom okolinom, od koje pomoću osjetila dobiva ulaz, a preko djelovatelja 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*". 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.

Evolucijsko računarsvo bavi se algoritmima pretraživanja temeljenima na prirodnoj selekciji. Ideja ovog pristupa je da se od početne proizvoljno generirane populacije jedinki, postupcima prirodne selekcije, križanja i mutacije, postupno generiraju bolje i prilagođenije jedinke. Detaljna razrada evolucijskog računarstva ne ulazi u opseg ovog rada, stoga ono ovdje neće biti detaljnije opisano. Više informacija čitatelj može pronaći u (Eiben i Smith, 2015).

Prilikom rada LCS sustava, 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 potpornog učenja, koje usmjerava potragu za boljima pravilima (Bull, 2004).

Potporno učenje je učenje temeljeno na pokušajima, nakon kojih sustav dobije određenu brojčanu nagradu. U ovisnosti o nagradi, sustav podešava svoje parametre s ciljem povećavanja buduće nagrade te se na taj način prilagođava problemu kojeg rješava. Detaljan opis postupaka potpornog učenja također izlazi iz opsega ovog rada, stoga čitatelj više informacija može pronaći u (Sutton i Barto, 1998).

Dva glavna pristupa u implementaciji LCS sustava su *Michigan-Style* LCS i *Pittsburgh-Style* LCS. Glavna razlika je u tome što *Pittsburgh-Style* LCS sustav koristi više skupova pravila, od kojih je svaki od tih skupova moguće konačno rješenje, a genetski algoritam

djeluje na jednom cijelom skupu pravila. S obzirom da *Pittsburgh-Style* LCS nije tema ovog rada, u nastavku je dan detaljniji opis *Michigan-Style* LCS sustava.

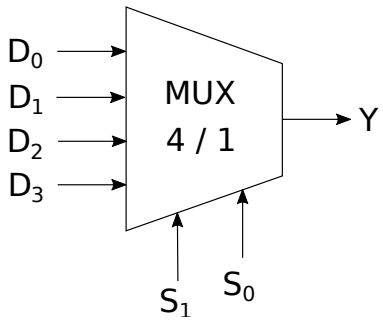
Michigan-Style LCS (u nastavku samo LCS) sustav prvi je formalizirao John Holland i u suradnji sa Judith Reitman dao njegovu implementaciju. S obzirom na složenost originalnog LCS sustava, malo jednostavniju i razumljiviju verziju dao je Stewart W. Wilson pod nazivom ZCS (*"zeroth-level" classifier system*). Nakon toga, Wilson je uveo još jednu verziju LCS sustava pod nazivom XCS, u kojemu je promijenio način na koji se računa *fitness* pojedinih pravila. U nastavku slijedi opis navedenih verzija LCS sustava, koji se detaljnije može pročitati u (Bull, 2004).

3. Opis problema

Problemi testirani u ovom radu su:

1. multipleksor,
2. paritetni bit,
3. najzastupljeniji bit,
4. bit prijenosa binarnog zbrajala.

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 sa k upravljačkih bitova funkcija od $k + 2^k$ varijabli. Izlaz iz multipleksora je uvijek točno 1 bit, koji ima vrijednost 0 ili 1. Primjer 4/1 multipleksora prikazanog nizom od 6 bitova je 011010. U ovom primjeru prva dva bita, 01, označavaju da je izlaz iz multipleksora drugi od preostalih bitova, odnosno 0.

Problem paritetnog bita svodi se na ispitivanje broja pojavljivanja bita 1 unutar ulaza. Ako je broj pojavljivanja bita 1 paran, izlaz treba biti 1, a u suprotnom 0. Primjer 6-bitnog ulaza je 011010, gdje je broj pojavljivanja bita 1 3, stoga izlaz treba biti 0.

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

Prilikom problema najzastupljenijeg bita, ispituje se frekvencija pojavljivanja pojedinih bitova. Izlaz je onaj bit koji se pojavljuje više puta. Na primjer, ako je 5-bitni ulaz 01101, bit 1 se pojavljuje 3 puta, a bit 0 2 puta, stoga izlaz treba biti 1. Ulaz za ovaj problem će uvijek biti neparne duljine, da izbjegnemo situaciju u kojoj se svaki bit pojavljuje jednak broj puta.

U posljednjem problemu, izlaz mora biti bit prijenosa prilikom zbrajanja dva binarna broja jednake duljine. Osigurano je da je ulaz uvijek parne duljine, kako bi mogao biti podijeljen na dva binarna broja. Ako ulaz ima n bitova, prvih $n/2$ bitova odgovara prvom broju, a posljednjih $n/2$ bitova drugom broju. Primjer ulaza za računanje prijenosa prilikom zbrajanja dva 3-bitna binarna broja je 011110, gdje je prvi broj 011, a drugi 110. Izlaz je ovdje 1, jer je nakon zbrajanja prijenos 1.

4. Opis algoritama

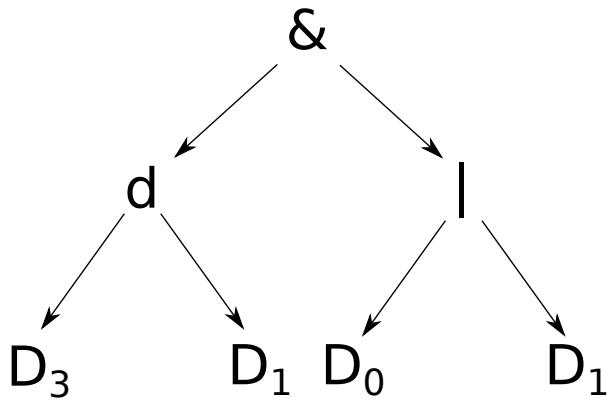
Prilikom razrade ovog sustava, bilo je potrebno ostvariti iskorištavanje znanja već naučenog na jednostavnijim problemima. Klasični bitovi uvjeta u ranije opisanim pravilima nam to onemogućavaju. Iz tog razloga, svaki bit uvjeta zamijenjen je programskim isječkom, koji ovisno o ulazu, vraća 0 ili 1. Takav XCS sustav, 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 čiji *fitness* je veći od prosječnog unutar te populacije pravila. (Iqbal et al., 2014)

U ovom radu, isječci koda su modelirani binarnim stablima duljine do najviše 2, što znači da možemo imati najviše 7 čvorova. Set funkcija koje čvorovi mogu obavljati je $\{AND, OR, NAND, NOR, NOT\}$. U primjerima, te će se funkcije redom označavati sa $\&, |, d, r, \sim$. Skup mogućih završnih 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 isječak koda na ulaz dobije cijeli ulaz dobiven od okoline, a na izlaz 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 *postfix* obliku. Iz slike također vidimo da se u binarnom stablu ne moraju pojavljivati svi bitovi ulaza, a također i da se pojedini bitovi mogu pojavljivati više puta.

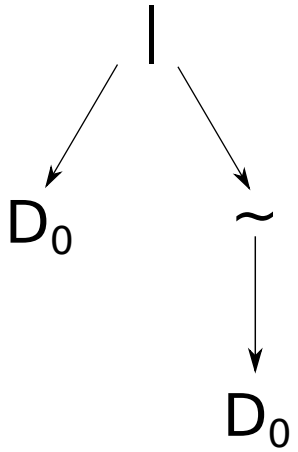
Potreban je i isječak koda koji označava *don't care* simbol, a koji će za svaki niz bitova koje dobije na ulazu vratiti 1. On je prikazan na slici 4.2 i označava operaciju $D_0D_0 \sim |$, a preuzet je iz (Iqbal et al., 2014).

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

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



Slika 4.1: Primjer programskog isječka prikazanog binarnim stablom.



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

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

U *Explore* načinu rada, na početku od okoline dobijemo ulazni podatak *s*. U ovisnosti o *s*, formira se *podudarni skup* (engl. *Match set*) [*M*]. [*M*] se sastoji od svih pravila iz populacije [*P*] koja 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 uopće nije bitan. Algoritam 1 prikazuje postupak evalucaije 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 u pravilu. *cl.cond* je polje svih programskih isječaka. Algoritam vraća vrijednost *true* ako svaki

[2]Od engleskog "*Classifier*"

programski isječak za zadani ulaz *s* vraća vrijednost 1, a *false* inače.

Algorithm 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 $true$

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 *don't care* simbol sa vjerojatnošću $P_{don'tCare}$, a sa 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*].

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 *Roulette-Wheel* postupkom, na temelju glasova koje svako pravilo daje. Postupak izračuna glasova prikazan je algoritmom 4. Glas svakog pravila temelji se na prosječnoj veličini akcijskog skupa. Razlog tomu je pokušaj ostvarenja približno jednakih veličina akcijskih skupova. Također, ako je pravilo dovoljno iskusno, a $fitness$ pravila je znatno manji od prosječnog $fitness$-a, vjerojatnost njegovog izbacivanja se dodatno povećava u ovisnosti o $fitness$-u. Time je osigurano izbacivanje lošijih pravila. Iskustvo pravila cl određeno je brojem pojavljivanja tog pravila unutar akcijskog skupa i pamti se u varijabli $cl.exp$. Konstanta θ_{del} određuje granicu iskustva nakon koje se može reći da je pravilo dovoljno iskusno za brisanje. Konstanta δ (iz intervala (0, 1]) određuje minimalni postotak prosječnog $fitness$-a populacije kojega pravilo mora imati da se njegov glas ne bi dodatno povećao. Vjerojatnost odabira svakog pravila prilikom *Roulette-Wheel* postupka jednaka je postotku glasa tog pravila u odnosu na ukupnu sumu glasova. Nakon što je pravilo cl izabrano za brisanje, provjerava se njegova

Algorithm 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

brojnost (engl. *()*numerosity) sadržana u $cl.n$. Ako je brojnost pravila veća od 1, ona se samo umanjuje za 1. U suprotnom, pravilo se izbacuje iz populacije.

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* (engl. *prediction array*). Ovisno o vrijednostima $P(a)$, *Roulette-Wheel* postupkom 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 (engl. *Action set*) [*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.

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

Algorithm 3 Operacija brisanja

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

Algorithm 4 Glas

Ulaz: cl – pravilo čiji glas računamo, $prosjecniFitness$ – prosječni $fitness$ populacije.
Izlaz: $glas$ – glas pravila cl
$vote := cl.as \cdot cl.n$
if $cl.exp > \theta_{del}$ and $cl.F/cl.n < \delta \cdot prosjecniFitness$ **then**
  $vote := vote \cdot prosjecniFitness/(cl.F/cl.n)$
end if
return $glas$

Po dobitku nagrade, dolazi do ažuriranja parametara svih pravila sadržanih u [*A*]. Redom se ažuriraju iskustvo pravila (engl. *Experience*) exp, njegovo predviđanje (engl. *Prediction*) p, pogreška u predviđanju (engl. *Prediction error*) ϵ, preciznost (engl. *Accuracy*) κ, relativna preciznost (engl. *Relative accuracy*) κ', *fitness* 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 formuli (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 formuli (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 formuli (4.4).

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

Prije ažuriranja $fitness$-a 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'.

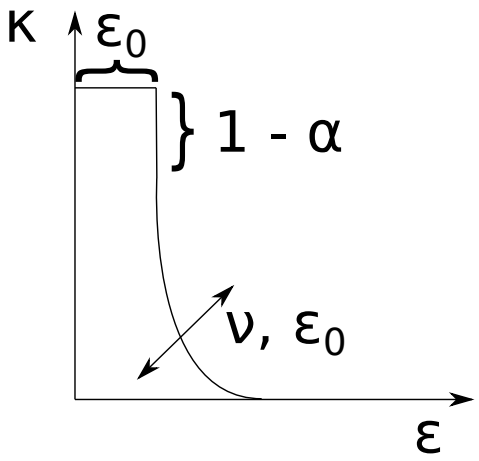
$$cl.\kappa := \begin{cases} 1 & \text{, ako je } \epsilon < \epsilon_0 \\ \alpha \cdot \left(\frac{\epsilon}{\epsilon_0}\right)^{-\nu} & \text{, inače} \end{cases} \tag{4.5}$$

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

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 formulom (4.5). Nakon što su izračunate preciznosti svih pravila unutar akcijskog seta [*A*], svaka preciznost se normira ukupnom sumom preciznosti, kao što je prikazano formulom (4.6). Kada je izračunata relativna preciznost, $fitness$ se ažurira prema formuli (4.7).

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

$Fitness$ vrijednost pravila je procjena njegove preciznosti s obzirom na ostala pravila iz [*A*]. Na slici 4.3 vidi se utjecaj pojedinih parametara prilikom računanja preciznosti κ. Parametar



Slika 4.3: Funkcija ovisnosti preciznosti κ o pogrešci u predviđanju ϵ. Slika preuzeta iz (Butz et al., 2004)

ϵ_0 određuje do koje granice će pravila imati jednaku, maksimalnu, preciznost, parametar α uvodi 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 pravilo sudjelovalo u akcijskom setu nad kojim se provela operacija istraživanja novih pravila. Operacija otkrivanja novih pravila provodi se ukoliko je prosječno vrijeme proteklo od prošlog pokretanja operacije otkrivanja na pravilima unutar akcijskog seta 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 $fitness$ parametru. Prilikom turnirske selekcije, nasumično se odabire unaprijed zadani broj pravila, te se kao pobjednika odabire ono koje od odabranih ima najveću $fitness$ vrijednost. Od izabranih roditeljskih pravila stvaraju se dva potomka, od kojih prvotno svaki ima iste programske isječke uvjeta kao jedan od roditelja.

Nakon toga, sa vjerojatnošću χ provodi se križanje (engl. *Crossover*) 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

Algorithm 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

na akcije. Tijekom mutacije, svaki *don't care* simbol zamjenjuje se proizvoljno generiranim programskim isječkom koji odgovara stanju s dobivenom iz okoline, a svaki drugi programski isječak zamjenjuje se *don't care* simbolom. 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 $fitness$ na srednju vrijednost $fitness$-a 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 ono 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 ternarnih simbola 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.,

Algorithm 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 $cl.cond[i] =$ *don't care* simbol **then**
      repeat
        $cf :=$ generiraj proizvoljni programski isječak
        $val := evaluiraj(cf, s)$
      until $val \neq 1$
      $cl.cond[i] := cf$
    else
      $cl.cond[i] :=$ *don't care* simbol
    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

2014). Operacija provjere je li jedno pravilo općenitije od drugog prikazana je algoritmom 7 Da bi pravilo cl_1 bilo općenitije od pravila cl_2, cl_1 mora imati više *don't care* simbola 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 *don't care* simbola. 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 pravilo 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 pronađenom pravilu brojnost poveća za

Algorithm 7 Općenitije pravilo

Ulaz: cl_1 – općenitije pravilo, cl_2 – specifičnije pravilo.
Izlaz: – je li pravilo cl_1 općenitije od cl_2
$x :=$ broj *don't care* simbola u cl_1
$y :=$ broj *don't care* simbola 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 $true$

1. Postupak provjere jednakosti pravila također je drugačiji u odnosu na klasični XCS sustav. 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 *don't care* simbola i skupovi ostali 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 njegov $fitness$ veći od prosječnog $fitness$-a 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 4/1 multipleksoru korišteni kao listovi unutar isječaka 8/1 multipleksora. Jednako tako, u 16/1 multipleksoru, korišteni su isječci naučeni i na 4/1 i 8/1 multipleksoru.

U *exploit* načinu rada, XCS sustav se ne mijenja, nego izvršava najbolju moguću akciju. Početak je isti kao i u *explore* načinu rada. Formira se podudarni skup koji se sastoji od klasifikatora koji odgovaraju ulazu s. U ovisnosti o klasifikatorima sadržanim u [*M*], formira se *polje predviđanja*, na isti način kako je opisano u *explore* načinu rada. Na temelju formiranog *polja predviđanja*, odabire se akcija a sa najvećom vrijednosti $P(a)$. Ovaj odabir je druga-

Algorithm 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 *don't care* simbola u cl_1
$y :=$ broj *don't care* simbola 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).

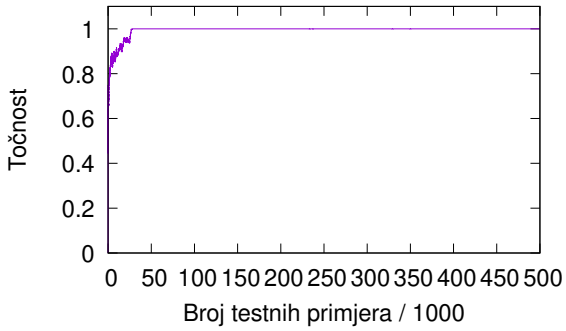
Multipleksor	Programski isječak	
	Ime	Izraz
MUX 4/1	L1_0	$D_1 D_0 D_4 dr$
	L1_1	$D_5 \sim D_1 D_0 \&\&$
	...	...
MUX 8/1	L2_0	$L1\_15 D_2 L1\_4 r\&$
	L2_1	$L1\_5 D_7 \mid L1\_11 D_3 \& r$
	...	...
MUX 16/1	L3_0	$L2\_9 L1\_7 D_{11} \mid r$
	L3_1	$L1\_10 D_{17} \mid L2\_1 D_0 r\&$
	...	...

čiji od onog opisanog u *explore* načinu rada, u kojem se akcija a odabire *Roulette-Wheel* postupkom u ovisnosti o vrijednosti $P(a)$.

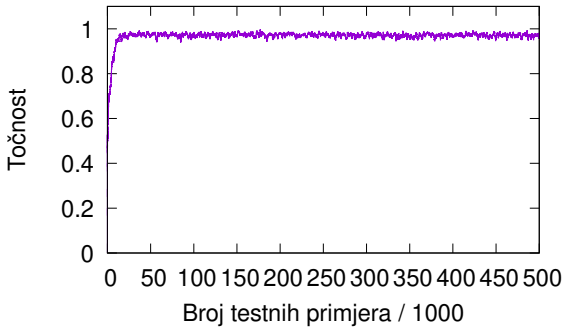
Explore i *exploit* načini rada se međusobno izmjenju, pri čemu *exploit* način rada služi testiranju sposobnosti sustava.

5. Rezultati

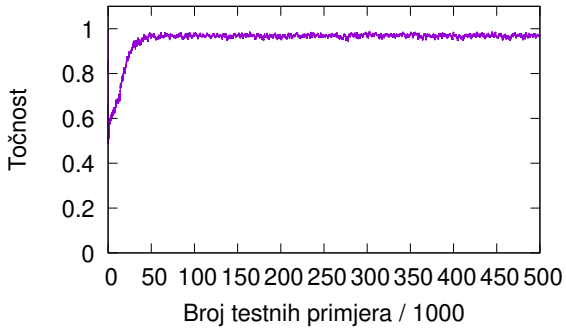
Prilikom testiranja rada sustava, korištene vrijednosti parametara sustava su podešene kao u (Iqbal et al., 2014). Stopa učenja $\beta = 0.2$, stopa propadanja dobrote $\alpha = 0.1$, granica pogreške predviđanja $\epsilon_0 = 10$, eksponent propadanja dobrote $\nu = 5$, granica prosječnog proteklog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



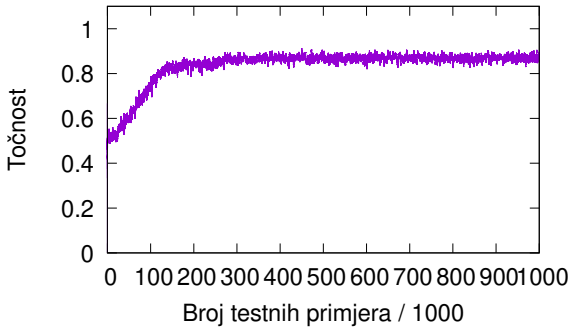
(a) Problem sa 6 bita.



(b) Problem sa 11 bita.

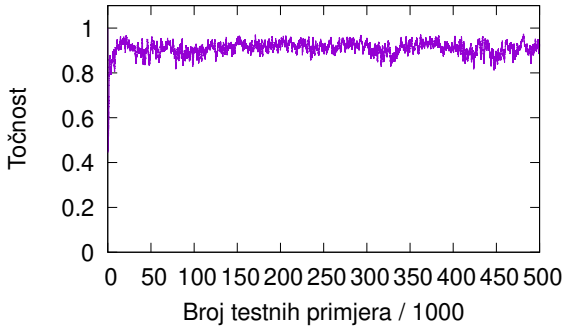


(c) Problem sa 20 bita.

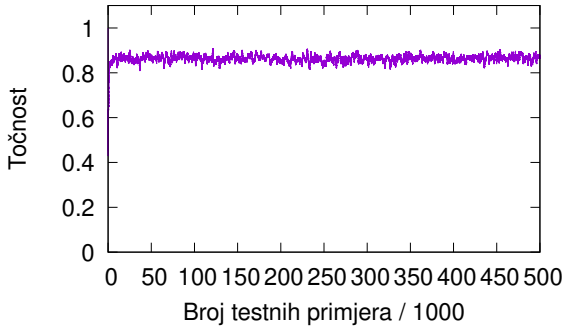


(d) Problem sa 37 bita.

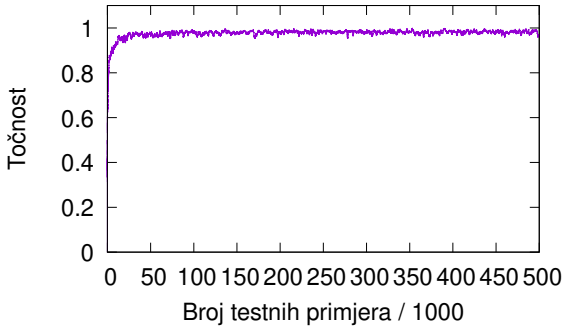
Slika 5.1: Ponašanje sustava na problemu multipleksora.



(a) Problem sa 3 bita.



(b) Problem sa 5 bita.



(c) Problem sa 7 bita.

Slika 5.2: Ponašanje sustava na problemu pronalaska bita s najvećom frekvencijom pojavljivanja.

6. Zaključak

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

Sažetak na hrvatskom jeziku.

Ključne riječi: Ključne riječi, odvojene zarezima.

Application of LCS on Classification Problems

Abstract

Abstract.

Keywords: Keywords.