SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 318

Klasifikatorski sustavi s mogućnošću učenja pravila jednostavne igre

Mario Lučić

Zagreb, lipanj 2008.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 318

Klasifikatorski sustavi s mogućnošću učenja pravila jednostavne igre

Mario Lučić

Zagreb, lipanj 2008.

Klasifikatorski sustavi s mogućnošću učenja pravila jednostavne igre

Sažetak

Klasifikatorski sustavi sa sposobnošću učenja su sustavi temeljeni na pravilima koji sjedinjuju tehnike strojnog učenja s genetskim algoritmom. U ovom radu detaljno su opisani sustavi *ZCS* i *XCS* koji su kasnije korišteni za određivanje pobjedničkih strategija u dvije logičke igre.

Ključne riječi: *Klasifikatorski sustavi, sustavi temeljeni na pravilima, ZCS, XCS, učenje potkrjepljenjem, genetski algoritam, teorija igara, igra Nim.*

Learning Classifier Systems applied to simple logic games playing

Abstract

Learning Classifier Systems are rule-based systems which combine machine learning with genetic algorithm. In this paper the *ZCS* and *XCS* classifier systems are discussed in detail and then used to determine the correct strategy in two logic games.

Key words: *Classifier systems, Rule-based systems, ZCS, XCS, reinforcement learning, genetic algorithm, game theory, The Game of Nim.*

Sadržaj

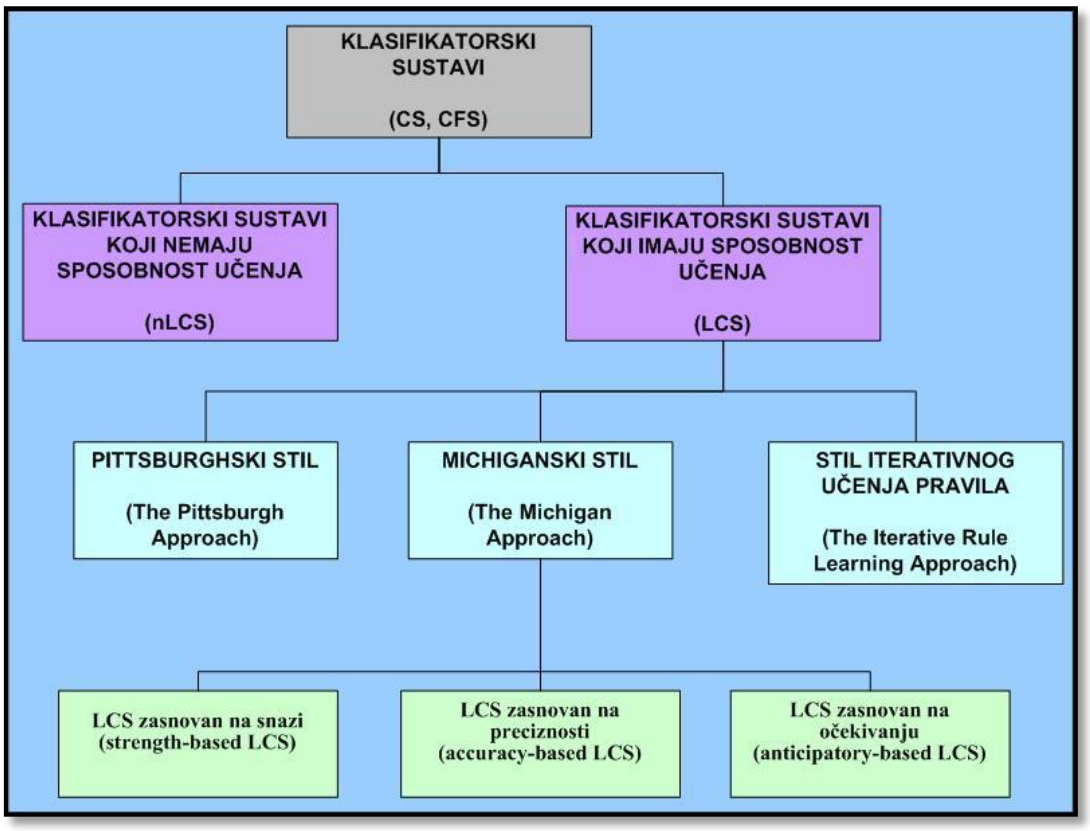
1 Uvod

Povećana složenost mnogih problemskih domena dovelo je do potrebe za rješenjima koje se mogu prilagoditi pojedinom zadatku. Korištenjem evolucijskih algoritama i učenja potkrjepljenjem modeliraju se sustavi koji imaju mogućnost prilagođavanja. Klasifikatorski sustavi sa sposobnošću učenja *(eng. Learning Classifier Systems, LCS)* temeljeni na pravilima jedna su od tehnika strojnog učenja *(eng. Machine Learning)*. Ideja strojnog učenja je načiniti sustav koji će automatski stjecati znanje kroz iskustvo, odnosno primjere. Cilj *LCS*-a je pronaći implicitnu funkciju koja preslikava skup stanja okoline na skup pravila sustava koja će maksimizirati ukupnost nagrada primljenih od okoline.
Evolucijski algoritmi temeljeni su na Darwinovim principima preživljavanja najspremnijih, dok sama populacija pravila predstavlja skup rješenja klasifikacijskog problema. Uz pomoć genetskog algoritma *(eng. Genetic algorithm)* nad populacijom se vrše stohastički procesi mutacije gena i rekombinacije. Ideja zajednička svim evolucijskim metodama je pretraživanje problemskog prostora evoluirajući početni nasumce odabrani skup rješenja tako da na kraju svakog koraka skup rješenja bude kvalitetniji, odnosno prilagođeniji problemu. Cilj svake jedinke je opstati, dok je cilj sustava ukloniti one jedinke koje ne pridonose maksimiziranju nagrada dobivenih iz okoline. *Učenje potkrjepljenjem (eng. Reinforcement learning)*, preslikano iz Pavlovljevih metoda u psihologiji, postiže se metodom pokušaja i promašaja uz nagrade ili kazne od strane okoline. Sustav se kroz niz iteracija prilagođava okolini i nakon određenog vremena jedinka ili populacija jedinki evoluira u rješenje klasifikacijskog problema.
Najčešće primjene *LCS-a* su na području *otkrivanja znanja u skupovima podataka (eng. Data Mining), upravljanja procesima* te *modeliranja i optimizacije* [3].

Cilj ovog rada je proučiti mogućnosti korištenja *LCS-a* u području logičkih igara. Rad će biti temeljen na dvije relativno jednostavne logičko-kombinatorne igre u kojima će optimalne strategije određivati *LCS*. Detaljno će se proučiti utjecaj pojedinih parametara sustava na ispravnu i efikasnu klasifikaciju problemskog prostora. U eksperimentalnom dijelu rada bit će opisan način treniranja sustava te će se analizirati problemi s kojima se sustav susreće.

2 Klasifikatorski sustavi

Klasifikatorski sustavi su sustavi temeljeni na skupu pravila odnosno klasifikatora koji određuju reakciju sustava na uvjete dane u njegovoj okolini. Klasifikatorski sustavi dijele se na klasifikatorske sustave koji nemaju sposobnost učenja i klasifikatorske sustave koji imaju sposobnost učenja *(eng. Non-Learning Classifier Systems, nLCS)* [10].
nLCS je sustav sastavljen od *detektora*, *efektora*, liste poruka i skupa pravila oblika *uvjet (uvjeti) : akcija* značenja *ako uvjet (uvjeti) onda akcija*. Pomoću detektora se stanje okoline prenosi u listu poruka. Zatim se iz skupa pravila izabire pravilo čiji uvjet odgovara sadržaju liste poruka. Akcija koju sustav treba izvesti određena je pravilom i njome se popunjava lista poruka. U zadnjem koraku se pomoću *efektora* izvodi akcija smještena u listi poruka [2].
Ukoliko se *nLCS* proširi komponentom za raspodjelu nagrada primljenih od okoline i komponentom za istraživanje prostora mogućih pravila, dobiva se *LCS* [9]. Cilj *LCS*-a je odrediti takav skup pravila kojim će se maksimizirati nagrade primljene od okoline. Treba napomenuti da je ovakvim proširenjem dobiven *michiganski* stil *LCS*-a.



Slika 2.1 Podjela LCS-a

LCS sustave prvi je opisao J. Holland 1975. kao sustave koji posjeduju kognitivne značajke [3], [4]. Tijekom 1980-tih kristalizirala su se dva stila *LCS*-a temeljena na dva različita pristupa. Holland i njegovi suradnici oblikovali su na Sveučilištu Michigan *michiganski stil (eng. The Michigan Approach)*, dok je Smith sa svojim suradnicima na Sveučilištu Pittsburgh

oblikovao *pittsburghski stil (eng. The Pittsburg Approach)* [2]. Uz *michiganski* i *pittsburghski* stil postoji i stil iterativnog učenja pravila *(eng. The Iterative Rule Learning Approach)*, prvi puta primijenjen na SIA sustavima (Venturini, 1993) **[14]**. Podjela CS-a i *LCS*-a prikazana je na slici 2.1.

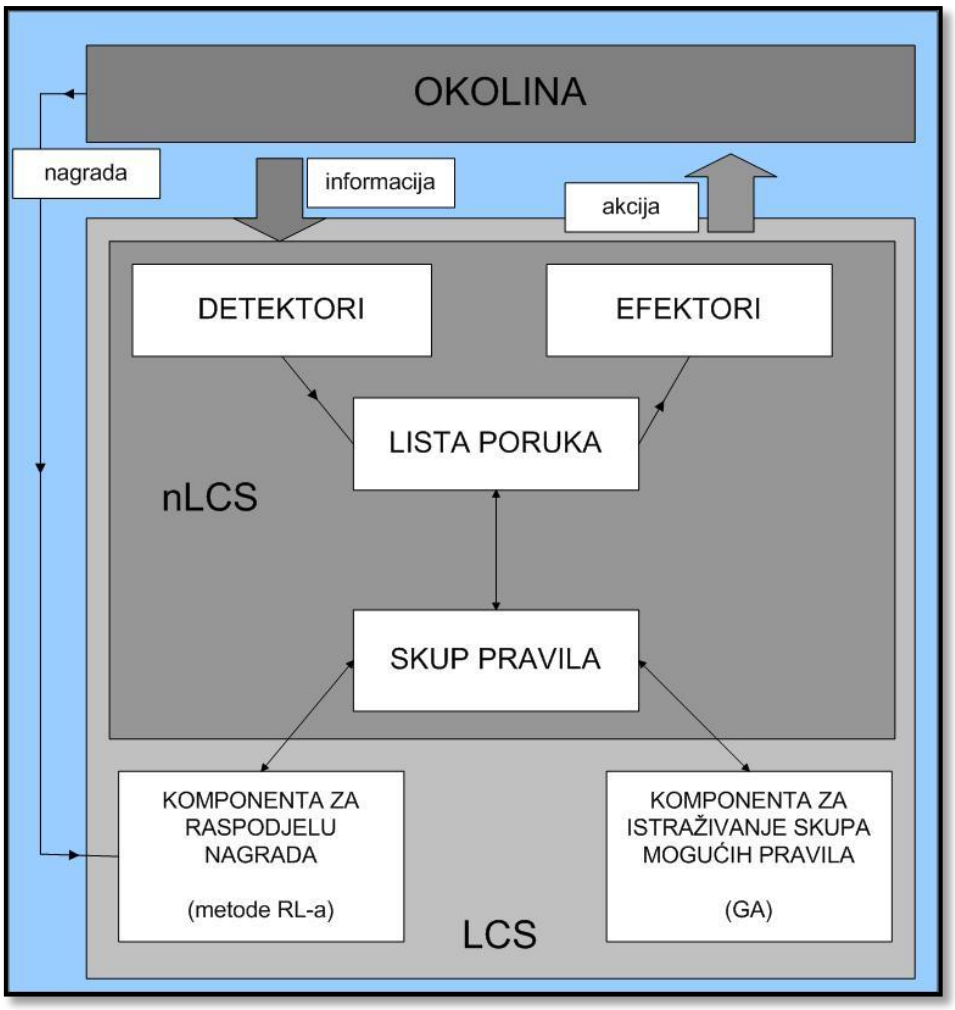
2.1 Michiganski stil LCS-a

Michiganski stil LCS-a karakterizira kombinacija učenja potkrjepljenjem i genetskog algoritma. *Učenje potkrjepljenjem* je tehnika učenja pokušajima i pogreškama putem primljenih nagrada, a u *LCS*-u predstavlja komponentu za raspodjelu nagrada primljenih od okoline [3].
Genetski algoritam je heuristička metoda optimizacije temeljena na evolucijskom procesu, a u *LCS*-u predstavlja komponentu za istraživanje prostora mogućih pravila. Ono što bitno razlikuje *michiganski stil LCS-a* od nekih drugih (klasičnih) tehnika učenja potkrjepljenjem je sposobnost generalizacije pravila: postoje pravila čiji uvjetni dio odgovara različitim stanjima okoline čime se smanjuje broj pravila kojima sustav treba raspolagati.
Potrebno je naglasiti kako prevelika generalizacija nije dobra. Naime, može se dogoditi da generalizirana pravila izbace specijalizirana pravila koja bolje odgovaraju na određeno stanje okoline [1].
U *michiganskom stilu* svaka jedinka predstavlja jedno pravilo. Evolucija i optimizacija pravila obavlja se lokalno, odnosno na pojedinim pravilima. Sustav nagrade iz okoline neposredno raspodjeljuje jedinkama (pravilima) čije su akcije nagrađene, a posredno i jedinkama (pravilima) čije su akcije prethodile nagrađenim akcijama.
Za dobivanje sustava koji će optimalno reagirati na stanja okoline, potrebno je uravnotežiti odnos potreba pojedine jedinke (pravila) i potrebe cijelog sustava [12]. Potreba jedinke (pravila) se očituje u tome da opstane, odnosno ne bude eliminirana od strane GA.
Ukoliko jedinka želi opstati mora težiti dobivanju što većeg dijela nagrade. S druge strane, potreba sustava je da jedinke zajedno djeluju na ispravan način. To znači da se nagrada dobivena određenom akcijom treba rasporediti ne samo na pravila koja su neposredno uzrokovala akciju, već i na ona pravila koja su dovela do stanja okoline u kojem je izvedena ta akcija.
Michiganski stil *LCS*-a pripada *on-line* sustavima i uči na temelju jedne instance problema [11]. Stil je pogodan za opisivanje *sustava u stvarnom vremenu* gdje nagle promjene ponašanja nisu dozvoljene [12].

Tri su porodice michiganskog stila *LCS*-a:

- *LCS* zasnovan na snazi *(eng. Strength-based LCS)*
- *LCS* zasnovan na preciznosti *(eng. Accuracy-based LCS)*
- *LCS* zasnovan na očekivanju *(eng. Anticipatory-based LCS)*
-

Iako su aktivna istraživanja i na drugim stilovima *LCS*-a, *michiganski* stil se smatra standardnom formom *LCS*-a [2]. Shematski prikaz *michiganskog stila LCS-a* predočen je na slici 2.2.



Slika 2.2 Odnos nLCS-a i michiganskog stila LCS-a

2.2 Kratak pogled na druge stilove LCS-a

Pittsburghski stil *LCS-a* pripada *off-line* sustavima i uči iterativno na temelju skupa instanci problema [11]. Optimizacija se provodi globalno nad skupovima pravila, a ne nad pojedinačnim pravilima kako je to slučaj u michiganskom stilu. Za razliku od michiganskog stila, u pittsburghskom stilu svaka jedinka predstavlja rješenje klasifikacijskog problema **[14]**. Istovremeno se raspolaže s nekoliko skupova pravila i to tako da jedna jedinka predstavlja jedan skup pravila. Pravila su fiksirane duljine, ali jedinka može sadržavati različiti broj pravila pa je jedinka promjenjive duljine. Dozvoljena su i generalizirana pravila.
Nad skupom jedinki poziva se GA koji koristi operacije rekombinacije, mutacije i selekcije prilagođene promjenjivim duljinama jedinki. Funkcija procjene (dobrote) određuje kvalitetu pojedine jedinke (skupa pravila) na temelju poklapanja pravila iz jedinke s primjerima za učenje, odnosno na temelju točnosti kojom jedinka obavlja klasifikaciju. Trajanje procesa učenja zadano je evolucijskim vremenom GA-a.

Stil iterativnog učenja pravila temelji se na pristupu „podijeli pa vladaj“ *(eng. Divide and Conquer).* Kao u *michiganskom stilu,* jedinka predstavlja jedno pravilo, dok se sličnosti sa *pittsburghskim stilom* pronalaze u kombiniranju SL-a sa GA-om**[13]**.
Sustav opisan stilom iterativnog učenja pravila uči skupom primjera koji se nalaze u skupu za učenje. Učenje traje sve dok skup pravila ne iskoristi zadane primjere. U svakom koraku se uzima nepokriveni primjer i uz pomoć GA-a se određuje dovoljno dobro pravilo za taj primjer. Ukoliko se pravilo ne nalazi u skupu pravila ono se dodaje, a iz skupa za učenje brišu se primjeri koji su pokriveni postojećim skupom pravila. Opisani ciklus ponavlja se dok god ima primjera u skupu za učenje. Funkcija procjene (dobrote) uzima u obzir generalizaciju (što veću pokrivenost primjera) i preciznost predviđanja pojedinih pravila. Model *LCS*-a temeljen na stilu iterativnog učenja pravila prikazan je na slici 2.4. Predstavnik ovoga stila je HIDER sustav **[13]**, **[14]**.

2.3 Tehnike i svojstva michiganskog stila LCS-A

Holland i Reitman su 1978. izložili prvu implementaciju LCS-a [3]. Iako je Hollandova verzija LCS-a bila implementirana i u rješavanju stvarnih problema, ona se pokazala složena za izvedbu i nešto kasnije našla je dobru zamjenu u jednostavnijim verzijama LCS-a. Ovdje se gradi model LCS-a koji se dijelom oslanja na izvornu verziju Hollandovog LCS-a. Zbog boljeg razumijevanja načina rada LCS-a prethodno je potrebno ukratko opisati GA i RL.

Kratak pogled na genetske algoritme

Genetski algoritam je heuristička metoda optimizacije zasnovana na principima Darwinove teorije evolucije. Algoritam je temeljen na populaciji rješenja i kroz niz koraka nastoji poboljšati populaciju tako da na kraju evolucijskog procesa najbolja jedinka populacije predstavlja dovoljno dobro rješenje optimizacijskog problema.

```
Genetski algoritam {
        Ponavljaj dok traje evolucija {
                Izaberi roditelje
                Križaj roditelje;
                Mutiraj dobivenu djecu
                Eliminiraj onoliko roditelja koliko je stvorene djece
        }
}
```

Slika 2.3 Pseudokod eliminacijskog GA

Na slici je opisan eliminacijski GA koji se koristi u michiganskom stilu *LCS*-a. Način odabira roditelja, vrsta križanja te tip eliminacije određuju genetski algoritam. Bitno je napomenuti da u genetskom algoritmu najbolja jedinka predstavlja rješenje problema, dok u *LCS*-u je rješenje problema predstavljeno čitavom populacijom. Ono što GA razlikuje od ostalih evolucijskih algoritama je korištenje operatora križanja i mutacije.

GA, koji je prvobitno bio tek dio *LCS*-a, s vremenom se počeo promatrati kao poseban algoritam i postao je mnogo popularniji od *LCS*-a [2]. U *LCS*-u GA ima funkciju istraživanja prostora mogućih pravila. Pri tome se uz GA mogu primijeniti i neke druge heurističke metode. Jedna od njih je i *cover* algoritam koji će kasnije biti opisan. [3].

Kratak pogled na učenje potkrjepljenjem

Učenje se primjenjuje svugdje gdje je potrebno efikasno pretraživati veliki prostor stanja. *Učenje potkrjepljenjem* temelji se na *operantnom uvjetovanju* (teoriji učenja u psihologiji), a odnosi se na područje *strojnog učenja* koje proučava kako agent treba djelovati u promatranoj okolini da bi primio što veću nagradu od okoline **[5]**. Riječ je o učenju nagradama i kaznama (uz pokušaje i pogreške). Za svaki dobar *odgovor* na stanje u okolini, okolina nagrađuje sustav, dok se loši odgovori kažnjavaju.

Formalno, učenje potkrjepljenjem se definira sljedećim skupovima:

- Skupom stanja okoline *S*;
- Skupom akcija *A*;
- Skupom nagrada (najčešće je to skup $\{0\ ,1\}$ ili interval realnih brojeva) **[7].**

U svakom koraku sustav na temelju stanja okoline odabire akciju iz skupa mogućih akcija. Okolina mijenja stanje te nagrađuje sustav u ovisnosti o kvaliteti akcije. Cilj sustava je implicitno odrediti onu funkciju pravila koja preslikava skup stanja okoline na skup akcija sustava tako da ukupnost nagrada od okoline bude najveća moguća. Za određivanje navedene funkcije ključno je riješiti *problem pridjeljivanja ocjene (eng. Credit Assignment problem)* odnosno određivanja stupnja u kojem svaki potez u nizu zaslužuje nagradu ili kaznu za konačni ishod igre. U *LCS*-u RL ima funkciju raspoređivanja korisnosti pojedinim pravilima [3].

2.4 Model LCS-a

Kao što je već rečeno, *LCS* se temelji na skupu pravila. Svako pravilo, odnosno klasifikator sljedećeg je oblika:

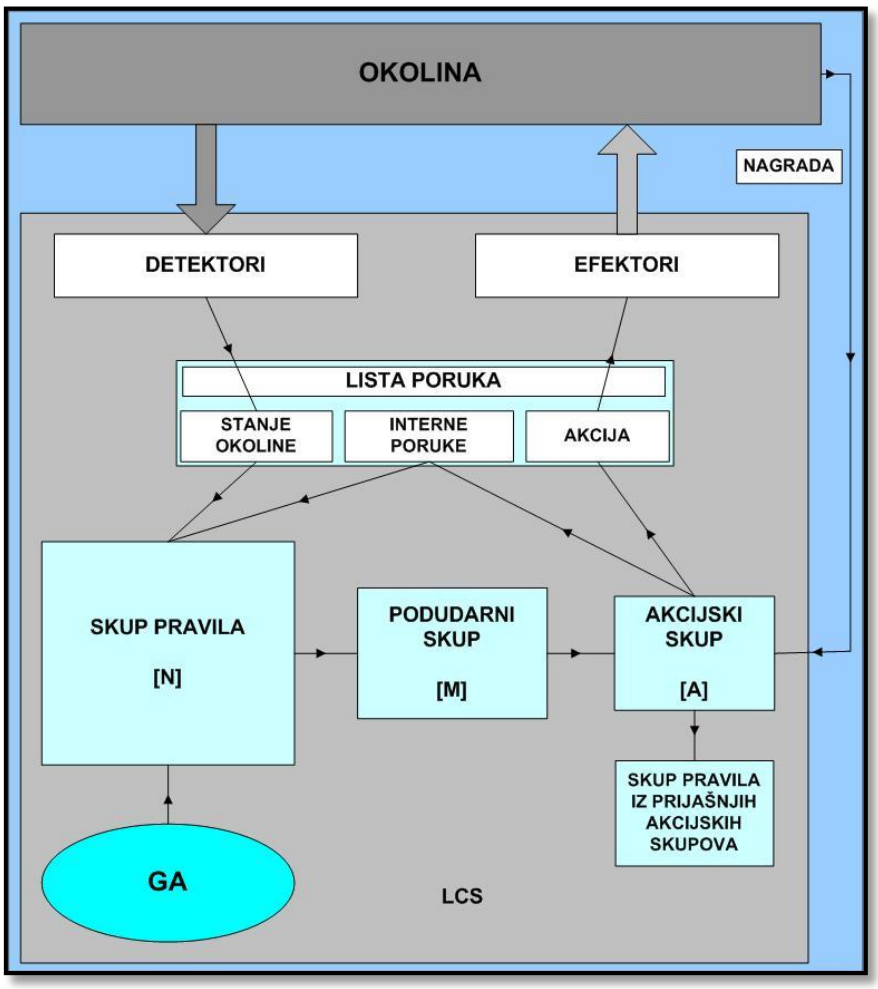
$$\boldsymbol{ako}\ uvjet\ \boldsymbol{onda}\ akcija$$

gdje je uvjet niz znakova iz ternarnog skupa $\{0, 1, \#\}$, a akcija niz znakova iz binarnog skupa $\{0, 1\}$ pri čemu znak # predstavlja i $0\ i\ 1$. Znak „#“ *(eng. Don't Care Symbol)* omogućava generalizaciju pravila koja će biti detaljno objašnjena u nastavku.

Kako bi prednosti i mane ovakvog načina kodiranja uvjeta i pravila postale vidljive, potrebno je na trenutak promotriti najpoznatije sustave koji su temeljeni na *ako-onda pravilima* – ekspertne sustave *(eng. Expert systems)*. U kontekstu ekspertnih sustava ta se pravila nazivaju produkcijska pravila. Za razliku od *LCS*-a uvjet i akcija predstavljeni su naredbama, jednadžbama ili pak programima.

$$\boldsymbol{ako}\ pada\ kiša\ \boldsymbol{onda}\ ulice\ su\ mokre$$

Prednost ovako definiranih pravila je eksplicitno prikazano znanje u ljudima intuitivnom i razumljivom obliku. Međutim, za izgradnju ovakvog sustava potrebno je znanje eksperta za pojedina područja i jako su nefleksibilni kada je riječ o odabiru pravila [20].
Bitno je napomenuti da je za *LCS* sama interpretacija niza znakova u *LCS*-u nebitna. Drugim riječima, niz 00101# može značiti *„ili pada kiša ili puše vjetar“* dok niz 1011 može značiti *„ulice su mokre“*. *LCS* je transparentan za značenje pojedinih nizova znakova – ponaša se kao *crna kutija*. Na ulaz sustava dolazi stanje okoline prikazano kao niz binarnih znakova, dok sustav na izlazu daje niz binarnih znakova. *LCS ne razumije* što pojedini niz znakova predstavlja – jedina informacija koju on dobiva od okoline jest ona o kvaliteti predložene akcije. Sama interpretacija znakova ostavljena je programeru okoline.
U ovisnosti o pojedinim akcijama, okolina nagrađuje sustav. Kako bi sustav s vremenom usvojio dobra pravila, a loša pravila izbacio, potrebno je nekako označiti kvalitetu pojedinog pravila. Zbog toga je, uz uvjet i akciju, u pravilo potrebno dodati i veličinu koja ima značenje korisnosti pravila. Time pravilo poprima oblik *uvjet:akcija→korisnost*. Model *LCS*-a shematski prikazan je na slici 2.4.1.



Slika 2.4.1. Model LCS-a

Proces koji se vrši je iterativan i opisuje se sljedećim koracima:

1. Stanje okoline se pomoću detektora prenosi u listu poruka i to u dio koji namijenjen za pohranjivanje stanja okoline. Lista poruka, uz dio namijenjen za stanje okoline, ima i dio namijenjen za interne poruke te dio namijenjen za akcije. Interne poruke nose informacije o prethodnim stanjima okoline i prethodnim akcijama sustava.
2. Za svako pravilo utvrđuje se odgovara li uvjet trenutnom stanju liste poruka i to na temelju dijela liste poruka namijenjenom za pohranjivanje stanja okoline i dijela za interne poruke. Uvjeti koji odgovaraju trenutnom stanju liste poruka stavljaju se u *podudarni skup (eng. Match set, [M])*. Ukoliko je skup [M] prazan stvara se novo pravilo s uvjetom koji odgovara trenutnom stanju liste poruka.
3. Pravila iz skupa [M] razvrstavaju se u skupine tako da pravila s istom akcijom pripadaju istoj skupini. Na temelju korisnosti svih pravila pojedine skupine predviđaju se nagrade pojedinih akcija i izabire se akcija. Pravila iz skupine iz koje je odabrana akcija smještaju se u *akcijski skup (eng. Action set, [A]).*
4. Akcija se šalje u listu poruka namijenjenu za akcije, a obnavlja se dio liste poruka namijenjen za interne poruke. Efektori šalju akcije okolini.
5. Obnavlja se dio pravila koji određuje korisnost (kvalitetu). Onim pravilima koji ne pripadaju skupu *[A]*, a pripadaju skupu [M] reducira se korisnost. Pravila iz skupa *[A]* i ona koja su u nekoliko prijašnjih koraka pripadala skupu *[A]* međusobno raspodjeljuju korisnost.
6. Primljena nagrada od okoline, koja se periodički šalje sustavu, raspoređuje se (jednoliko) među pravilima skupa *[A]*. Uzevši u obzir i 5. korak, pravila koja prethode pravilima koja su (prije) nagrađivana također poprimaju veću korisnost.
7. Periodički ili s određenom vjerojatnošću pokreće se GA nad cijelim skupom pravila (skup [N]), nad skupom [M] ili nad skupom *[A]*. Najčešće se koristi eliminacijski GA[1], [3].

Ovim modelom opisana je struktura *LCS*-a. Bitna načela koja *LCS* treba slijediti su:

- Pronaći skup pravila koji za što veći broj mogućih uvjeta okoline sadrži pravilo koje donosi dovoljno veliku nagradu okoline. To se posebice odnosi na pravila koja daju akcije za često pojavljivane uvjete okoline.
- Generalizirati skup pravila tako da uvjet jednog pravila odgovara što većem broju stanja u okolini. Pritom treba paziti da se ne izbace specijalizirana pravila koja daju bolje akcije na određeno stanje okoline nego neka generalizirana pravila.
- Ostvariti natjecanje među pravilima, ali samo onima koji imaju odgovarajuće dijelove uvjeta, odnosno mogu odgovoriti na isto stanje okoline. Ostvariti princip da se ulančano nagrađuju pravila, tako da se nagrade i ona pravila čije su akcije prethodile nagrađenim akcijama, a ne samo pravila koja su neposredno uzrokovala nagrađenu akciju [1].

3 Porodice michiganskog stila LCS-a

Kao što je već spomenuto, michiganski stil *LCS*-a se dijeli na tri porodice: *LCS* zasnovan na snazi (*strength-base LCS*), *LCS* zasnovan na preciznosti (*accuracy-based LCS*) i *LCS* zasnovan na očekivanju (*anticipatory-based LCS*).

***LCS* zasnovan na snazi (*strength-based LCS*)** je najjednostavnija porodica michiganskog stila. Korisnost pojedinog pravila je određena skalarnom veličinom zvanom snaga. Snaga predstavlja procjenu nagrade koje će sustav primiti od okoline ako se ponaša u skladu s promatranim pravilom. Pomoću snage se vrši i procjenu pojedinog pravila budući da je to jedina veličina po kojoj se može odrediti kvalitetu pravila. Ovakav pristup i nije baš najbolji. Naime, pravila koja uzrokuju ranije akcije mogu u početku izvršavanja algoritma poprimiti neproporcionalno velike iznose snage [1]. Također, zapostavljena su pravila koja dovode do manje nagrade iako ona mogu u određenoj okolnosti biti najbolji izbor [1]. Naposljetku, najveći problem predstavljaju previše generalizirana pravila koja često vode sub-optimalnom rješenju [2]. Pravila koja su više generalizirana imaju veću vjerojatnost pojavljivanja u skupu akcija (skup *[A]*) pa im snage poprimaju znatno veće vrijednosti nego manje generaliziranim pravilima iako im predviđanje nagrada okoline može biti manje precizno.
Ipak, za mnoge aplikacije ova porodica *LCS*-a je sasvim prihvatljiva. Najpoznatiji predstavnik ove porodice je *ZCS*.

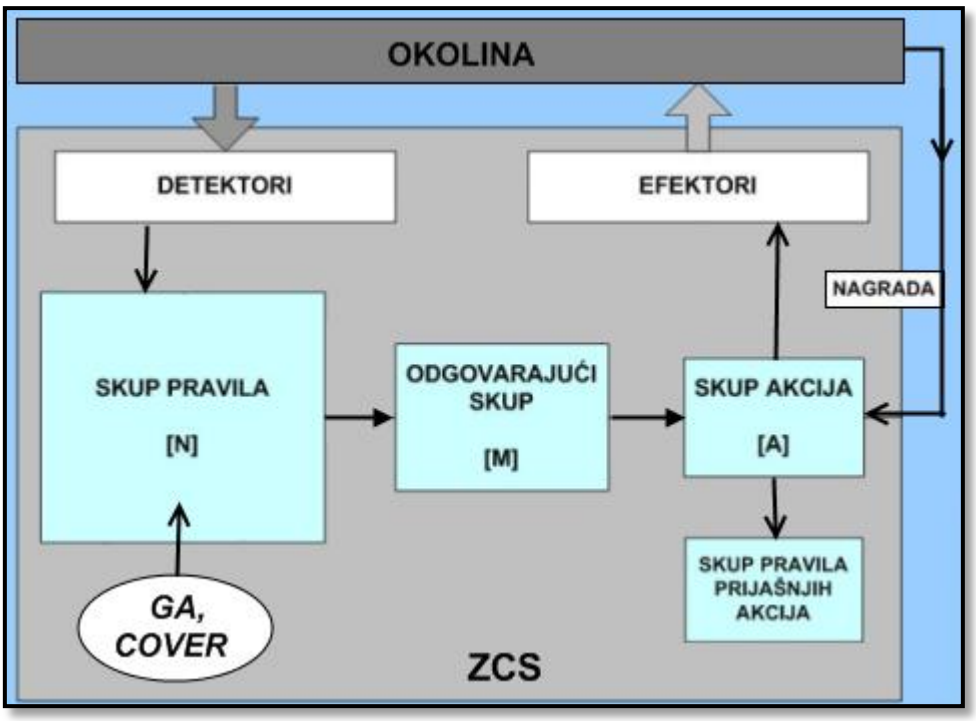
***LCS* zasnovan na preciznosti (*accuracy-based LCS*)** nadopunjuje korisnost pravila sa procjenom pogreške koju pravilo čini pri procjeni nagrade. Kvalitetu pravila više ne određuje snaga, već nova veličina povezana sa preciznošću pravila i to tako da su preciznija pravila kvalitetnija. Iako je sustav postao složeniji, uvođenjem spomenutih parametara sprječava se prevelika generalizacija pravila, a time je i prostor mogućih pravila učinkovitije prekriven. Najpoznatiji predstavnik ove porodice *LCS*-a je *XCS*.

***LCS* zasnovan na očekivanju (*anticipatory-based LCS*)** prikazuje pravila u obliku *uvjet:akcija→očekivanje* pri čemu *očekivanje* predstavlja očekivano stanje okoline nakon primijenjene *akcije* na okolinu stanja *uvjeta*. Ovakva vrsta pravila (klasifikatora) gradi *model prijelaza* (eng. *model of transitions*) [2]. *Očekivanje* može, ovisno o verzijama *LCS*-a, sadržavati i posebne znakove = i ?. Znak = označava da se atribut stanja okoline ne mijenja, dok znak ? označava da se atribut stanja ne može predvidjeti [2]. Primjerice, ako na stanje okoline 101 djelujemo akcijom 0, tada pravilo #01:0→=1= predviđa stanje okoline 111. Znak *?* je sličan znaku # (*don't care*) koji se pojavljuje u uvjetnom dijelu pravila.
Na oblik pravila *uvjet:akcija→očekivanje* dodaju se parametri koji predstavljaju kvalitetu pojedinog pravila. Kvaliteta jedinke (pravila) je određena preciznošću predviđanja sljedećeg stanja okoline [15]. Jedan od poznatijih predstavnika ove porodice *LCS*-a je ACS2.

3.1 Klasifikatorski sustav ZCS

Najpoznatiji predstavnik *LCS*-a zasnovanih na snazi *(eng. Fitness)* je ***ZCS*** [1]. Kao i osnovni model *LCS*-a, opisao ga je Wilson. Naziv mu dolazi od imena „*zeroth level LCS*“. Naime, u svrhu pojednostavljenja sustava i poboljšanja performansi iz sustava je u potpunosti uklonjena lista poruka, a odabir akcije i *učenje potkrjepljenjem* vrši se nad skupom pravila. Za razliku od Hollandovog sustava, u akcijskom dijelu pravila nisu dozvoljeni *don't care* znakovi.

Nakon detekcije stanja okoline sva se odgovarajuća pravila smještaju u *podudarni skup [M]*. Nadalje, iz podudarnog se skupa odabire pravilo koje ima najveću snagu – ono se sustavu čini kao trenutno najbolji izbor. Sva pravila iz podudarnog skupa koja imaju istu akciju kao i navedeno pravilo smještaju se u *akcijski skup*. Okolini se preko *efektora* prosljeđuje akcija definirana jedinkama u akcijskom skupu, i stanje okoline S mijenja se u stanje S^{+1}. Model *ZCS*-a prikazan je na slici 3.1.



Slika 3.1 Model ZCS-a

Učenje potkrjepljenjem realizira se pomoću varijante *algoritma trenutne razlike (eng. Temporal Difference, TD). U s*vakom se koraku pamti trenutni akcijski skup, a određeni dio snage svakog pravila iz akcijskog skupa stavlja se u interni spremnik. Ukoliko sustav dobije nagradu od okoline ona se jednoliko raspoređuje na snagu svakog klasifikatora u trenutnom akcijskom skupu. Ključ procesa učenja je nagrađivanje, odnosno kažnjavanje pravila koja su dovela do stanja okoline $\boldsymbol{S^{+1}}$. Snaga pravila iz prethodnog akcijskog skupa ažurira se sa ukupnom snagom u internom spremniku.

Ažuriranje snage opisano je sljedećom jednadžbom:

$$rule.F = rule.F + \beta\,(reward + \gamma * bucket - rule.F\,), \forall rule \in [A] \qquad (3.1)$$

gdje su β i γ predefinirane konstante učenja, *F* snaga pravila te

$$bucket = \sum pravilo.F\,, \forall pravilo \in [A]^{+1} \qquad (3.2)$$

Nadalje, svim se pravilima koja su u podudarnom skupu pravila *[M],* a nisu u akcijskom skupu *[A]* smanjuje snaga za faktor τ da bi se u sljedećim koracima smanjila šansa za odabir istih.

Napredovanje sustava prema skupu pravila koja maksimiziraju ukupnost nagrada iz okoline *ZCS* primjenjuje dvije heurističke tehnike generiranja novih jedinki – genetski algoritam i *cover* algoritam.

U svakom se koraku s određenom vjerojatnošću nad čitavom populacijom pokreće eliminacijski GA. Koristeći *jednostavnu selekciju (eng. Roulette wheel selection)*, a uzimajući u obzir snagu pravila, odabiru se dva roditelja koja stvaraju dva potomka, koji se potom mutira. Potomci se smještaju u *skup svih pravila [N]*, a snaga im je određena polovinom snage roditelja. Naposljetku, dvije jedinke odabrane *jednostavnom selekcijom*, uzimajući u obzir recipročnu vrijednost snage, uklanjaju se iz skupa pravila *[N]*.
Cover operator u populaciju dodaje pravilo koje odgovara trenutnom stanju okoline, a čija se akcija bira nasumce. Poziva se kada u podudarnom skupu nema pravila ili kad je prosječna snaga svih pravila u podudarnom skupu manja od određene vrijednosti.

```
RADI {
        stanje = okolina: stanje
        [M] = GENERIRAJ PODUDARNI SKUP
        AKO ( |M| = ∅ ILI pravila nisu dovoljno dobra)
                POZOVI COVER ALGORITAM
        act = ODABER AKCIJU koristeći jednostavnu selekciju
        [A] = GENERIRAJ AKCIJSKI SKUP koristeći [M] i act
        nagrada = rp: nagrada
        AŽURIRAJ SNAGU PRAVILA
        POZOVI GENETSKI ALGORITAM nad [M] s vjerojatnošću P_GA
        [A]_-1 = [A]
} DOK ( uvjet zaustavljanja nije ispunjen )
```

Slika 3.2 Pseudokod rada ZCS-a

Prve primjene *ZCS* sustava bile su na području otkrivanja znanja iz skupa medicinskih podataka te na području modeliranja agenata za simulaciju ekonomskog tržišta. Iako je *ZCS*

jako osjetljiv na neke parametre, rezultati su pokazali kako može doseći optimalne performanse. Međutim, u upotrebi ga je potpuno zasjenio njegov nasljednik – *XCS*.

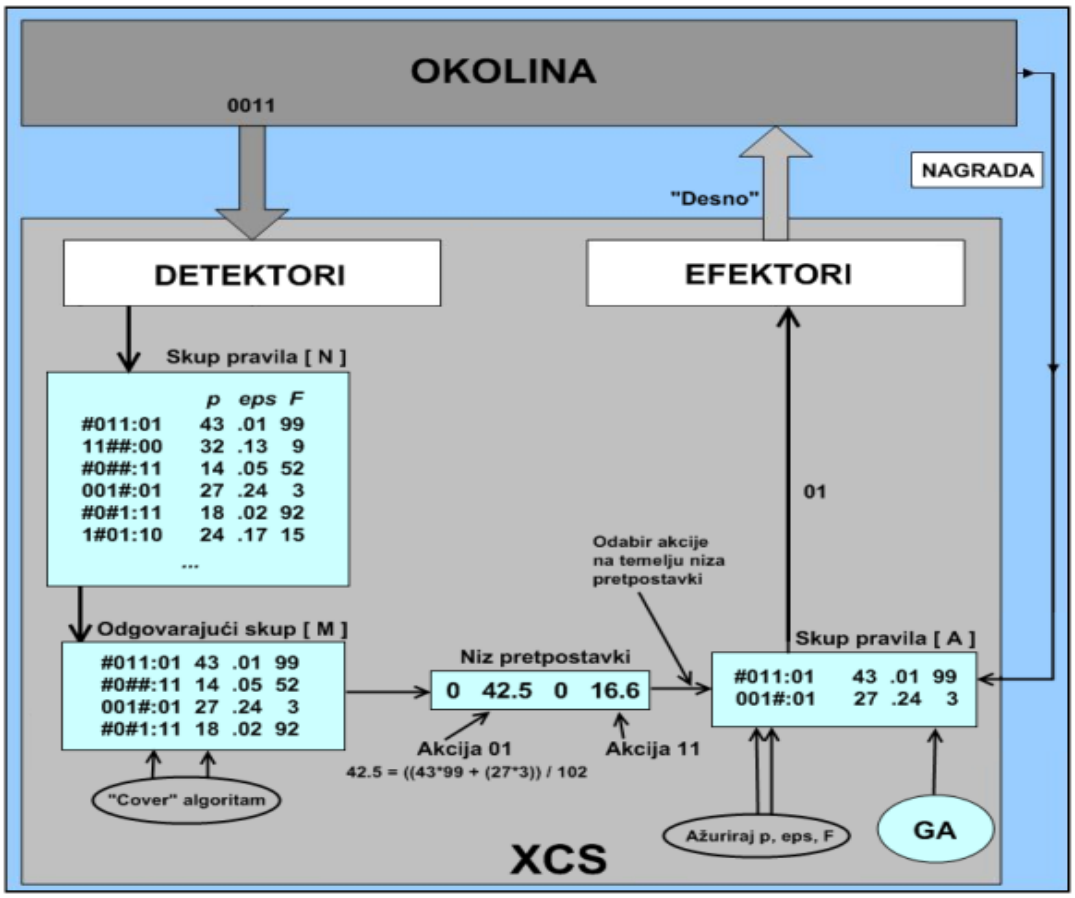
3.2 Klasifikatorski sustav XCS

Nekoliko godina nakon predstavljanja *ZCS* sustava Willson je predstavio novi sustav – *XCS (eng. eXtensible Classifier System).* Glavna razlika u odnosu na *ZCS* jest u tome da snaga pravila nije temeljena samo na predviđenoj nagradi nego i na preciznosti tog predviđanja. Cilj sustava je potpuno i precizno preslikavanje stanja okoline na skup akcija. Koristeći efikasnu generalizaciju pravila kroz *učenje potkrjepljenjem* sustav može riješiti kompleksne probleme u kojima je broj kombinacija stanja i mogućih akcija jako velik. Prednost *XCS*-a su poboljšanje performanse te mogućnost kreiranja maksimalnih generalizacija.

Svako pravilo u sustavu uz uvjet i akciju ima dodatna tri parametra : ε, p i P. Parametar ε predstavlja grešku u predviđanju, p predviđenu nagradu od okoline ukoliko se ovo pravilo iskoristi, a P predstavlja snagu klasifikatora. Koristeći navedene parametre sustav određuje kvalitetu pojedinog pravila prilikom selekcije za GA.

Kao i kod *ZCS* sustava, u svakom se koraku kreira podudarni skup *[M]*. Ukoliko je podudarni skup prazan, koristi se *cover* operator. Uzevši u obzir akcije koje predlažu pravila podudarnog skupa kreira se lista predviđanja. Svaki element liste predstavlja predviđanje kvalitete pojedine akcije na temelju parametara pravila podudarnog skupa koja tu akciju predlažu. Na temelju niza vrijednosti iz liste predviđanja sustav može deterministički ili stohastički izabrati akciju. S ciljem boljeg pretraživanja problemskog prostora sustav koristi „istraži-iskoristi“ tehniku *(eng. Explore-exploit strategy).* U jednom koraku sustav odabire akciju s najboljim predviđanjem dok u drugom nasumce odabire neku od preostalih akcija iz liste predviđanja. Koristeći takav ne pohlepan pristup sustav uspješno izbjegava lokalne minimume.

Nakon odabira, akcija se preko *efektora* šalje okolini, a sva pravila iz podudarnog skupa koja predlažu tu akciju smještaju se u akcijski skup. Za razliku od *ZCS*-a genetski algoritam vrši selekciju roditelja samo nad akcijskim skupom. Ukoliko je proteklo vrijeme od posljednjeg djelovanja GA nad elementima trenutnog akcijskog skupa prešlo određenu granicu, vrši se GA. Nakon selekcije dvaju roditelja na temelju snage stvaraju se dvoje djece koja zamjenjuju neka odabrana dva pravila iz populacije. Model *XCS* sustava prikazan je na slici 3.3. Bitno je primijetiti da se GA vrši nad trenutnim akcijskim skupom.



Slika 3.3 Model XCS sustava

Učenje potkrjepljenjem sastoji se od ažuriranja parametara pravila prema relativnoj preciznosti pojedinog pravila unutar skupa pravila kroz sljedećih pet koraka:

1. Ažuriranje pogreške predviđanja:

$$\varepsilon_j = \varepsilon_j + \beta\left(\left|P - p_j\right| - \varepsilon_j\right) \tag{3.3}$$

2. Ažuriranje predviđanja

$$p_j = p_j + \beta\,(P - p_j) \tag{3.4}$$

3. Ažuriranje preciznosti pravila K_j:

$$K_j = \begin{cases} \alpha\left(\frac{\varepsilon_0}{\varepsilon}\right), \varepsilon \geq \varepsilon_0 \\ 1, \qquad \varepsilon < \varepsilon_0 \end{cases} \tag{3.5}$$

4. Ažuriranje relativne preciznost pravila K_j:

$$K_j' = \frac{K_j}{\sum K_i}\ \forall i \tag{3.6}$$

5. Ažuriranje snage pravila koristeći *MAM postupak* i relativnu preciznost:

$$F_j = \begin{cases} F_j + \beta(K_j' - F_j), & \text{snaga ažurirana } \frac{1}{\beta} \text{ puta} \\ \frac{K_j + K_j'}{2}, & \text{inače} \end{cases} \quad (3.7)$$

Naposljetku, najveća vrijednost u nizu predviđanja umanjena za određeni faktor koristi se za ažuriranje pravila iz prethodnog akcijskog skupa. Prethodno opisana verzija učenja s potkrjepljenjem poznata je pod nazivom Q-učenje *(eng. Q-Learning)* [3].

```
prethodnaNagrada = 0;
prethodnoStanje = 0;
RADI {
        stanje = okolina: stanje
        [M] = GENERIRAJ PODUDARNI SKUP
        PA = GENERIRAJ NIZ PREDVIĐANJA koristeći M
        act = ODABER AKCIJU koristeći PA
        [A] = GENERIRAJ AKCIJSKI SKUP koristeći [M] i act
        nagrada = rp: nagrada

        AKO ( [A]_-1 ≠ ∅ )
                P = prethodnaNagrada + γ * max(PA)
                AŽURIRAJ SKUP [A] − 1 koristeći P
                POKRENI GENETSKI ALGORITAM nad [A]_-1 koristeći stanje_-1
        AKO ( rp: problem riješen )
                P = NAGRADA
                AŽURIRAJ SKUP [A]koristeći P
                [A]_-1 = ∅
        INAČE
                [A]_-1 = [A]
                prethodnaNagrada = nagrada
                prethodnoStanje = stanje;

} DOK ( uvjet zaustavljanja nije ispunjen )
```

Slika 3.4 Pseudokod rada XCS sustava

Kompleksnost *XCS* sustava leži u prilagođavanja mnoštva parametara pojedinom problemu. Unatoč toj činjenici većina klasifikatorskih sustava gradi se upravo na *XCS* modelu. Više o samoj primjeni *XCS*-a i drugih klasifikatorskih sustava bit će riječi u sljedećem poglavlju.

4 Primjena LCS-a

4.1 Područja primjene

LCS se uglavnom koristi u rješavanju jednog od ova četiri tipa problema:

- klasifikacijski problemi *(eng. classification problems)*
- problemi učenja potkrjepljenjem *(eng. reinforcement learning problems)*
- problemi aproksimacije funkcija *(eng. function approximation problems)*
- problemi predviđanja *(eng. prediction problems)*

Klasifikacijski problemi predstavljaju probleme čiji je cilj raspoređivanje (klasificiranje) skupine objekata u podskupine (klase) i određivanje pravila po kojima se objekti raspoređuju. *LCS* koji se primjenjuje na klasifikacijske probleme nastoji pronaći takav skup pravila po kojima se klasificiranje obavlja što preciznije. Klasifikacijski problemi tipični su za područje otkrivanja znanja u skupovima podataka.

Problemi učenja potkrjepljenjem svode se na pronalaženje optimalne funkcije ponašanja, a u *LCS*-u je ta funkcija predstavljena skupom pravila. Jedan od primjera problema učenja potkrjepljenjem je problem labirinta (*maze problem*).

Cilj *LCS*-a koji se primjenjuje nad problemima aproksimacije funkcija je što točnija aproksimacija funkcije skupom pravila koja se djelomično preklapaju. Primjer takvog problema je (problem) aproksimacija sinusne funkcije linearnim dijelovima [11].

LCS se može primijeniti u rješavanju gotovo svih problema predviđanja. Tako je *LCS* uspješno implementiran i u problemu predviđanja koordinacijskog broja amino kiseline u proteinskoj strukturi.

Od područja primjene *LCS-a* svakako treba spomenuti *otkrivanje znanja u skupovima podataka, upravljanje procesima* te *modeliranje i optimizaciju* [3]. Primjer nekoliko aplikacija koje upotrebljavaju *LCS* dan je u tablici 4.1.

Tablica 4.1 Područja primjene LCS-a

Područja primjene *LCS*-a	Primjeri aplikacija
Otkrivanje znanja u skupovima podataka	Klinička istraživanja: procjenjivanje opasnosti ozljede analizom podataka o preživljavanju te ozljede
	Klasifikacija gljiva
Upravljanje procesima	Kontrola protoka u plinskim cjevovodima
	Kontrola signalizacije na prometnim raskrižjima
	Distribuirano usmjeravanje u komunikacijskim mrežama
Modeliranje i optimizacija	Modeliranje ponašanja brokera
	Navigacija robota

4.2 Single step i multi step problemi

Problemi koji se rješavaju klasifikatorskim sustavom podijeljeni su u dvije skupine: *single step* i *multi step* problemi.
Single step problemi su nešto jednostavniji i od sustava zahtijevaju da ispravno klasificira stanje na ulazu. Tipičan primjer „single step“ problema je određivanje izlaza multipleksora za zadani ulaz *(eng. Mux problem)*. Problem multipleksora sa 6 ulaza je klasifikacijski problem u kojem je cilj za bilo koji 6-bitni ulaz dati izlaznu vrijednost (0 ili 1) koju bi dao i multipleksor 4/1. Multipleksor 4/1
ima 4 podatkovna i 2 adresna ulaza (ukupno 6 ulaza), a izlaz je definiran vrijednošću onog podatkovnog ulaza kojeg adresira adresni dio ulaza. Klasifikacija LCS-om zahtjeva određivanje ispravnog izlaza iz multipleksora koji će se usporediti sa akcijom sustava.

Problemi u kojima je sljedeće stanje okoline izravno ovisno o trenutnoj akciji sustava, a do rješenja problema se dolazi kroz nekoliko koraka nazivaju se *multi step* problemi. Budući da akcije iz prethodnih koraka indirektno utječu na buduće akcije sustav mora poduzeti nekoliko uzastopnih ispravnih akcija kako bi došao do rješenja. Tipičan *multi step* problem je problem *animata (eng. Artificial Animal). Animat* je u potrazi za hranom, a cilj mu je u što manje koraka doći do hrane. Zadaća sustava je na temelju njegove trenutne pozicije odrediti smjer gibanja.

5 Logičko – kombinatorne igre

5.1 Kratak uvod u teoriju igara

Teorija igara je grana primijenjene matematike koja se bavi situacijama konflikata između dvaju ili više sudionika. Dijeli se na dvije grane: kooperativnu i nekooperativnu teoriju igara. Nekooperativna teorija igara bavi se igrama čija su pravila precizno definirana i potpuna, temeljne jedinice odluke su individualni igrači te nije moguće stvaranje kompromisa. Cilj teoretskog razmatranja je odrediti strategiju ponašanja sudionika koja je za njih najpovoljnija, pod pretpostavkom da su sudionici racionalni. Konflikt među sudionicima reguliran je strogo definiranim pravilima kao u društvenim igrama poput pokera, "Čovječe ne ljuti se" i sličnim. Teorija igara omogućuje preciznu analizu takvih igara, određivanje pobjedničkih strategija i određivanje postojanja rješenja. Naravno, pod nazivom *igre* ne misli se samo na društvene igre niti sudionici moraju biti osobe.
Zbog činjenice da konflikti u stvarnom životu nisu podložni jednostavnim pravilima teorija igara je uglavnom ostala u okvirima akademskih primjera dok se u praksi dosta rijetko koristi. Međutim, radi se o matematičkoj teoriji koja povezuje nekoliko grana matematike i daje važne doprinose razumijevanju ponašanja u ekonomiji, psihologiji, sociologiji i teoriji evolucije [21][22].

Matematički model igre temelji se na skupu igrača, skupu poteza koje ti igrači mogu napraviti i definiciji funkcije nagrade za pojedine strategije ili ishode. Skup igrača u igrama koje će biti promatrane sastoji se od dvaju igrača. Skup poteza kao i funkcija nagrade bit će eksplicitno definirani za pojedinu igru.
Svaku igru možemo igrati bez neke očite strategije, npr. nasumičnim odabirom poteza. Međutim, ako je cilj pobijediti, to možda i nije najpametnija strategija. U teorijskoj razradi igara pretpostavlja se da igrači igraju *optimalno* – žele pobijediti.

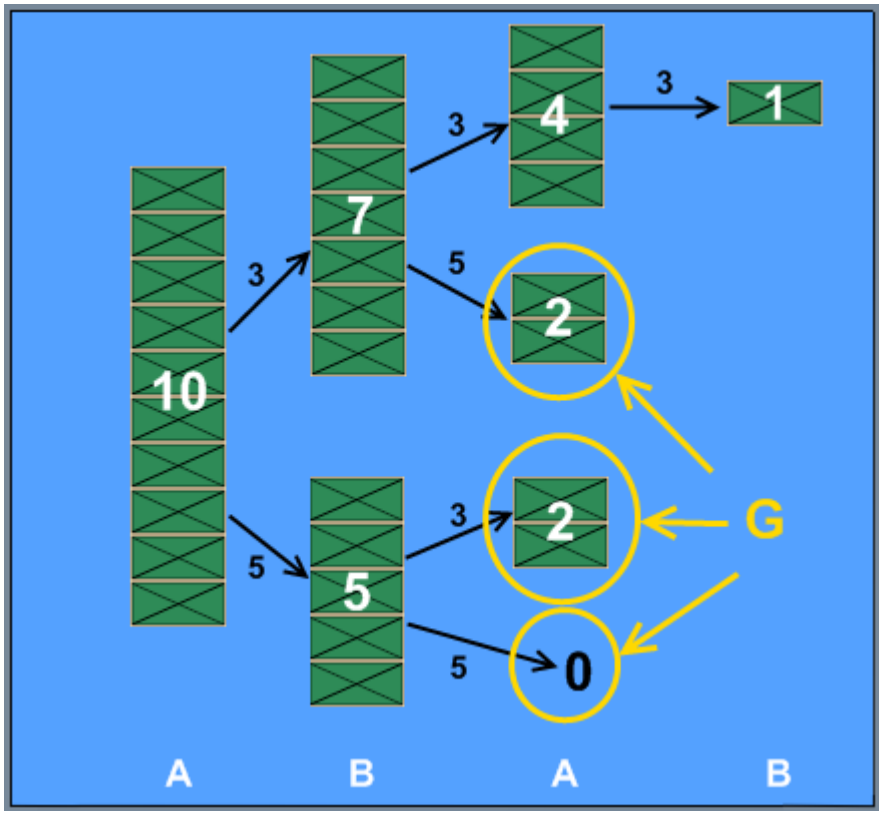
U igrama koje će biti promatrane svaka pozicija je pobjednička ili gubitnička – igre ne mogu završiti neodlučeno. Za igre koje imaju navedeno svojstvo kaže se da su *determinirane*.[22]
Pozicija je definirana kao pobjednička ukoliko igrač koji je trenutno na potezu može pobijediti neovisno o sljedećim potezima protivnika. Stanja igre koja nemaju to svojstvo nazivaju se *gubitničkim pozicijama*. Potezi koji iz pobjedničke pozicije dovode u gubitničku poziciju nazivaju se *ispravni* ili *korektni potezi*. S druge strane, potezi koji iz pobjedničke pozicije dovode u pobjedničku poziciju nazivaju se *neispravni* ili *nekorektni potezi*.

Ukoliko se problem može reducirati na linearnu funkciju više varijabli uz uvjete zadane linearnim nejednakostima, rješenje se može naći linearnim programiranjem. Postoji više efikasnih algoritama za rješavanje takvih problema, među kojima je najpopularniji *simplex* algoritam. Međutim, većina problema se ne može reducirati na navedeni oblik, a velika većina nije determinirana. U mnogim igrama je broj mogućih strategija prevelik za eksplicitno računanje i vrednovanje. Za rješavanje većine problema određivanja optimalnih strategija koriste se složeniji postupci u kojima se kombiniraju pretraživanja problemskog prostora sa raznim heuristikama.

U ovom poglavlju bit će opisane dvije relativno jednostavne logičko-kombinatorne igre. Bit će prikazane pobjedničke strategije u ovisnosti o stanjima, kao i pogled s aspekta klasifikatorskih sustava.

5.2 Kutije

Igra „*Kutije*“ je logičko-kombinatorna igra konstruirana kako bi zadovoljavala prethodno opisan model. Riječ je o igri u kojoj dva igrača izmjenjuju poteze uklanjajući *Kutije* sa hrpe. Uklanjanje tri ili pet kutija sa hrpe smatra se potezom. Igrač koji napravi posljednji potez je pobjednik.
Broj preostalih kutija na hrpi u danom trenutku predstavlja stanje okoline(poziciju). U igri postoje pobjedničke i gubitničke pozicije. Za zadanu poziciju moguće je odrediti postoji li strategija za koju igrač koji je trenutno na potezu pobjeđuje. Na slici 5.1 prikazana je jedna gubitnička pozicija.



Slika 5.1 Gubitnička pozicija u igri Kutije

U igri prikazanoj na slici igrači *A* i *B* nakon nekoliko poteza dolaze u stanje 10. Na potezu je igrač *A*. Kako svaki potez dovodi igrača *B* u pobjedničku poziciju ova pozicija je gubitnička za igrača koji trenutno igra. Drugim riječima, neovisno o potezu igrača *A*, igrač *B* ima pobjedničku strategiju. Primjerice, ukoliko igrač *A* ukloni pet kutija sa hrpe igrač *B će ukloniti* ili tri ili pet i pobijediti. Ukoliko pak *A* ukloni tri *Kutije B će ukloniti* pet i pobijediti. U tablici 5.1 prikazani su podatci za prvih 30 stanja igre. U prvom i drugom stupcu nalazi se stanje igre

u dekadskom odnosno binarnom zapisu. U trećem stupcu nalazi se tip pozicije – pobjednička ili gubitnička. U četvrtom i petom stupcu nalaze se ispravni potezi za pojedinu poziciju.

Tablica 5.1 Prvih 30 stanja igre Kutije

Stanje	Stanje - Binarno	Pozicija	Akcija	
0	00000	gubitnička	/	/
1	00001	gubitnička	/	/
2	00010	gubitnička	/	/
3	00011	pobjednička	3	/
4	00100	pobjednička	3	/
5	00101	pobjednička	3	5
6	00110	pobjednička	3	5
7	00111	pobjednička	/	5
8	01000	gubitnička	/	/
9	01001	gubitnička	/	/
10	01010	gubitnička	/	/
11	01011	pobjednička	3	/
12	01100	pobjednička	3	/
13	01101	pobjednička	3	5
14	01110	pobjednička	/	5
15	01111	pobjednička	/	5
16	10000	gubitnička	/	/
17	10001	gubitnička	/	/
18	10010	gubitnička	/	/
19	10011	pobjednička	3	/
20	10100	pobjednička	3	/
21	10101	pobjednička	3	5
22	10110	pobjednička	/	5
23	10111	pobjednička	/	5
24	11000	gubitnička	/	/
25	11001	gubitnička	/	/
26	11010	gubitnička	/	/
27	11011	pobjednička	3	/
28	11100	pobjednička	3	/
29	11101	pobjednička	3	5

Iz tablice 5.1 može se očitati da je stanje okoline 16 gubitničko, dok je stanje 29 pobjedničko. Za stanje okoline 29 ispravna akcija je uklanjanje tri ili pet kutija. Za stanje okoline 20 ispravna akcija je uklanjanje tri kutije.

Za otkrivanje oblika ispravnih klasifikatora potrebno je primijetiti da se akcije pojavljuju u ciklusu od 8. Primjerice, ukoliko je stanje okoline 6, 14, 22 ili 30 ispravan potez je ukloniti tri kutije. Analogno, stanja okoline 2, 10, 18 i 26 su gubitnička. Bitno za napomenuti je da se isto razmišljanje može proširiti i na skup od proizvoljno mnogo kutija.

Pobjednička strategija za igrača koji trenutno igra je intuitivna:

- Trenutna pozicija je pobjednička - odigraj potez koji dovodi protivnika u gubitničku poziciju

- Trenutna pozicija je gubitnička – odigraj bilo koji potez

Ukoliko N označava trenutno stanje igre, a A ostatak cjelobrojnog dijeljenja broja N sa 8 ispravan potez može se dobiti na sljedeći način:

- $A = N \% 8$
  - $ako\ A = 3\ ili\ A = 4\ ,\ ukloniti\ 3\ kutije\ sa\ hrpe$
  - $ako\ A = 5\ \ ,\ ukloniti\ ili\ 3\ ili\ 5\ kutija\ sa\ hrpe$
  - $ako\ A = 6\ ili\ A = 7\ ,\ ukloniti\ 5\ kutija\ sa\ hrpe$

Igra Kutije s aspekta klasifikatorskih sustava

Iako lanac akcija dovodi nekog igrača do pobjede, ovaj problem ipak nije *multi step* problem. Naime, stanje okoline ne ovisi samo o akcijama klasifikatorskog sustava nego se mijenja i u ovisnosti o akcijama drugih igrača. Bitno je primijetiti kako su akcije u pobjedničkom lancu akcija ispravni potezi za trenutnu poziciju. Prema tome, igra se može podijeliti u *pod-igre* koje su *single step* problemi. Jedna pod-igra, odnosno jedan *single step* problem bit će određivanje ispravne akcije za zadano stanje okoline.
Trenutno stanje igre, odnosno broj kutija koji se nalazi na hrpi predstavlja stanje okoline. Za svaku pobjedničku poziciju postoji ili jedna ili dvije optimalne akcije koje sustav treba naučiti. U gubitničkim pozicijama akcija sustava nije relevantna.

Za određivanje ispravnog poteza dovoljno je promatrati ostatak pri cjelobrojnom dijeljenju broja kutija na hrpi sa 8. Prema tome, rješenje problema potpuno je opisano sa osam klasifikatora koji generaliziraju sva stanja okoline – po jedan za svaku od prvih osam pozicija u igri. Bitno je napomenuti kako broj potrebnih klasifikatora ne ovisi o maksimalnom broju kutija na hrpi. U tablici 5.2 prikazani su navedeni klasifikatori oblika „*ako uvjet onda akcija*“ gdje uvjet predstavlja stanje okoline zapisan u binarnom obliku, a akcije 0 i 1 uklanjanje 3 odnosno 5 kutija sa hrpe.

Tablica 5.2 Ispravni klasifikatori za igru Kutije

Pravila	
Uvjet	*Akcija*
##000	0 ili 1
##001	0 ili 1
##010	0 ili 1
##011	0
##100	0
##101	0 ili 1
##110	1
##111	1

Tablica je podijeljena na dva dijela. Pravila iz prvog dijela odgovaraju gubitničkim pozicijama dok su u drugom pravila koja odgovaraju pobjedničkim pozicijama. Za očekivati je da će navedena pravila imati najveću snagu u populaciji klasifikatorskog sustava.

5.3 Nim

Nim je jedna od najstarijih i najpoznatijih matematičko-strateških igara. Ime igre dolazi od njemačke riječi *„nehmen“* što znači *uzeti*. U igri dva igrača izmjenjuju poteze uklanjajući objekte sa različitih hrpa. U svakom potezu mora se ukloniti barem jedan objekt.
Igru je moguće igrati na *normalan* i *mizerni* (*fra. misère*) način**.** U *normalnom* načinu igranja onaj igrač koji ukloni posljednji objekt pobjeđuje. U *mizernom* načinu gubi onaj igrač koji ukloni posljednji objekt.
Kao i igra *Kutije*, *Nim* je determinirana igra. Međutim, do pobjedničke strategije se dolazi na nešto teži način. Glavna razlika koju treba primijetiti u odnosu na igru *Kutije* je da se u ovoj igri radi o nekoliko hrpa, a na svakoj hrpi može biti različiti broj objekata. Osim toga, broj mogućih akcija je sada drastično veći.

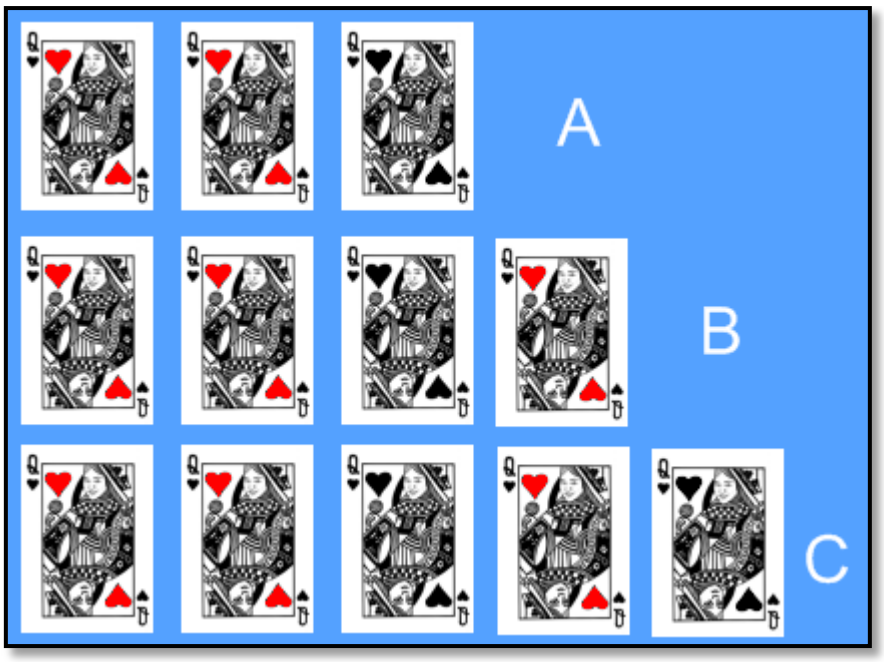
U nastavku teksta razmatrat će se *normalan* način igre. Varijanta Nim igre s X hrpa i najviše Y objekata na hrpi označavat će se s *Nim X/Y*. Stanje igre označavat će se kao neuređene *n-torke*, gdje je *n* broj hrpa. Igra s primjerice 3 hrpe te *A* karata na prvoj, *B* karata na drugoj i *C* karata na trećoj hrpi bit će označena kao neuređena trojka *(A, B, C)*. Svaka pozicija dobivena permutacijom članova neuređene *n-torke* odgovara jednako vrijednoj poziciji. Drugim riječima, ukoliko je pozicija *(A,B,C)* pobjednička, pobjedničke su i pozicije *(A,C,B)*, *(B,A,C)*, *(B,C,A)*, *(C,A,B)* te *(C,B,A)*. U ostatku poglavlja promatrat će se *normalan* način igre.

Binarna suma broja objekata na svakoj hrpi uz zanemarivanje prijenosa prema višim bitovima naziva se *nim-suma*. Drugi naziv je *ekskluzivni ili*, odnosno vektorski zbroj nad *Galoisovim poljem(2) (eng. Galois Field)*. Kako bi se *nim-suma* razlikovala od obične sume, oznaka za *nim-sumu x i y* bit će $x \oplus y$.

Za određivanje pobjedničke strategije potrebno se pozvati na teorem kojeg je 1901. godine postavio *Charles L. Bouton*.

Teorem. U normalnoj *Nim* igri prvi igrač ima pobjedničku strategiju ako i samo ako je *nim-suma* broja objekata na hrpama različita od nule. U suprotnom, drugi igrač ima pobjedničku strategiju.

Dokaz *teorema* neće biti naveden iako se *teorem* relativno jednostavno dokazuje indukcijom po broju objekata na pojedinoj hrpi nakon odgovarajućeg poteza. Prema *teoremu*, ukoliko je igrač u pobjedničkoj poziciji mora odigrati potez kojim će *nim-suma* postati nula.



Slika 5.2 Pobjedničko stanje u igri Nim

Na slici 5.2. prikazano je pobjedničko stanje u igri *Nim 3/5*. Ukoliko se *nim-sumu* svih hrpa označi s X, a broj karata na pojedinoj hrpi s *A*, *B* odnosno *C*, *nim-suma* svih hrpa je:

$$X = A \oplus B \oplus C = 3 \oplus 4 \oplus 5 = 2 \qquad (5.1)$$

Budući da je *nim-suma* različita od nule, prema *teoremu* se može zaključiti kako je riječ o pobjedničkoj poziciji. Sljedeći korak u određivanju ispravnog poteza je računanje *nim-sume* početnih vrijednosti za svaku hrpu i prethodno izračunate *nim-sume* svih hrpa.

$$\begin{aligned} A' &= A \oplus X = 3 \oplus 2 = 1 \\ B' &= B \oplus X = 4 \oplus 2 = 6 \\ C' &= C \oplus X = 5 \oplus 2 = 7 \end{aligned} \qquad (5.2)$$

Potrebno je primijetiti da se, ukoliko je riječ o pobjedničkoj poziciji, barem jedna od vrijednosti *A'*, *B' odnosno C'* mora smanjiti u odnosu na odgovarajuću početnu vrijednost. Pobjednička strategija svodi se na uklanjanje određenog broja objekata upravo s te hrpe. Novi broj karata na hrpi mora biti *nim-suma* početne veličine i X.
U gornjem primjeru jedina hrpa kojoj se veličina smanjila u odnosu na početnu je prva hrpa. Početna veličina hrpe je tri. Kako na hrpi mora ostati jedna karta, ispravan potez je za ovu poziciju ukloniti dvije karte s prve hrpe.
Za provjeru ispravnosti poteza računa se nova vrijednost *nim-sume X'*. Za očekivati je da će *X'* biti nula. Broj karata na prvoj hrpi je sad $A' = 1$ dok se brojevi karata na drugoj i trećoj hrpi nisu promijenili.

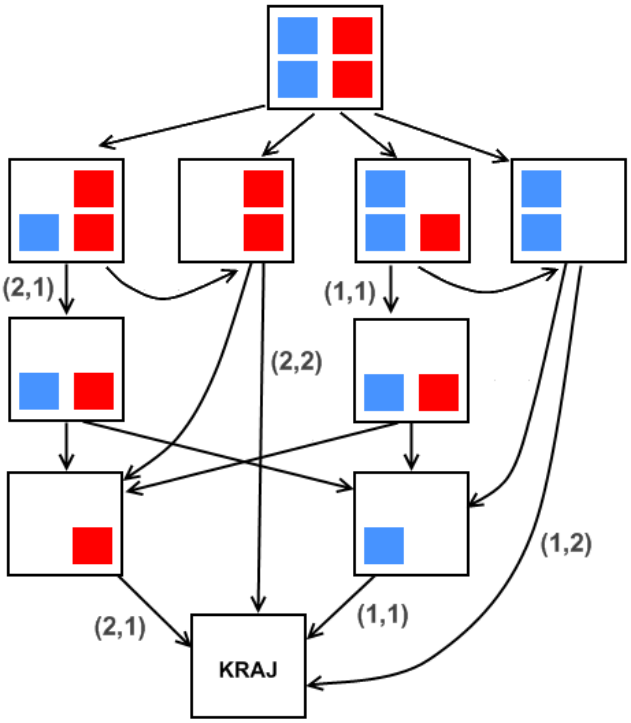
$$X' = A' \oplus B \oplus C = 1 \oplus 6 \oplus 7 = 0 \qquad (5.3)$$

Prema *(5.3)* klasa pozicija *(3,4,5)* je pobjednička. Pomoću *(5.2)* moguće je odrediti ispravan potez. U tablici 5.3 prikazana su stanja igre *Nim 2/3*. U prvom i drugom stupcu nalazi se broj objekata na odgovarajućoj hrpi. U trećem stupcu je stanje igre u binarnom zapisu – po dva bita za svaku hrpu. U četvrtom stupcu nalazi se tip stanja. Ispravna akcija za pojedino stanje navedena je u petom stupcu.

Tablica 5.3 Stanja igre Nim 2/7

Hrpa 1.	Hrpa 2.	Binarno	Pozicija	Ispravna akcija
0	0	0000	gubitnička	/
0	1	0001	pobjednička	(2,1)
0	2	0010	pobjednička	(2,2)
0	3	0011	pobjednička	(2,3)
1	0	0100	pobjednička	(1,1)
1	1	0101	gubitnička	/
1	2	0110	pobjednička	(2,1)
1	3	0111	pobjednička	(2,2)
2	0	1000	pobjednička	(1,2)
2	1	1001	pobjednička	(1,1)
2	2	1010	gubitnička	/
2	3	1011	pobjednička	(2,1)
3	0	1100	pobjednička	(1,3)
3	1	1101	pobjednička	(1,2)
3	2	1110	pobjednička	(1,1)
3	3	1111	gubitnička	/

Iz podataka u tablici može se očitati kako je pobjednička strategija u igri s dvije hrpe pratiti poteze drugog igrača. Naime, *nim-suma* će biti nula čim su hrpe jednake veličine. Primjerice, za stanje igre *(2,3)* potrebno je ukloniti jedan objekt s druge hrpe i tako dovesti protivnika u gubitničko stanje *(2,2)*.
Na slici 5.3 prikazan je graf stanja svih poteza za poziciju (2,2) te su istaknuti ispravni potezi za drugog igrača.



Slika 5.3 Ispravni potezi za poziciju (2,2) igre Nim

Igra Nim s aspekta klasifikatorskih sustava

Kao i kod igre *Kutije* moguće je reducirati problem na *single step* problem. Igra će na isti način biti podijeljena u *pod-igre* koje će predstavljati *single step* probleme. Rješenje pojedinog pod-problema predstavlja ispravan potez za trenutnu akciju. Cilj sustava je za svako stanje okoline naučiti ispravnu akciju.

S aspekta klasifikatorskih sustava dvije promatrane igre vrlo su različite, iako su logički promatrano jako slične. Temeljna razlika krije se u broju mogućih akcija koje su dostupne klasifikatorskom sustavu u svakom potezu. Primjerice, ukoliko je riječ o igri *Nim 4/7* broj mogućih akcija za najgori slučaj je 28. Hrpu je moguće izabrati na četiri načina, dok je broj objekata za uklanjanje moguće izabrati na 7 načina. Ovaj problem uzrokuje značajno povećanje vremena potrebnog sustavu da odredi ispravne akcije.

Puno ozbiljniji, a pokazuje se i nerješiv problem za sustav predstavlja nepostojanje efikasne generalizacije pravila koja je ključna za proces učenja. Problem će biti detaljno proučen na podatcima iz tablice 5.4. Tablica opisuje neka stanja i ispravne akcije igre *Nim 2/7*. Za binarni prikaz akcije dovoljna su četiri bita informacije. Prvi bit određuje hrpu dok preostala dva određuju broj objekata koje treba ukloniti. Primjerice, akcija *0101* predlaže uklanjanje pet objekta s prve hrpe. Potrebno je promotriti stanja igre u kojima je na drugoj hrpi broj objekata uvijek za jedan veći od broja objekata na prvoj hrpi.

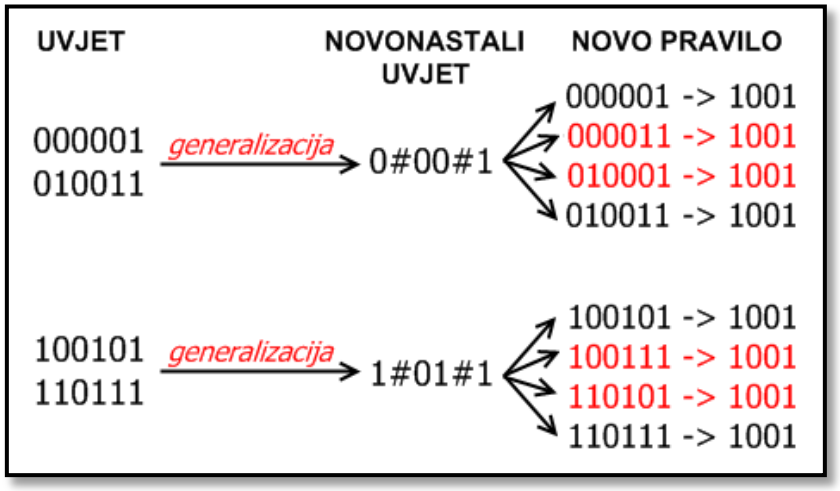
Tablica 5.4 Stanja igre Nim 2/7

Pravilo	Hrpa 1.	Hrpa 2.	Binarno	Ispravna akcija	
1.	0	1	000001	(2,1)	1001
2.	1	2	001010	(2,1)	1001
3.	2	3	010011	(2,1)	1001
4.	3	4	011100	(2,1)	1001
5.	4	5	100101	(2,1)	1001
6.	5	6	101110	(2,1)	1001
7.	6	7	110111	(2,1)	1001

Pobjednička strategija u igri s dvije hrpe je uvijek napraviti potez koji će izjednačiti veličinu hrpa (nim-suma dva jednaka broja je nula). Sve akcije navedene u tablici su ispravne – predlažu uklanjanje jednog objekta s druge hrpe.

Cilj generalizacije je uspostaviti klase ekvivalencije pravila s obzirom na akciju. Pokazat će se da je za ovaj problem to nemoguće ukoliko su pravila kodirana binarno.

Na slici 5.4 nalazi se pokušaj generalizacije uvjetnih dijelova dvaju parova pravila. Na lijevoj strani nalaze se uvjetni dijelovi pravila koji se razlikuju na točno dva mjesta – u drugom i petom bitu. Na slici su istaknuti neispravni klasifikatori dobiveni generalizacijom.



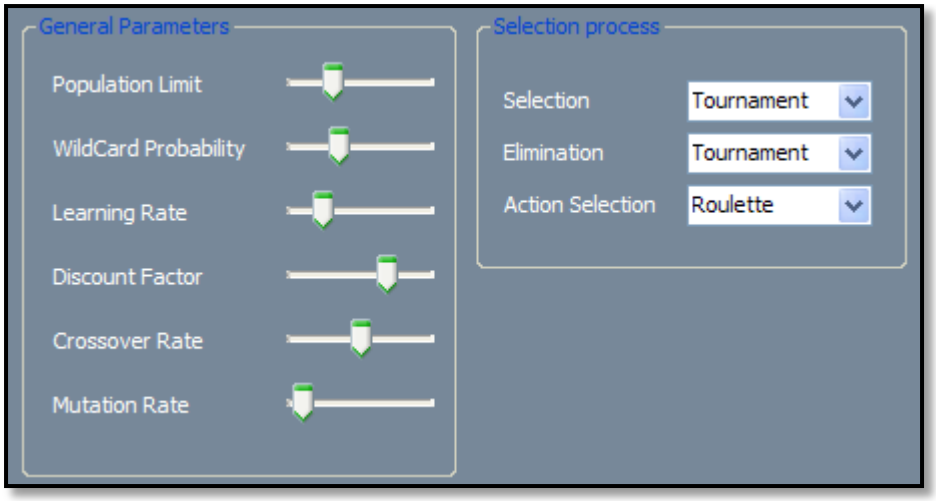
Slika 5.4 Neispravna pravila nastala generalizacijom

Ukoliko se u skup pravila klasifikatorskog sustava dodaju i dva novonastala neispravna pravila postoji velika vjerojatnost da će sustav predlagati neispravne akcije. Prema pravilu *000011*:*1001* sa slike 5.4 slijedi da, ukoliko na prvoj hrpi nema objekata, a na drugoj se nalaze tri objekta, sustav će predložiti očito pogrešnu akciju uklanjanja jednog objekta s druge hrpe. Tu će nepreciznost sustava lako iskoristiti drugi igrač te će ukloniti dva preostala objekta i pobijediti. Kako je za svako pobjedničko stanje igre potrebno jedno ispravno pravilo, učenje ispravnih poteza u igri Nim za klasifikatorski problem predstavlja značajan problem. Primjerice, za potpuno preslikavanje u igri Nim 4/7 bilo bi potrebno 3800 klasifikatora.

6 Treniranje klasifikatorskih sustava

U ovom poglavlju bit će detaljno razrađen model treniranja klasifikatorskih sustava *ZCS* i *XCS* kako bi *naučili* igrati do sada obrađene igre. U prethodnom poglavlju pokazano je kako se problem određivanja pobjedničke strategije u navedenim igrama može reducirati na rješavanje *single-step* problema.

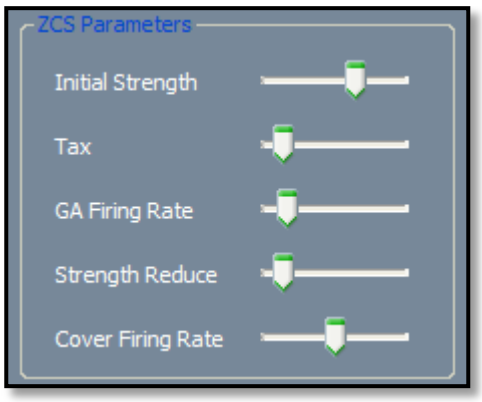
Za računalni program (sustav) kažemo da uči kroz iskustvo *E* u odnosu na neki skup zadataka *T* i s obzirom na neku mjeru uspješnosti *P*, ako se povećava uspješnost obavljanja zadataka *T*, kroz iskustvo *E*, mjerena mjerom uspješnosti *P[20]*.
Iskustvo će biti predstavljeno nizom nasumce odabranih stanja okoline za koja sustav mora pronaći ispravnu akciju. Mjera uspješnosti bit će broj uspješno klasificiranih stanja u prethodnih 1000 iteracija. Cilj sustava je postići 100%-tnu preciznost, odnosno ispravnu klasifikaciju svakog stanja okoline.
Sustav će biti treniran za igranje dvije varijante igre *Kutije*. U prvoj varijanti maksimalni broj kutija na hrpi bit će 1024. U drugoj će taj broj biti daleko veći - 32767. Budući da je u jednostavnijem slučaju broj mogućih stanja okoline 2^{10} , za uvjetni dio pravila bit će dovoljno 10 bitova. U drugoj varijanti broj bitova potreban za uvjetni dio pravila je 15. U obje varijante postoje samo dvije moguće akcije sustava pa će za akcijski dio pravila biti dovoljan je jedan bit.
Sustav će također biti treniran za igranje dvije varijante igre *Nim*. Prva varijante bit će igra Nim 2/3. Druga varijanta bit će Nim 4/7. U prvoj varijanti igre sva stanja okoline moguće je opisati sa šest bitova, po tri za svaku hrpu. Za akcijski dio pravila potrebno je četiri bita. Opis parametara koji se koriste u sustavima dan je u dodatku B. Za očekivati je da će za teže varijante obiju igara brzina konvergencije biti manja. Na slici 6.1 prikazani su parametri *ZCS* i *XCS* sustava.



Slika 6.1 Parametri zajednički za klasifikatorske sustave ZCS i XCS

6.1 Treniranje ZCS-a

Provedeno treniranje implementiranog *ZCS*-a na problemu *Kutije* pokazalo je da sustav može postići optimum. Međutim, u težoj varijanti obje igre sustav zbog slabe mogućnosti generalizacije jako sporo konvergira ka rješenju.



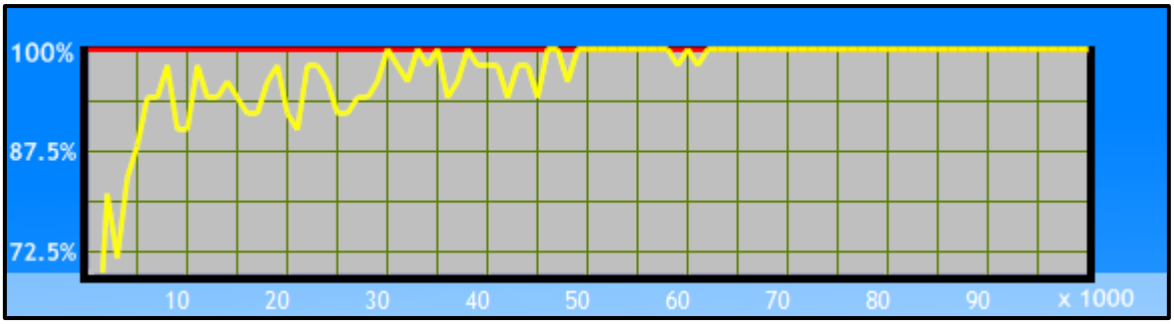
Slika 6.2 Parametri ZCS-a

Kutije

Pri treniranju sustava za učenje jednostavne varijante korišteni su sljedeći parametri:

- Veličina populacije (P) = 400
- Početna snaga pravila pri kreiranju *ZCS* sustava (S_0) = 20.0
- Parametar β = 0.75
- Parametar τ = 0.9
- Vjerojatnost odabira *don't care* simbola pri stvaranju pravila (P_{wld}) = 0.333
- Vjerojatnost pokretanja GA-a u pojedinoj iteraciji (P_{ga}) = 0.125
- Vjerojatnost križanja (λ) = 0.5
- Vjerojatnost mutacije (μ) = 0.003
- Koeficijent smanjivanja snage djece (R_{ch}) = 0.1
- Parametar koji određuje pokretanje *cover* algoritma Φ = 0.5
- Primljena nagrada za ispravnu akciju = 1000.0
- Selekcija roditelja: jednostavna
- Eliminacija pravila: turnirska
- Odabir akcije: jednostavna

Parametar γ se ne pojavljuje jer se u svakoj iteraciji prazni prethodni akcijski skup. Akcije nemaju međusobne poveznice jer se stanje okoline odnosno broj kutija na hrpi generira slučajnim postupkom. Parametri β i τ su blizu jedinice kako bi se pravilima koja krivo klasificiraju što više smanjila snaga. U jednostavnijem slučaju sustav jako brzo konvergira prema 100%-tnoj preciznosti. Rezultati treniranja prikazani su na slici 6.3. Sustav dostiže optimum u prosjeku nakon 15000 iteracija.

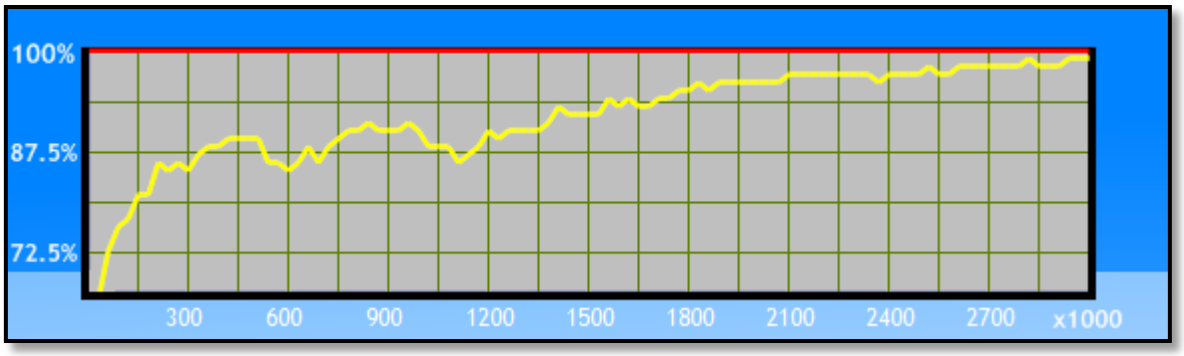


Slika 6.3 Rezultati treniranja ZCS-a za igru Kutije 1024

Za rješavanje teže varijante problema promijenjeni su sljedeći parametri:

- N = 2000
- Selekcija roditelja: Turnirska
- Eliminacija pravila: jednostavna
- Odabir akcije: jednostavna

Zbog slabih mogućnost generalizacije verzija sa 2^{16} kutija predstavlja gornju granicu za *ZCS* sustav. Dodavanjem jednog bita stanju okoline broj potrebnih iteracija poveća se približno 8 puta. Rezultati treniranja za težu varijantu igre *Kutije* nalaze se na slici 6.4.



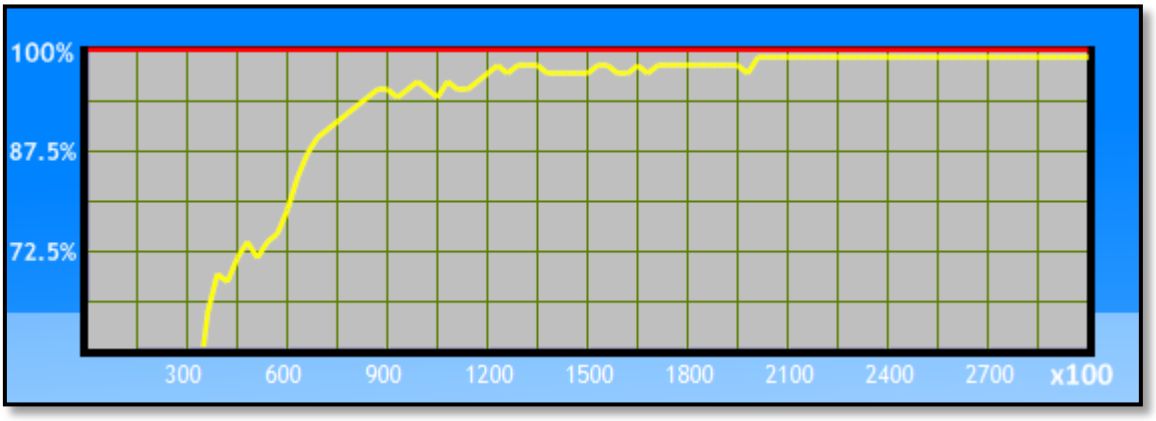
Slika 6.4 Treniranje ZCS-a za igre Kutije 32768

Nim

Kako su za ispravnu klasifikaciju stanja igre *Nim* potrebni specijalizirani klasifikatori, podešavanjem parametara sustava cilj je ukloniti efekt generalizacije pravila. Parametri koji su promijenjeni u odnosu na jednostavnu varijantu igre *Kutije* su:

- N = 1200
- Vjerojatnost odabira *don't care* simbola pri stvaranju pravila (P_{wld}) = 0.15
- Vjerojatnost pokretanja GA-a u pojedinoj iteraciji (P_{ga}) = 0.175
- Vjerojatnost križanja (λ) = 0.2
- Vjerojatnost mutacije (μ) = 0.001
- Parametar koji određuje pokretanje *cover* algoritma Φ = 0.9

Budući da je potrebno preslikati 2^6 stanja okoline na jednu od 16 akcija od kojih je samo jedna ispravna pretpostavljeni broj iteracija je nekoliko stotina tisuća. Smanjena je vjerojatnost mutacije i križanja, dok je vjerojatnost *cover* algoritma povećana. S ciljem pretraživanja problemskog prostora stvaranjem specifičnih klasifikatora *cover* algoritmom su smanjene vjerojatnosti mutacije i križanja. Vjerojatnost pokretanja *GA* u svakoj iteraciji je povećana kako bi se iz sustava brzo uklonila neispravna pravila. Rezultati treniranja za jednostavniju varijantu problema nalaze se na slici 6.5.

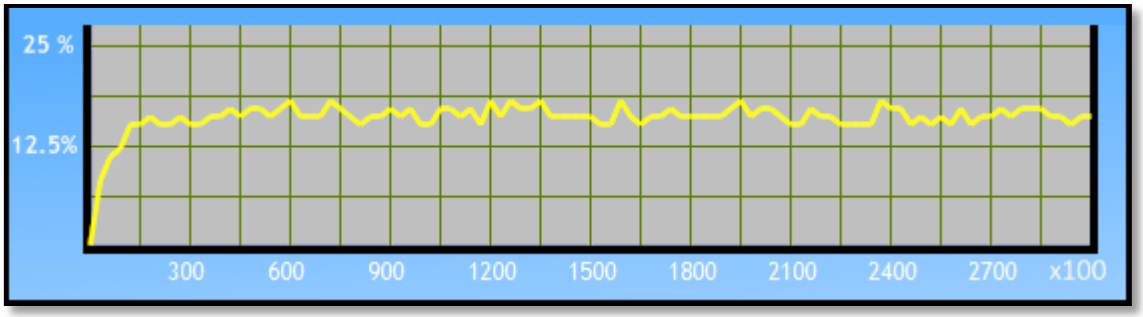


Slika 6.5 Rezultati treniranja ZCS-a za Igru Nim 2/3

Pri treniranju sustava za varijantu igre *Nim 4/7* pojavili su se predviđeni problemi. Parametri koji su promijenjeni u odnosu na igru *Nim 2/3* su:

- $N = 4000$
- Vjerojatnost odabira *don't care* simbola pri stvaranju pravila (P_{wld}) = 0.15
- Vjerojatnost pokretanja GA-a u pojedinoj iteraciji (P_{ga}) = 0.175
- Parametar koji određuje pokretanje *cover* algoritma $\Phi = 0.8$

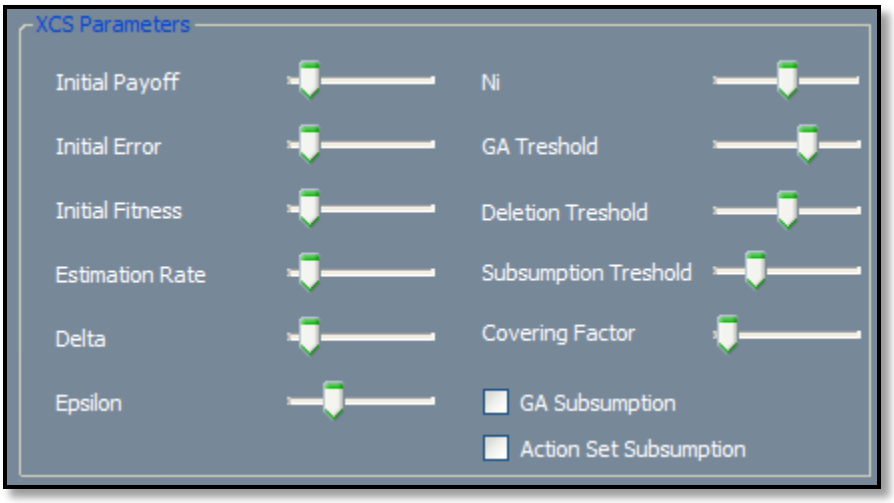
Nakon 10^6 iteracija sustav nije uspio postići preciznost veću od 20%. Povećanjem broja iteracija postiže se jedva zamjetno poboljšanje preciznosti. Nemogućnost efikasne generalizacije i velik problemski prostor za *ZCS* predstavljaju nerješiv problem. Napredovanje sustava prikazano je na slici 6.6.



Slika 6.6 Rezultati treniranja ZCS-a za Igru Nim 4/7

6.2 Treniranje XCS-a

Provedeno treniranje *XCS*-a na problemu *Kutije* pokazalo je da sustav u obje varijante igre vrlo brzo postiže optimum. U prethodnom poglavlju pokazano je kako bi igra Nim trebala sustavu zadavati poteškoće. Grafički prikaza napredovanja sustava za obje verzije problema to i potvrđuje.



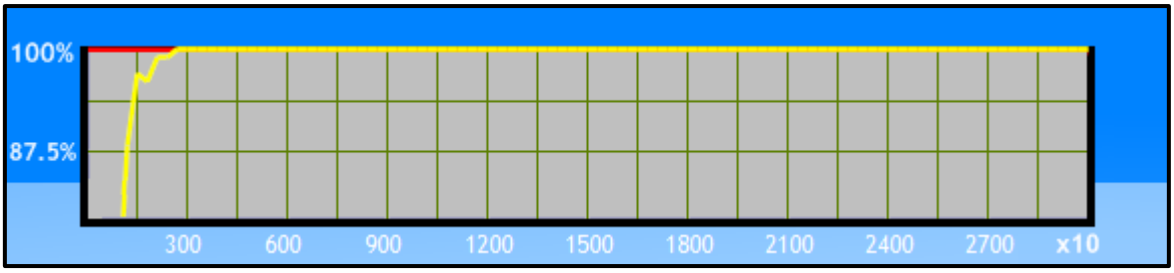
Slika 6.7 Parametri XCS-a

Kutije

Pri treniranju sustava za obje varijante igre *Kutije* korišteni su sljedeći parametri:

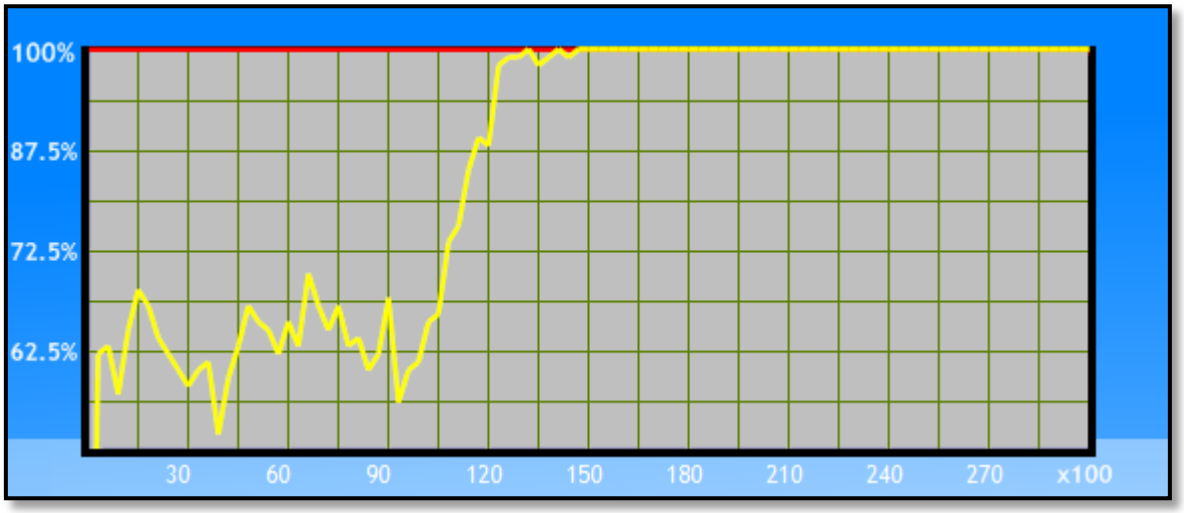
- Veličina populacije N = 400;
- Početno predviđanje nagrade pravila pri kreiranju *XCS* sustava (p_0) = 10.0;
- Vjerojatnost odabira *don't care* simbola pri stvaranju pravila (P_{wld}) = 0.333;
- Parametar β = 0.2;
- Parametar τ = 5;
- Parametar $\nu = 5$
- Vjerojatnost križanja (λ) = 0.5;
- Vjerojatnost mutacije (μ) = 0.005;
- Primljena nagrada za ispravnu akciju = 1000.0.
- Selekcija roditelja: jednostavna
- Eliminacija pravila: turnirska
- Odabir akcije: *explore / exploit strategija*

Na slici 6.8 prikazani su rezultati treniranja za jednostavnu varijantu igre.



Slika 6.8 Rezultati treniranja XCS-a za igru Kutije 1024

Na slici 6.9 prikazani su rezultati treniranja za težu varijantu igre. Sustav konvergira ka optimalnom rješenju nakon prosječno 15* 10^3 iteracija.



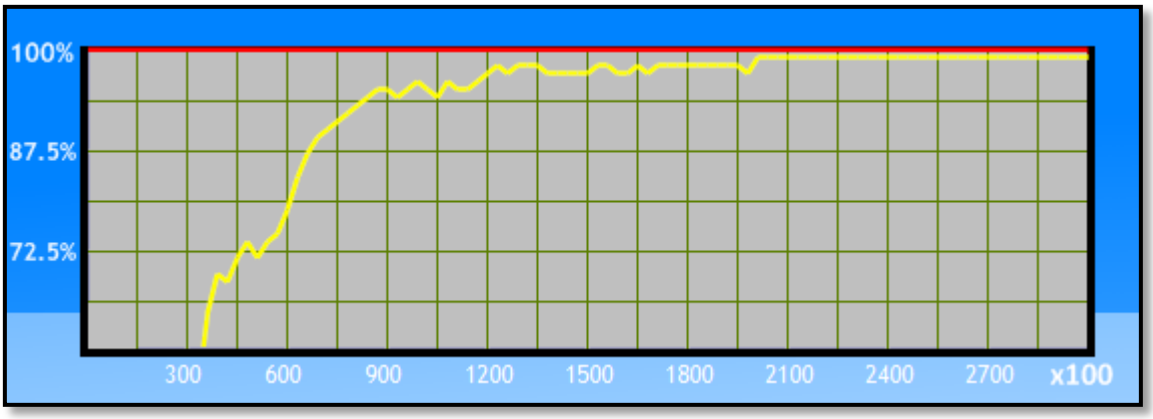
Slika 6.9 Rezultati treniranja XCS-a za igru Kutije 32768

Nim

Pri treniranju sustava za obje varijante igre *Nim* korišteni su sljedeći parametri:

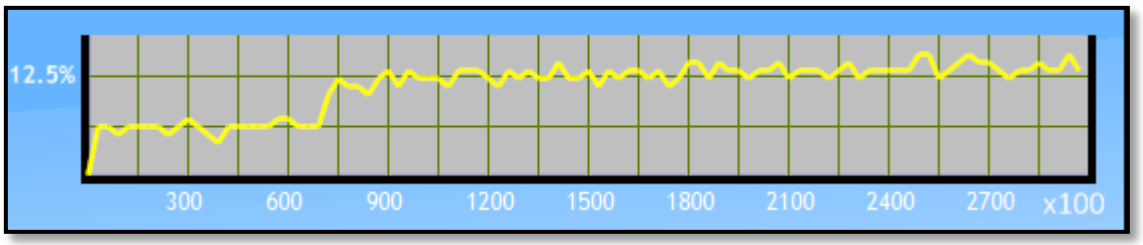
- Veličina populacije N = 4000;
- Početno predviđanje nagrade pravila pri kreiranju *XCS* sustava (p_0) = 20.0;
- Početna greška = 20.00;
- Vjerojatnost odabira *don't care* simbola pri stvaranju pravila (P_{wld}) = 0.2;
- Parametar β = 0.2;
- Parametar τ = 5;
- Parametar $\nu = 5$
- Vjerojatnost križanja (λ)= 0.1;
- Vjerojatnost mutacije (μ) = 0.001;
- Primljena nagrada za ispravnu akciju = 1000.0.
- Selekcija roditelja: jednostavna
- Eliminacija pravila: turnirska
- Odabir akcije: *explore / exploit strategija*

Na slici 6.10 prikazani su rezultati treniranja za igru *Nim 2/3.* Zanimljivo je da sustav ne dostiže optimalnu preciznost neovisno o broju iteracija. Sustav stvara jako velik broj generaliziranih pravila pa se događa da za neka stanja okoline sustav koristi generalizirani klasifikator, iako se u populaciji nalazi ispravni klasifikator. Budući da je generalizirani klasifikator za igru *Nim* najčešće i pogrešan klasifikator, sustav griješi.



Slika 6.10 Rezultati treniranja XCS-a za igru Nim 2/3

Pri učenju pravila za igru Nim 4/7 pojavljuju se slični problemi kao i kod *ZCS*-a. Neovisno o broju klasifikatora i ostalim parametrima sustava, maksimalna preciznost koju sustav može dosegnuti je 20%.



Slika 6.11 Rezultati treniranja XCS-a za igru Nim 4/7

7 Zaključak

Rezultati ispitivanja pokazuju da je moguće naučiti klasifikatorski sustav igrati logičke igre. Ispravni potezi za pojedinu poziciju dobiveni matematičkim zaključivanjem poklapaju se s potezima koje sustav predlaže nakon treniranja. Brzina učenja ponajprije ovisi o broju pozicija i poteza u odgovarajućoj igri.

Za učenje su odabrane dvije igre koje otkrivaju prednosti i mane učenja pobjedničkih strategija pomoću klasifikatorskih sustava. Učenje klasifikatorskog sustava reducirano je na rješavanje *single step* problema.

S aspekta klasifikatorskih sustava logičke igre mogu se podijeliti u dvije skupine. Prvoj skupini pripadaju igre za koje postoji mogućnost efikasne generalizacije stanja okoline. Rezultati ispitivanja u takvim igrama pokazuju kako su klasifikatorski sustavi *ZCS* i *XCS* sposobni jako brzo naučiti ispravne poteze te konkurirati klasičnim tehnikama pretraživanja problemskog prostora. U igrama za koje ne postoji efikasna generalizacija stanja okoline preciznost sustava sporo konvergira ili uopće ne konvergira ka optimalnoj.

Literatura

[1] Eiben A. E., Smith J. E., *Introduction to Evolution Computing*, 1. izdanje, Berlin: Springer-Verlang, 2003.

[2] Sigaud O., Wilson S., *Learning Classifier Systems: A Survey*, dostupno na: http://animatlab.lip6.fr/papers/sigaud_wilson_*LCS*_survey.pdf, listopad 2007.

[3] Bull L., *Learning Classifier Systems: A Brief Introduction*, dostupno na: http://www.cems.uwe.ac.uk/*LCS*g/introchap.pdf, listopad 2007.

[4] Bernando-Mansilla E., Garrell-Guiu J. M., *Accuracy-Based Learning Classifier Systems: Models, Analysis and Applications to Classification Tasks*, dostupno na: http://sci2s.ugr.es/keel/pdf/keel/articulo/bernado03accuracy.pdf, listopad 2007.

[5] -, Wikipedia: *Machine learning*, *Reinforcement learning*, *Supervised learning*, dostupno na: http://en.wikipedia.org/wiki/Machine_learning, studeni 2007.

[6] Golub M., *Genetski algoritam*, rujan 2004., dostupno na: http://www.zemris.fer.hr/~golub/ga/ga_skripta1.pdf, listopad 2007.

[7] Kaelbling L. P., Littman M. L., Moore A. W., *Reinforcement Learning: A Survey*, dostupno na: http://www.cs.cmu.edu/afs/cs/project/jair/pub/volume4/kaelbling96a.pdf, studeni 2007.

[8] Vasilyev A., *Synergetic Approach in Adaptive Systems*, magistarski rad, Transport and Telecommunication Institute, Riga, 2002., dostupno na: http://www.cs.rtu.lv/dssg/download/thesis/bimsc/2002/Vasilyev-MSc-2002-en.pdf, studeni 2007.

[9] Weise T., Achler S., Gob M., Voigtmann C., Zapf M., *Evolving Classifiers – Evolutionary Algorithms in Data Mining*, dostupno na: https://kobra.bibliothek.uni-kassel.de/bitstream/urn:nbn:de:hebis:34-2007092819260/1/Technicalreport2007_4.pdf, studeni 2007.

[10] Beasley D., *Q1.4: What's a Classifier system (CFS)?*, svibanj 2007., dostupno na: http://www.faqs.org/faqs/ai-faq/genetic/part2/section-5.html, listopad 2007.

[11] Butz M. V., *Learning Classifier Systems*, dostupno na: http://delivery.acm.org/10.1145/1280000/1274104/p3035-butz.pdf?key1=1274104&key2=4672214911&coll=portal&dl=ACM,GUIDE&CFID=15151515&CFTOKEN=6184618, listopad 2007.

[12] Cordon O., del Jesus M. J., Herrera F., *Evolutionary Approaches To The Learning Of Fuzzy Rule–based Classification Systems*, dostupno na: http://sci2s.ugr.es/publications/ficheros/cordon-del_jesus-herrera-Evolutionary%20approaches-1999-107-160.pdf, studeni 2007.

[13] Bacardit J., Krasnogor N., *Introduction to Learning Classifier Systems*, dostupno na: www.cs.nott.ac.uk/~nxk/TEACHING/G53BIO/*LCS*-1.ppt, studeni 2007.

[14] Garrell-Guiu J. M., *Pittsburgh Genetic-Based Machine Learning in the Data Mining era: Representations, generalization, and run-time*, doktorat, Universitat Ramon Llull: Enginyera i Arquitectura La Salle, Barcelona, listopad 2004., dostupno na: http://sci2s.ugr.es/keel/pdf/keel/tesis/bacardit.pdf, studeni 2007.
[15] Kusiak A., *Learning Classifier Systems: An Introduction*, dostupno na: http://www.icaen.uiowa.edu/~ie238/Lecture/*LCS*_2.pdf, studeni 2007.
[16] Toft I., *A Java Implementation of a Zeroth Level Classifier System*, rujan 2005., dostupno na: http://www2.cmp.uea.ac.uk/~it/zcs/zcs.html, siječanj 2008.
[17] Wilson S. W., *ZCS: A Zeroth Level Classifier System*, Evolutionary Computation, kolovoz 1993., Vol. 2, No. 1, str. 1-18
[18] Butz M., Wilson S., *An algorithmic description of XCS*, dostupno na: http://citeseer.ist.psu.edu/butz01algorithmic.html, siječanj 2008.
[19] Kovacs T., Kerber M., *A Study of Structural and Parametric learning in XCS*, dostupno na: http://www.mitpressjournals.org/doi/pdf/10.1162/evco.2006.14.1.1, siječanj 2008.
[20] Dalbelo B., *Bilješke sa predavanja UI*, dostupno na: http://fer.hr/_download/repository/Strojno_ucenje-UVOD-2008.pdf, siječanj 2008.
[21] Botničan M., *Kombinatorne igre,dostupno na:* http://e.math.hr/igre/index.html, svibanj 2008.
[22] Krčadinac V., *Teorija igara - matematičko modeliranje konfliktnih situacija*, dostupno na: http://e.math.hr/teorijaigara/ , svibanj 2008.
[23] Lučić M., Radanović G., *Evolucijski algoritmi*: *Klasifikatorski sustavi*, projekt, Fakultet elektrotehnike i računarstva, Zagreb, 2008.

Dodatak A: Parametri LCS sustava

Tablica A.1 Parametri koji se pojavljuju u LCS sustavima

Oznaka	Objašnjenje	
α	Parametar korišten pri evaluaciji pravila	/
β	Brzina učenja	*Learning rate*
γ	Koeficijent smanjenja nagrade	*Discount factor*
δ	Prag dobrote XCS pravila pri eliminaciji	/
ε	Procjena pogreške predviđanja XCS pravila	*Estimated error*
ε_0	Gornja granica vrijednosti pogreške	/
Θ_{DEL}	Minimalno iskustvo XCS pravila potrebno da se pri eliminaciji tog pravila uzme u obzir njegova dobrota	*Deletion threshold*
Θ_{GA}	Prag za pokretanje GA	*GA threshold*
Θ_{SUB}	Prag koji određuje iskustvo XCS pravila potrebno da bi pravilo obuhvatilo manje općenita pravila	*Subsumption threshold*
κ	Preciznost XCS pravila	*Wild-card probability*
κ_r	Relativna preciznost XCS pravila	*Learning rate*
λ	Vjerojatnost križanja	*Crossover rate*
μ	Vjerojatnost mutacije	*Mutation rate*
ν	Parametar korišten pri evaluaciji pravila	/
ρ	Nagrada od okoline	*Environment reward*
σ	Stanje okoline	*Environment state*
τ	Dio snage koje se uzima pravilima iz podudarnog skupa u *ZCS*-u	*Tax rate*