SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 322

ZCS i XCS klasifikatorski sustavi

Goran Radanović

Zagreb, lipanj 2008.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 322

ZCS i XCS klasifikatorski sustavi

Goran Radanović

Zagreb, lipanj 2008.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA ZAVRŠNI RAD MODULA

Zagreb, 27. veljače 2008.

ZAVRŠNI ZADATAK br. 322

Pristupnik: **Goran Radanović**
Studij: Računarstvo
Modul: Računarska znanost

Zadatak: **ZCS i XCS klasifikatorski sustavi**

Opis zadatka:

Proučiti i opisati klasifikatorske sustave s mogućnošću učenja zasnovane na snazi, očekivanju i preciznosti. Posebnu pažnju posvetiti ZCS klasifikatorskom sustavu zasnovanom na snazi te XCS klasifikatorskom sustavu zasnovanom na preciznosti. Primijeniti ZCS i XCS klasifikatorske sustave za rješavanje problema labirinta. Načiniti programski sustav s odgovarajućim grafičkim sučeljem.

Zadatak uručen pristupniku: 27. veljače 2008.
Rok za predaju rada: 13. lipnja 2008.

Mentor:

Doc. dr. sc. Marin Golub

Djelovođa:

Doc. dr. sc. Domagoj Jakobović

Predsjednik odbora za završni rad modula:

Prof. dr. sc. Bojana Dalbelo-Bašić

Sažetak

Klasifikatorski sustavi s mogućnošću učenja su sustavi zasnovani na skupu pravila koji kombiniraju mehanizme strojnog učenja s evolucijskim procesom. Rad daje uvid u stilove klasifikatorskih sustava s mogućnošću učenja i temeljit opis ZCS i XCS klasifikatorskih sustava primijenjenih u rješavanju problema labirinta.

Ključne riječi: *Klasifikatorski sustavi s mogućnošću učenja, Sustavi zasnovani na skupu pravila,Učenje potkrjepljenjem (podražajem), Učenje pod nadzorom, Genetski algoritam, Problemi labirinta, Generalizacija.*

Abstract

Learning Classifier Systems are rule-based systems that can be considered as a combination of machine learning techniques and evolutionary process. This paper provides insight into different approaches to learning classifier systems and thorough description of ZCS and XCS classifier systems which were applied to solve maze problems.

Key words: *Learning Classifier Systems, Rule-based Systems, Reinforcement Learning, Supervised Learning, Genetic Algorithm, Maze Problems, Generalization.*

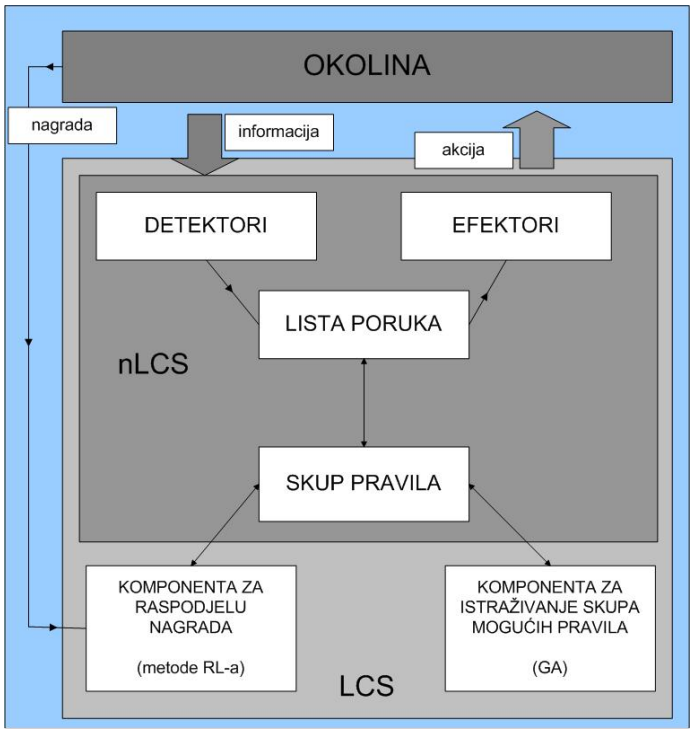
Sadržaj

1. Uvod

Početkom 1970.-ih John Holland je došao na ideju da optimizacijske probleme riješi programskim modelom koji se temelji na Darwinovoj teoriji evolucije. Tako su nastali, danas dobro poznati, genetski algoritmi. Nedugo kasnije Holland je opisao kognitivne sustave zasnovane na pravilima koji u sebi sadrže genetski algoritam kao algoritam pretrage prostora pravila. Danas su takvi kognitivni sustavi poznati pod nazivom **klasifikatorski sustavi koji imaju sposobnost učenja (*Learning Classifier Systems, LCS*)**. Iako nastali u vrijeme kada i genetski algoritmi, LCS sustavi su sve do danas ostali u sjeni genetskih algoritama, ponajprije zbog toga što su genetski algoritmi po strukturi jednostavniji, a u rješavanju optimizacijskih problema i znatno bolji.
LCS se može shvatiti kao sustav strojnog učenja koji koristi genetske algoritme (ili neku drugu heurističku tehniku pretraživanja prostora pravila) u postupku svoga učenja. Pravila evoluiraju iz ciklusa u ciklus zahvaljujući genetskom algoritmu (ili nekoj drugoj heurističkoj tehnici pretraživanja prostora pravila) i sustavu nagrađivanja. LCS sustavi posjeduju sposobnost generalizacije pravila što znači da jedno generalizirano pravilo može predstavljati veći broj specijaliziranih pravila. Kako bi sustav učio, on treba upotrebljavati jednu od tehnika strojnog učenja (učenje pod nadzorom, učenje bez nadzora, učenje potkrjepljenjem (podražajem)). Tako michiganski stil LCS-a upotrebljava tehniku učenja potkrjepljenjem (podražajem), dok pittsburški stil i stil iterativnog učenja pravila upotrebljavaju tehniku učenja pod nadzorom (učenja primjerima).
LCS pripada klasi evolucijskih algoritama koji spadaju u područje evolucijskog računanja, a evolucijsko računanje spada u područje umjetne inteligencije. LCS se upotrebljava za klasifikaciju, u problemima učenja potkrjepljenjem, u aproksimacijskim problemima i za probleme predviđanja. Od područja primjene valja izdvojiti dubinsku analizu podataka, upravljanje procesima te modeliranje i optimizaciju.
Iako bi se u klasifikatorske sustave moglo ubrojiti mnoge sustave zasnovane na pravilima, u radu je interes sveden prvenstveno na sustave zasnovane na pravilima koji imaju sposobnost evolucije skupa pravila. Podjelom klasifikatorskih sustava se bavi drugo poglavlje. Treće poglavlje opisuje kanonski model michiganskog stila LCS-a, dok četvrto poglavlje daje podrobniji uvid u porodice michiganskog stila: LCS zasnovan na snazi, LCS zasnovan na preciznosti i LCS zasnovan na očekivanju. U petom poglavlju se daju objašnjenja za *multi step* (višekoračne) i *single step* (jednokoračne) probleme kao i primjeri navedenih problema. U šestom poglavlju se detaljnije opisuju implementirani ZCS i XCS klasifikatorski sustavi, a u sedmom poglavlju su dani rezultati mjerenja provedeni nad implementiranim sustavima. Osmo poglavlje sadrži opis ostvarene aplikacije, a u devetom poglavlju je ukratko opisana primjena LCS-a.

2. Pregled klasifikatorskih sustava

Klasifikatorski sustavi (*Classifier systems*, CS, CFS) su sustavi temeljeni na skupu pravila (klasifikatora) koja određuju reakciju sustava na uvjete dane u njegovoj okolini. Klasifikatorski sustavi se dijele na **klasifikatorske sustave koji nemaju sposobnost učenja (*non-learning classifier systems*, nLCS)** i **klasifikatorske sustave koji imaju sposobnost učenja (*learning classifier systems*, LCS)** [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 prenosi stanje okoline 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 puni 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, dobivamo 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 [25].

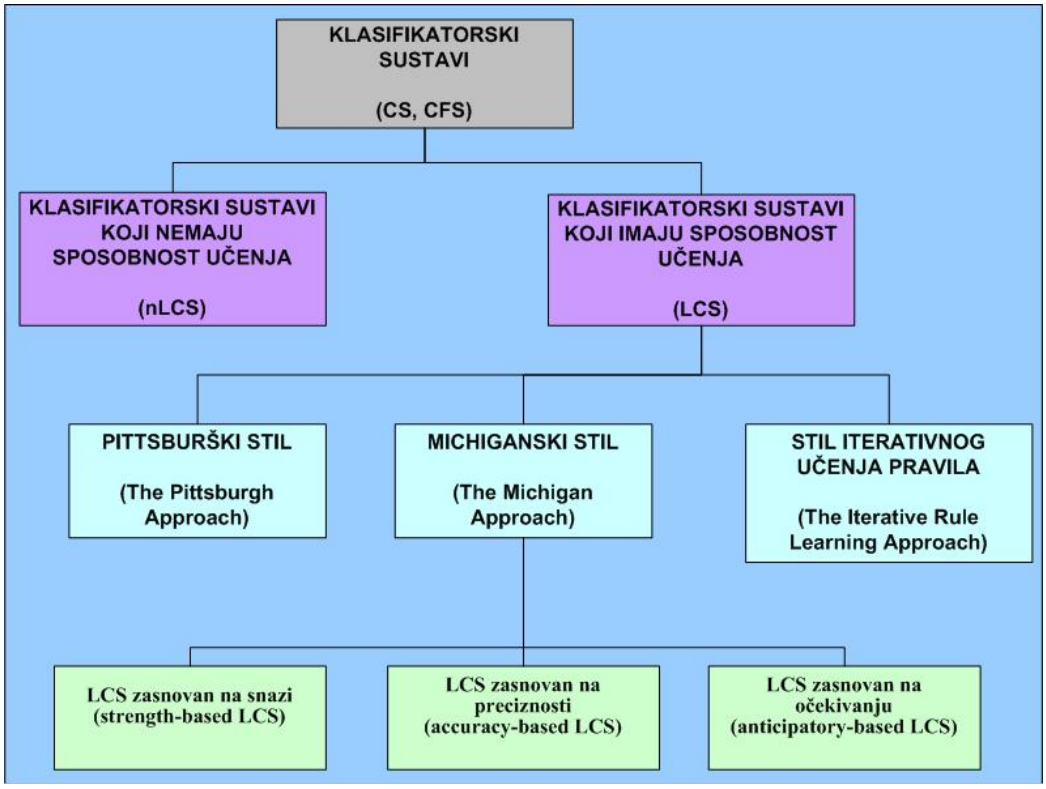


Slika 2.1 Odnos nLCS-a i michiganskog stila LCS-a

2.1 Podjela LCS-a

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

***Pittsburgh Approach*)** [2]. Uz michiganski i pittsburški stil postoji i **stil iterativnog učenja pravila (*The Iterative Rule Learning Approach*)**, prvi puta primijenjen na SIA sustavima (Venturini, 1993) [14]. Podjela CS-a i LCS-a prikazana je na slici (2.2) [25].



Slika 2.2 Podjela LCS-a

2.2 Michiganski stil LCS-a

Michiganski stil LCS-a karakterizira kombinacija **učenja potkrjepljenjem (*Reinforcement Learning*, RL) i genetskog algoritma (*Genetic Algorithm*, GA)**. RL je tehnika učenja pokušajima i pogreškama putem primljenih nagrada, a u LCS-u predstavlja komponentu za raspodjelu nagrada primljenih od okoline [3]. GA je heuristička metoda optimizacije temeljena na evolucijskom procesu, a u LCS-u predstavlja komponentu za istraživanje prostora mogućih pravila. Svakako treba spomenuti da se umjesto GA-a može koristiti i neka druga heuristička metoda istraživanja prostora pravila kao i da se GA može nadopuniti brojnim heuristikama. Ono što bitno razlikuje michiganski stil LCS od nekih drugih (klasičnih) tehnika RL-a je sposobnost generalizacija pravila: postoje pravila čiji uvjetni dio odgovara različitim stanjima okoline čime se smanjuje broj pravila kojima sustav treba raspolagati. Ipak, treba naglasiti da prevelika generalizacija nije dobra; može dovesti do pojave 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 se obavlja lokalno, tj. 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

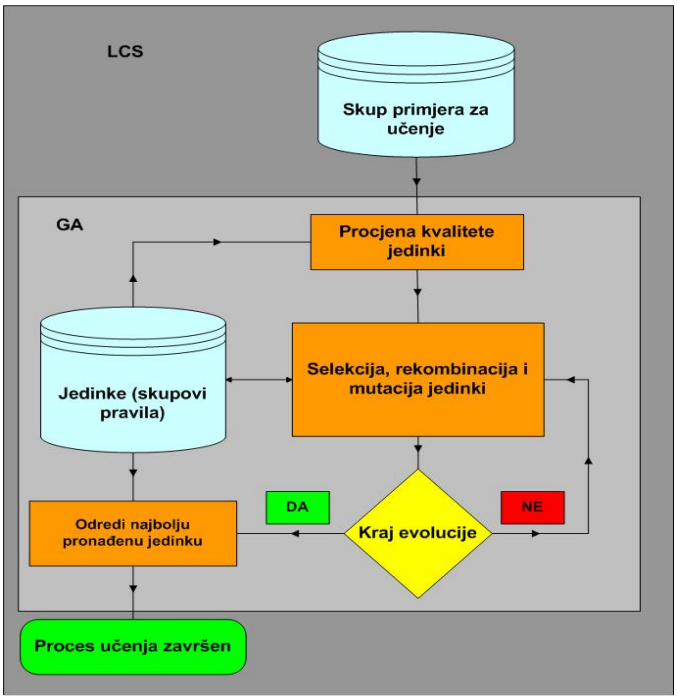
optimalno reagirati na stanja okoline, potrebno je uravnotežiti odnos potreba pojedine jedinke (pravila) i potreba cijelog sustava [12]. Potreba jedinke (pravila) se očituje u tome da opstane, tj. ne bude eliminirana GA-om. To znači da svaka jedinka treba težiti dobivanju što većeg dijela nagrade. S druge strane, potreba sustava je da jedinke (pravila) 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. Shematski prikaz michiganskog stila LCS-a (i njegovog odnosa sa nLCS-om) dan je na slici (2.1).
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* (*real-time systems*) gdje nagle promjene ponašanja nisu dozvoljene [12].
Tri su porodice michiganskog stila LCS-a:

- LCS zasnovan na snazi (*strength-based LCS*)
- LCS zasnovan na preciznosti (*accuracy-based LCS*)
- LCS zasnovan na očekivanju (*anticipatory-based LCS*).

Iako su aktivna istraživanja i na drugim stilovima LCS-a, michiganski stil se smatra standardnom formom LCS-a [2][25].

2.3 Pittsburški stil LCS-a

Pittsburški stil LCS-a kombinira metode **učenja pod nadzorom (*Supervised Learning*, SL)** sa GA-om. SL je tehnika strojnog učenja koja stvara odgovarajuću funkciju na temelju *skupa za učenje* (*training set*) sastavljenog od primjera oblika *ulaz-izlaz* (*input-output*) [5]. Cilj sustav koji uči SL tehnikom je, nakon što prođe kroz primjere zadane u skupu za učenje, predvidjeti željeni *izlaz* za odgovarajući *ulaz.*

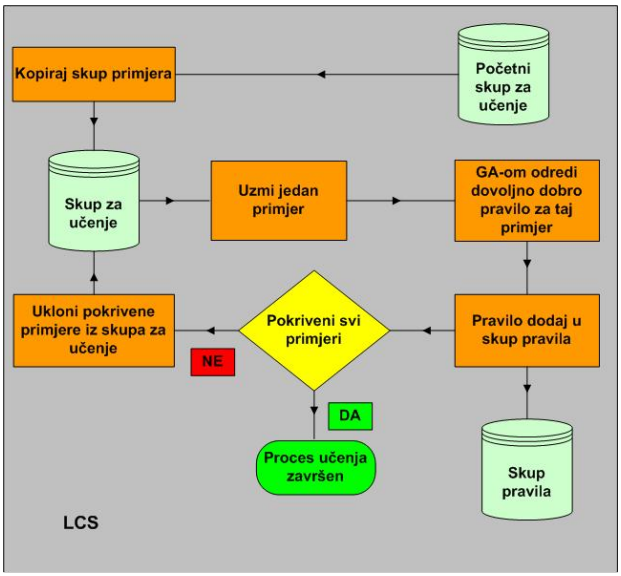


Slika 2.3 Pittsburški stil LCS-a

Za razliku od michiganskog stila, u pittsburškom 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 jedna jedinka može sadržavati različiti broj pravila pa je jedinka promjenjive duljine. Dozvoljena su i generalizirana pravila.
Nad skupom jedinki se pokreće GA čije su 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, tj. na temelju točnosti kojom jedinka obavlja klasifikaciju. Prilikom određivanja kvalitete mogu se uključiti i neke posebne karakteristike jedinke poput njene duljine. Trajanje procesa učenja zadano je evolucijskim vremenom GA-a. Model pittsburškog stila LCS-a je prikazan na slici (2.3).
Pittsburški 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. Zbog predstavljanja cijelog skupa pravila jednom jedinkom, veliki problem predstavljaju zahtijevane količine memorije i procesorskog vremena [12].
Problem učenja u pittsburškom stilu je doslovno sveden na optimizaciju GA-om, zbog čega je i upitno je li ovaj stil bliži samom GA-u ili LCS-u.
Tipični predstavnici pittsburškog stila LCS-a su algoritmi GABIL i GIL [14][25].

2.4 Stil iterativnog učenja pravila

Stil iterativnog učenja pravila se temelji na pristupu podjeli pa vladaj. Kao u michiganskom



Slika 2.4 Stil iterativnog učenja pravila

stilu, jedna jedinka predstavlja jedno pravilo, dok se sličnosti sa pittsburškim 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 pokrije zadane primjere. U svakom koraku se uzima nepokriveni primjer i uz pomoć GA-a se određuje dovoljno dobro pravilo za taj primjer. Ukoliko pravilo ne postoji u skupu pravila, ono se nadodaje, a iz skupa za učenje se brišu primjeri koji su pokriveni postojećim skupom pravila. Opisani ciklus se ponavlja 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 je prikazan na slici (2.4). Predstavnik ovoga stila je sustav HIDER [13][14][25].

3. Podrobniji pogled u model 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 [25].

3.1 Kratak pogled na genetske algoritme

Genetski algoritam (*Genetic Algorithm*, GA) je heuristička metoda optimizacije zasnovana na principima Darwinove teorije evolucije. Algoritam je baziran 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. Ne ulazeći u detalje GA-a, treba spomenuti da se u michiganskom stilu LCS-a upotrebljava eliminacijski GA (*steady-state GA*) opisan na slici (3.1).

```
Genetski algoritam{
     Ponavljaj dok traje evolucija{
             izaberi roditelje;
             križaj roditelje;
             mutiraj dobivenu djecu;
             eliminiraj onoliko roditelja koliko je stvorene djece;
     }
}
```

Slika 3.1 Pseudokod eliminacijskog GA-a

Iako je LCS opisan nedugo nakon nastanka GA-a, ostao je zasjenjen GA-om čija popularnost proizlazi iz njegove jednostavne strukture i dobrih optimizacijskih svojstava [2]. U LCS-u GA ima funkciju istraživanja prostora mogućih pravila. Pri tome se uz GA mogu primijeniti i neke druge heurističke metode [3][25].

3.2 Kratak pogled na učenje potkrjepljenjem

Učenje potkrjepljenjem (podražajem) (*Reinforcement Learning*, RL) temelji se na operantnom uvjetovanju (teoriji učenja u psihologiji), a odnosi se na područje *strojnog učenja* (*machine learning*) 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. Formalnije se RL definira:

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

U svakom trenutku (koraku) t sustav na temelju stanja okoline s_t ($s_t \in S$) odabire akciju a iz skupa mogućih akcija A. Okolina mijenja stanje s_t u s_{t+1} te daje sustavu nagradu r_t. Cilj sustava je odrediti funkciju pravila $\pi: S \rightarrow A$ koja preslikava skup stanja okoline u skup akcija sustava tako da se maksimizira ukupnost nagrada okoline.
U LCS-u RL ima funkciju raspoređivanja korisnosti pojedinim pravilima [3][25].

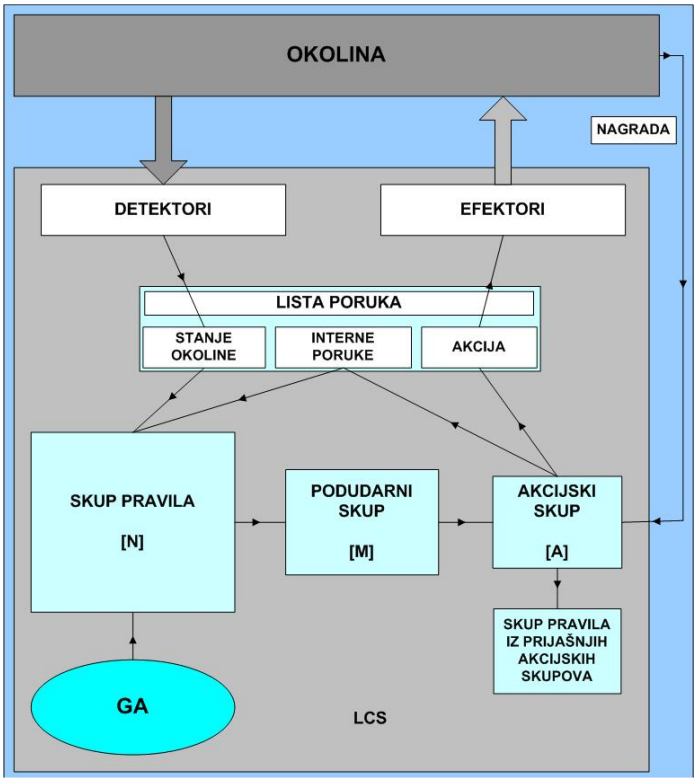
3.3 Kanonski model LCS-a

LCS spada u sustave temeljene na skupu pravila. Pravila (klasifikatori) su oblika *uvjet:akcija* značenja *ako uvjet onda akcija*. Drugim riječima, izborom pravila koje zadovoljava uvjet okoline određena je akcija prema okolini.
Uvjet je u najjednostavnijem slučaju niz znakova iz ternarnog skupa {0, 1, #} pri čemu znak # predstavlja i 0 i 1 (tzv. *don't care* znak, *wild card* znak). Primjerice, uvjet 1# predstavlja i uvjet 10 i uvjet 11. Time je omogućena generalizacija (općenitost) pravila. Pojedini simboli uvjeta se nazivaju atributi uvjeta. Akcija je u najjednostavnijem slučaju niz binarnih znakova (znakova 0 i 1) i predstavlja kod onoga što sustav izvršava.
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*. Pravilu (klasifikatoru) se može dodati i parametar brojnosti koji označava koliko se puta klasifikator pojavljuje u skupu pravila. Kako se u skupu pravila često nalazi više istih pravila, uvođenjem parametra brojnosti se štedi na memorijskom prostoru i vremenu potrebnom za jedan prolazak kroz skup pravila. Pravila koja ne sadržavaju parametar brojnosti zovu se mikro-pravila (mikro-klasifikatori), dok se pravila koja sadržavaju parametar brojnosti zovu makro-pravila (makro-klasifikatori). Veličina skupa pravila koja koriste makro-pravila je određena zbrojem brojnosti svih makro-pravila. U daljnjem tekstu se neće, ukoliko je kontekst jasan, naglašavati je li pravilo makro-pravilo ili mikro-pravilo.
Kanonski model LCS-a je shematski prikazan na slici (3.2). Proces koji se obavlja je iterativan i opisuje se sljedećim koracima:

1. Stanje okoline se pomoću detektora prenosi u listu poruka i to u dio koji je 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 namijenjenog za pohranjivanje stanja okoline i dijela za interne poruke. Uvjeti koji odgovaraju trenutnom stanju liste poruka stavljaju se u *podudarni skup* (*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 se smještaju u *akcijski skup* (*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 prethodnih koraka pripadala skupu [A] (skup $[A]_{-n}$) 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 (*steady-state GA*) [1], [3].



Slika 3.2 Kanonski model LCS-a

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

- Pronaći skup pravila koji za što veći broj mogućih uvjeta okoline sadrži pravilo koje donosi dovoljno veliku nagradu okoline. Posebice se to 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 korespondentne dijelove uvjeta, tj. 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][25].

4. 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*).

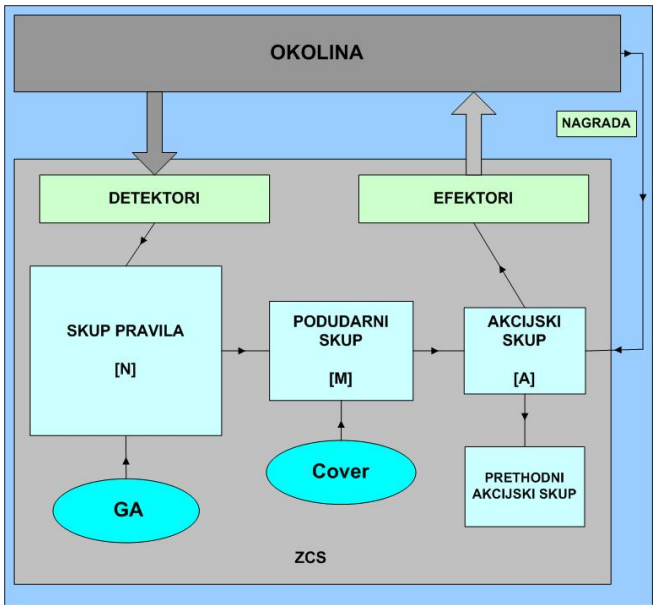
LCS zasnovan na snazi (*strength-based LCS*) je najjednostavnija porodica michiganskog stila. Korisnost pojedinog pravila je određena skalarnom veličinom zvanom snaga. Snaga predstavlja procjenu nagrade koju će sustav primiti od okoline ako se ponaša u skladu s promatranim pravilom. Pomoću snage vršimo i procjenu pojedinog pravila budući da je to jedina veličina po kojoj možemo odrediti kvalitetu pravila. No 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 određenim okolnostima biti najbolji izbor [1]. Naposljetku, najveći problem predstavljaju previše generalizirana pravila koja često vode suboptimalnom rješenju [2]. Naime, 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 nepreciznije.
Ipak, za mnoge aplikacije ova porodica LCS-a je sasvim prihvatljiva. Najpoznatiji predstavnik ove porodice je ZCS [25].

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 preciznosti 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 [25].

LCS zasnovan na očekivanju (*anticipatory-based LCS*) prikazuje pravila u obliku *uvjet:akcija:očekivanje→korisnost* pri čemu *očekivanje (effect)* predstavlja očekivano stanje okoline nakon primijenjene *akcije* na okolinu stanja *uvjeta*. Ovakva pravila (klasifikatori) grade *model prijelaza* (*model of transitions*) [2]. *Očekivanje* može, ovisno o verzijama LCS-a, sadržavati i posebne znakove = i ?. Znak = (*don't change symbol*) označava da se atribut stanja okoline ne mijenja, dok znak ? (*don't know symbol*) označava da se atribut stanja ne može predvidjeti [2]. Primjerice, ako na stanje okoline *101* djelujemo akcijom *0*, tada pravilo *#01:0:=1=→korisnost* predviđa stanje okoline *111*, a pravilo *#01:0→?1?→korisnost* predviđa samo pojavu jedinice na srednjem mjestu binarno kodiranog stanja. Znakovi = i ? imaju sličnu ulogu u dijelu pravila za očekivanje kakvu znak # (*don't care*) ima u uvjetnom dijelu pravila. Kvaliteta jedinke (pravila) je određena preciznošću predviđanja sljedećeg stanja okoline [15]. Od poznatijih predstavnika ove porodice LCS-a valja izdvojiti ACS, ACS2, YACS i MACS.

4.1 Klasifikatorski sustav ZCS

Najpoznatiji predstavnik porodice LCS-a zasnovanih na snazi je ZCS (*Zeroth-level Classifier System*). Uveo ga je Wilson kako bi povećao razumljivost i poboljšao karakteristike LCS-a [3]. ZCS se može promatrati kao pojednostavljenje kanonskog modela LCS-a, pa je pri definiranju ZCS-a dovoljno opisati razlike u odnosu na kanonski model LCS-a i naglasiti neke specifičnosti vezane uz sam ZCS.



Slika 4.1 ZCS sustav

Za razliku od kanonskog modela LCS-a ZCS nema internu listu poruka: detektor je izravno povezan sa skupom pravila ([N]), a akcijski skup ([A]) je izravno povezan s efektorom. Karakteristike između povezanih akcija se prenose implicitno (a ne eksplicitno uz pomoć interne liste poruka) i to kroz evaluaciju pravila (obnavljanje parametara pravila). Skup pravila (populacija pravila) je konstantne veličine i sastoji se od mikro-pravila, a skup koji sadržava pravila akcijskih skupova iz nekoliko prethodnih koraka ($[A]_{-n}$) reduciran je na skup pravila iz prethodnog akcijskog skupa ($[A]_{-1}$). Korisnost pravila određena je skalarnom veličinom koja se naziva snaga (*strength*, *s*). Odabir akcije se vrši jednostavnom selekcijom (*roullete-wheel selection*): akciju sustava određuje pravilo odabrano jednostavnom selekcijom. Drugim riječima, svako pravilo u podudarnom skupu može odrediti akciju sustava i to s vjerojatnošću proporcionalnom njegovoj kvaliteti (snazi). Umjesto jednostavne selekcije, odabir pravila koje određuje akciju može se vršiti i *ε-Greedy* selekcijom: vjerojatnost slučajnog odabir pravila je *ε*, dok se najbolje pravilo odabire s vjerojatnošću *1 - ε*. Evaluacija pravila (obnavljanje parametara pravila) se obavlja na sljedeći način:

1. Pravilima koji su u skupu [M], ali nisu u skupu [A] reducira se snaga za koeficijent τ ($0 \leq \tau < 1$, *tax*), tj. množi se s koeficijentom $(1 - \tau)$. Postupak je izražen u formuli (4.1).

$$\forall p \in [M] \backslash [A] \quad p.Snaga(t+1) = (1-\tau) \cdot p.Snaga(t) \tag{4.1}$$

2. Pravilima iz skupa [A] se reducira snaga za koeficijent β ($0 \leq \beta < 1$, *learning rate*), tj. množi se s koeficijentom $(1 - \beta)$. Dijelovi snage koji su oduzeti pravilima iz skupa [A] se

množe sa koeficijentom γ ($0 \leq \gamma < 1$, *discount factor*) i jednoliko raspoređuju među pravilima iz skupa $[A]_{-1}$. Postupak obnavljanja snaga pravilima prethodnog akcijskog skupa je izražen formulom (4.2).

$$\forall p \in [A]_{-1} \quad p.Snaga(t+1) = p.Snaga(t) + \gamma \cdot \frac{\sum_{p' \in [A]} \beta \cdot p'.Snaga(t)}{|[A]_{-1}|} \tag{4.2}$$

3. Primljena nagrada iz okoline (ρ) se množi sa koeficijentom β ($0 \leq \beta < 1$, *learning rate*), i jednoliko raspoređuje među pravilima iz skupa [A] [3]. Postupak obnavljanja snaga pravilima akcijskog skupa je izražen formulom (4.3)

$$\forall p \in [A] \quad p.Snaga(t+1) = (1-\beta) \cdot p.Snaga(t) + \beta \cdot \frac{Nagrada(t)}{|[A]|} \tag{4.3}$$

Dva su mehanizma koja istražuju prostor pravila u ZCS-u: eliminacijski GA i *cover* mehanizam [2]. GA se u svakoj iteraciji pokreće s određenom vjerojatnošću. U GA-u se jednostavnom selekcijom (*roullete-wheel selection*) (u odnosu na snagu pravila) izabiru dva roditelja iz skupa pravila. Roditelji se rekombiniraju (križanje s jednom točkom prekida) s određenom vjerojatnošću i potom se dobivenu djecu (ili klonove roditelja ukoliko se ne dogodi rekombinacija) mutira jednostavnom mutacijom. Iz skupa pravila ([N]) se jednostavnom selekcijom (u odnosu na recipročnu vrijednost snage pravila) određuju dva pravila koja će biti zamijenjena djecom. Time završava ciklus GA-a. *Cover* algoritam se pokreće kada je podudarni skup prazan ili kada je ukupna snaga svih pravila koji se u njemu nalaze za koeficijent Φ slabija od prosjeka. *Cover* algoritam stvara pravilo koje odgovara stanju okoline i koje nadomješta prethodno eliminirano pravilo iz skupa [N]. Snaga novostvorenog pravila se postavlja na srednju vrijednost snaga svih pravila, a izbor pravila za eliminaciju se obavlja jednostavnom selekcijom (u odnosu na recipročnu vrijednost snage pravila). Slika (4.1) prikazuje ZCS sustav.

Svi nedostaci vezani, općenito, uz LCS sustave zasnovane na snazi vrijede i kod ZCS sustava. Osim navedenih nedostataka, potrebno je spomenuti i problem nepotpunog preslikavanja iz kartezijevog produkta skupa stanja okoline i skupa akcija u skup veličina koje predstavljaju korisnost (skup snaga) [1]. Naime, eliminacija pravila se vrši na temelju njihove snage koja je blisko vezana uz nagradu koju pravilo primi od okoline. To znači da će manje kvalitetna biti eliminirana, pa ZCS neće sadržavati informaciju o manje kvalitetnim pravilima.

Ovako opisani ZCS može se proširiti brojnim heuristikama koje pridonose kvaliteti rješenja. Primjerice, eliminacijski GA stvara djecu čija je kvaliteta srednja vrijednost roditelja. Križanjem roditelja sa velikim snagama mogu se dobiti djeca koja se nikada ne koriste (jer im uvjetni dio ne odgovara niti jednom stanju okoline), a imaju velike snage. To povećava prosjek populacije pa pravila koja se neposredno ne nagrađuju, ali dovode do nagrade, mogu biti zapostavljenija od pravila generiranih *cover* algoritmom (onog trenutka kada im se snaga spusti ispod prosjeka snage svih pravila). Određivanjem snage djece nekom drugom heuristikom može se izbjeći nedostatak da populacija konstantne veličine ZCS-a sadrži velik broj pravila koja se nikad ne koriste, a dobivena su genetskim algoritmom.

ZCS pokazuje parametarsku osjetljivost (male promjene nekih parametara mogu dovesti do znatno bolje/lošije klasifikacije), a podešavanjem parametara se, barem za velik skup dobro poznatih problema, može postići optimalna klasifikacija [2].

4.2 Klasifikatorski sustav XCS

Kao logična posljedica nedostataka ZCS-a javila se potreba za njegovim proširivanjem u nešto složeniji model LCS-a. Nedugo nakon stvaranja ZCS-a, Wilson uvodi novi sustav kojeg naziva XCS (*Extended Classifier System*). Osnovna (Wilson-ova) struktura XCS-a se može definirati proširivanjem ZCS-a.
Problemi nepotpunog preslikavanja i pregeneraliziranih pravila doveli su do proširenja korisnosti novim veličinama. Tako u XCS-u pravilo, uz uvjetni i akcijski dio, sadržava još tri veličine: veličinu koja predviđa nagradu okoline (*payoff, p*) (pandem snage u ZCS-u), veličinu koja predstavlja mjeru pogreške predviđanja (*error, ε*) i veličinu koja predstavlja dobrotu (*fitness, F*) pojedinog pravila. Broj pravila u skupu pravila nije konstantan, ali ima gornju među (populacijski limit). U osnovnom modelu XCS-a se koristite mikro-pravila. Ipak, treba spomenuti da modernije verzije XCS-a češće koriste makro-pravila zbog veličine skupa pravila (populacije). Podudarni skup XCS-a je podijeljen u logičke cjeline, tako da svaka logička cjelina sadržava pravila koja predviđaju istu akciju. U sustav se nadodaje *polje predviđanja* (*prediction array, PA*) koje ima ključnu ulogu pri odabiru akcije. U polju predviđanja se za svaku akciju koja ima odgovarajuću logičku cjelinu u podudarnom skupu nalazi predviđena nagrada od okoline. Predviđena nagrada pojedine akcije se dobiva iz predviđenih nagrada pojedinih pravila odgovarajuće logičke cjeline i to tako da se zbroje predviđene nagrade pravila logičke cjeline pomnožene sa udjelom dobrote pojedinog pravila u zbroju dobrota pravila logičke cjeline. Formalni postupak opisuje formula (4.4) [3].

$$\forall a \in \{x | \exists p \in [M] \wedge x = p.Akcija\} \quad PA(a) = \frac{\sum_{p \in [M] \wedge p.Akcija = a} p.Predviđanje(t) \cdot p.Dobrota(t)}{\sum_{p \in [M] \wedge p.Akcija = a} p.Dobrota\,(t)} \tag{4.4}$$

Nakon popunjavanja polja predviđanja određuje se akcija. Akcija se određuje na temelju polja predviđanja. Odabir akcije se može vršiti stohastički (npr. jednostavnom selekcijom), deterministički (npr. izborom akcije s najvećom predviđenom nagradom) ili se kombiniraju oba postupka. Najčešće se koristi *ε-Greedy* strategija. Evaluacija pravila (obnavljanje parametara pravila) se kod XCS obavlja isključivo nad pravilima prethodnog akcijskog skupa [1]. Evaluacija pravila se odvija u sljedećim koracima:

1. Određuje se ukupna nagrada *P* prema formuli (4.5). Ona je jednaka zbroju nagrade okoline s najboljim predviđanjem u polju predviđanja pomnoženim s koeficijentom γ ($0 \leq \gamma < 1$, *discount factor*).

$$P = Nagrada(t-1) + \gamma \cdot \max_{p \in [M]} \big(PA(p.Akcija)\big) \tag{4.5}$$

2. Za svako pravilo predhodnog akcijskog skupa se obnavlja pogreška predviđanja ε. Postupak je izražen formulom (4.6) pri čemu se koristi koeficijent β ($0 \leq \beta < 1$, *learning rate*).

$$\forall p \in [A]_{-1} \quad p.\varepsilon(t+1) = p.\varepsilon(t) + \beta \cdot \big(|P - p.Predviđanje(t)| - p.\varepsilon(t)\big) \tag{4.6}$$

3. Za svako pravilo iz prethodnog akcijskog skupa se obnavlja veličina predviđanja. Postupak je izražen formulom (4.7) pri čemu se koristi koeficijent β ($0 \leq \beta < 1$, *learning rate*).

$$\forall p \in [A]_{-1} \quad p.Predviđanje(t+1) = p.Predviđanje(t) + \beta \cdot (P - p.Predviđanje(t)) \tag{4.7}$$

4. Za svako pravilo iz prethodnog akcijskog skupa određuje se preciznost κ. Postupak je izražen formulom (4.8) pri čemu se koristi konstanta α ($0 \leq \alpha < 1$, *estimation rate*), konstanta ε_0 ($\varepsilon_0 > 0$) i konstanta ν ($\nu > 0$).

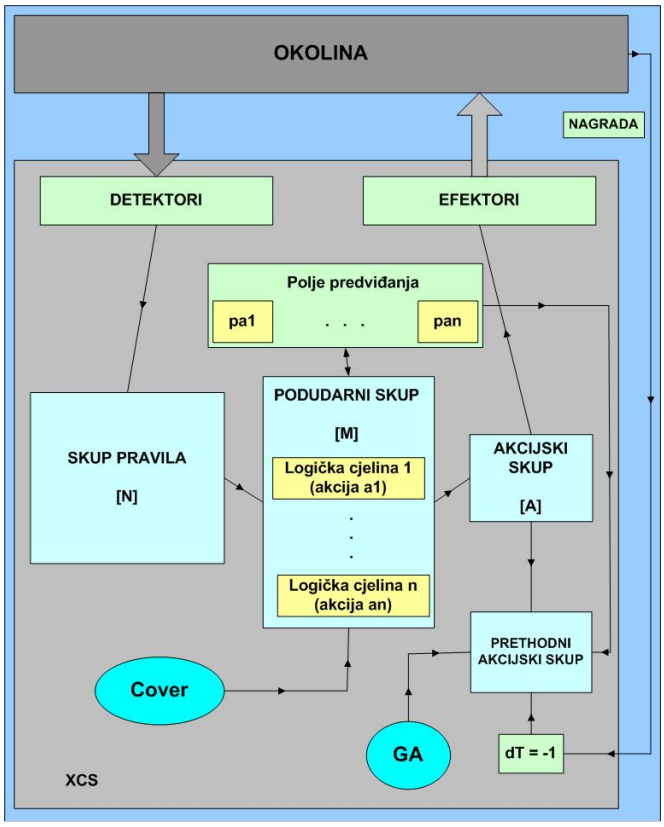
$$\forall p \in [A]_{-1} \quad \kappa(p) = \begin{cases} 1 & ako\ \varepsilon < \varepsilon_0 \\ \alpha \cdot \left(\frac{\varepsilon}{\varepsilon_0}\right)^{-\nu} & ako\ \varepsilon \geq \varepsilon_0 \end{cases} \tag{4.8}$$

5. Za svako pravilo iz prethodnog akcijskog skupa određuje se relativna preciznost κ_r. Relativna preciznost se određuje u odnosu na druga pravila iz prethodnog akcijskog skupa i to tako da je relativna preciznost udio preciznosti određenog pravila u ukupnom zbroju preciznosti pravila iz prethodnog akcijskog skupa. Postupak je izražen formulom (4.9).

$$\forall p \in [A]_{-1} \quad \kappa_r(p) = \frac{\kappa(p)}{\sum_{p \in [A]_{-1}} \kappa(p)} \tag{4.9}$$

6. Za svako pravilo se obnavlja parametar dobrote. Postupak je izražen formulom (4.10) pri čemu se koristi koeficijent β ($0 \leq \beta < 1$, *learning rate*).

$$\forall p \in [A]_{-1} \quad p.Dobrota(t+1) = p.Dobrota(t) + \beta \cdot (\kappa_r - p.Dobrota) \tag{4.10}$$



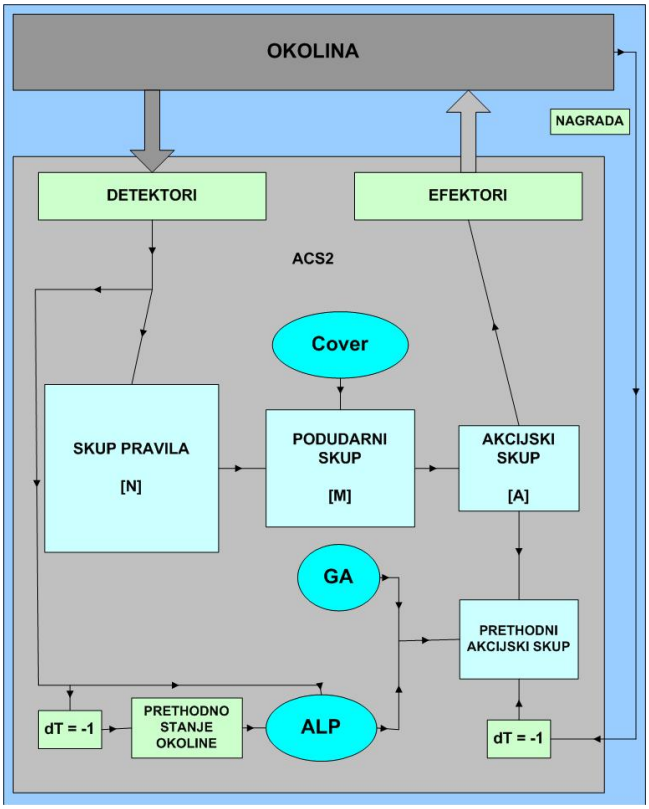
Slika 4.2 XCS sustav

Svakako treba spomenuti da se obnavljanje parametara XCS-a može modificirati tako da se obnavljanje vrši slično kao kod ZCS-a, tj. djelomično nad akcijskim skupom, a djelomično nad prethodnim akcijskim skupom. Pri tome se tehnika nagrađivanja neznatno promijeni pa u konačnici ne uzrokuje znatnije promjene ponašanje sustava.
Kao i kod ZCS-a, mehanizmi pretraživanja prostora pravila su GA i *cover* algoritam. Kod XCS-a se GA poziva nad prethodnim akcijskim skupom (a ne nad skupom pravila) i to periodički (a ne s određenom vjerojatnosti), a *cover* mehanizam se poziva ukoliko je podudarni skup prazan ili ima manji broj akcija od zadanog broja (taj broj se prosljeđuje sustavu kao parametar sustava) [2]. Zbog sličnosti operacija genetskog algoritma i *cover* mehanizma kod XCS-a i ZCS-a, ovdje se navode samo bitne razlike. Osnovni model XCS-a koristi jednostavnu selekciju (*roullete-wheel selection*) (u odnosu na dobrotu pravila). Rekombinacija se obavlja s jednom ili dvije točke prekida, i to samo nad uvjetnim dijelom pravila (jer su akcijski dijelovi pravila isti). Parametri predviđanja nagrade i preciznosti predviđanja pravila djece se postavljaju na srednje vrijednosti odgovarajućih parametara roditelja, dok je parametar dobrote pravila djece desetina srednje vrijednosti dobrota roditelja. Mutacija je jednostavna i obavlja se i nad uvjetnim i nad akcijskim dijelom pravila. Eliminacija se obavlja samo ako je dosegnut populacijski limit (gornja granica broja pravila), i to tako da se iz skupa pravila ([N]) eliminira jedno pravilo izabrano jednostavnom selekcijom (u odnosu na recipročnu vrijednost dobrote pravila).
Ovako opisan sustav pokazuje veću tendenciju k optimalnom rješenju od ZCS-a. Razlog tome je činjenica da XCS nastoji napraviti što potpunije preslikavanje iz skupa stanja u skup akcija, a kvaliteta pravila je određena i preciznošću predviđanja nagrade. No tendencija da se napravi potpuno preslikavanje iz skupa stanja u skup akcija za posljedicu ima da XCS redovito treba raspolagati većim brojem pravila od ZCS-a. Efektivno smanjenje broja pravila se postiže uvođenjem makro-pravila, ali i brojnim dodatnim heuristikama. To su prvenstveno metode obuhvaćanja (*subsumption methods*) koje specijalizirana pravila nastoje zamijeniti već postojećim općenitijim pravilima [18]. Također, i prilikom selekcije jedinke za eliminaciju se mogu koristiti složenije heuristike, a selekcija roditelja za križanje može umjesto *jednostavne* selekcije koristiti neku drugu selekciju (primjerice *turnirsku* selekciju (*tournament selection*)). Uvođenjem metode gradijentnog spusta u sustav nagrađivanja se znatno poboljšavaju performanse sustava [18]. Iako je opisani model XCS-a okrnjen jer ne sadrži iznad navedena poboljšanja, on daje dobar uvid u mogućnosti sustava zasnovanih na preciznosti i njihove prednosti pred, nešto jednostavnijim, sustavima zasnovanima na snazi.

4.3 Klasifikatorski sustav ACS2

Za predstavnika LCS-a zasnovanih na očekivanju odabran je ACS2 (*Anticipatory Classifier System 2*). Osnovnu strukturu ACS-a (*Anticipatory Classifier System*) je prvi izložio Stolzmann 1997., a Butz je stvorio nadopunjenje ACS-a poznato pod nazivom ACS2 [19]. ACS2 i dalje zadržava strukturu sustava kakvu nalazimo u ZCS-u ili XCS-u, a novina je što se dodaje **proces učenja na očekivanju (*Anticipatory Learning Process*, ALP)**. Ovdje se predstavlja pojednostavljeni opis ACS2-a koji daje dobar uvid u nešto drugačiji tip sustava nego što su ZCS i XCS.
Kao i svaki LCS zasnovan na očekivanju, ACS2 sadržava pravila koja, uz *uvjet* i *akciju*, imaju i *očekivanje*. Očekivanje poprima vrijednosti iz ternarnog niza {0, 1, =}, gdje je znak = već spomenuti *don't change* znak. *Očekivanje* predstavlja preslikavanje trenutnog stanja okoline u

očekivano stanje okoline. Pravilo ACS2-a sadržava i tri dodatna parametra koja, između ostalog, određuju i korisnost (kvalitetu) pravila: parametar oznake (*mark*, *M*), parametar kvalitete (*quality*, *q*) i parametar mjere nagrade (*reward measure*, *r*). Parametar oznake sadrži za svaki atribut stanja okoline listu njegovih vrijednosti kada su napravljena kriva predviđanja tog atributa [19]. Parametar kvalitete vrednuje preciznost, a parametar mjere nagrade predstavlja očekivanu nagradu okoline za akciju izvršenu tim pravilom. ACS2 raspolaže makro-pravilima, a veličina populacije određena je populacijskim limitom. Radi jednostavnosti, pri opisu sustava će parametar brojnosti često biti zanemarivan.
Korak iteracije počinje određivanjem stanja okoline i punjenjem podudarnog skupa. Ukoliko je podudarni skup prazan poziva se *cover* algoritam koji popunjava podudarni skup s određenim brojem pravila. Nakon što se postavi podudarni skup sa pravilima koja odgovaraju na stanje okoline slijedi pokretanje ALP procesa, a zatim i RL procesa, nad prethodnim akcijskim skupom [21].



Slika 4.3 ACS2 sustav

ALP proces je najsloženija komponenta ACS2 sustava. Proces se primjenjuje na sva pravila prethodnog akcijskog skupa. Pojednostavljeno, ALP razlikuje dva slučaja:

- Slučaj kada preslikavanje između stanja okoline koje definira pojedino pravilo ispravno određuje trenutno (jer se ALP primjenjuje na prethodnom akcijskom skupu, ali može se reći i sljedeće) stanje okoline.
- Slučaj kada preslikavanje između stanja okoline koje definira pojedino pravilo krivo određuje trenutno (sljedeće) stanje okoline.

U prvom slučaju parametar kvalitete se obnavlja po formuli (4.11) pri čemu se koristi koeficijent β ($0 \leq \beta < 1$, *learning rate*). Novo pravilo se stvara tako da se *uvjet* i *očekivanje*

$$\forall p \in [A]_{-1} \quad p.Kvaliteta(t+1) = (1-\beta) \cdot p.Kvaliteta(t) + \beta \tag{4.11}$$

preoblikuje u ovisnosti o parametru oznake. Primjerice u ACS-u, koji koristi nešto jednostavniju tehniku prepravljanja *uvjeta* i *očekivanja*, se za pravilo *#10#:01:=01=→({{0},{0},{0},{1,0}},q,r)* i stanje okoline u trenutnom (sljedećem) koraku *0010* te stanje okoline u prethodnom (trenutnom) koraku *0100* stvara novo pravilo sa preoblikovanim *uvjetom* i *očekivanjem* tako da novo pravilo poprima oblik *#100:01:=010→({},q',r')*. Drugim riječima, u ACS-u se stvara specijalizirano pravilo na način da se atributi koji su generalizirani, a imaju barem jednu vrijednost u parametru oznake koja se razlikuje od korespondentnog atributa u trenutnom (sljedećem) stanju okoline, promijene u korespondentni atribut trenutnog (sljedećeg) stanja okoline [19]. Kvaliteta novog pravila je polovica kvalitete roditelja. U drugom slučaju parametar kvalitete se obnavlja po formuli (4.12) pri čemu se koristi koeficijent β ($0 \leq \beta < 1$, *learning rate*). Obnavlja se i parametar oznake i to tako da se svaki atribut stanja okoline, ukoliko se već ne nalazi u odgovarajućoj listi parametra oznake, dodaje u odgovarajuću listu u parametru oznake [20]. Ukoliko je moguće poboljšati preciznost pravila specijalizacijom, to se čini stvaranjem specijaliziranog

$$\forall p \in [A]_{-1} \quad p.Kvaliteta(t+1) = (1-\beta) \cdot p.Kvaliteta\,(t) \tag{4.12}$$

pravila. Primjerice, za pravilo *#10#:01:=01=→({{0},{0},{0},{1,0},q,r)* i stanje okoline u trenutnom (sljedećem) koraku *0011* te stanje okoline u prethodnom (trenutnom) koraku *0100*, stvara se pravilo oblika *#100:01:=011→({},q',r')*. Kvaliteta novog pravila je polovica kvalitete roditelja. Ukoliko nema pravila čije je očekivanje ispravno za taj ciklus, ALP djeluje svojim internim *cover* algoritmom kojim se stvara pravilo koje u *uvjetu* i *očekivanju* ima generalizirane znakove # i = u svim atributima za koje su korespondentni atributi trenutnog(sljedećeg) i prethodnog (trenutnog) stanja okoline jednaki, inače poprimaju vrijednosti korespondentnih atributa trenutnog (sljedećeg), odnosno prethodnog (trenutnog) stanja okoline. Kvaliteta tako stvorenog pravila je postavljena na 0.5.
RL proces nagrađuje prethodni akcijski skup, tj. obavlja se evaluacija pravila iz prethodnog akcijskog skupa. Prvo se odredi ukupna nagrada sustava *P* po formuli (4.13) pri čemu se koristi koeficijent γ ($0 \leq \gamma < 1$, *discount factor*). Nakon toga se za svako pravilo koje pripada

$$P = Nagrada(t-1) + \gamma \cdot \max_{p \in [M]}(p.Kvaliteta(t) \cdot p.MjeraNagrade(t)) \tag{4.13}$$

prethodnom akcijskom skupu obnavlja mjera nagrade po formuli (4.14) pri čemu se koristi koeficijent β ($0 \leq \beta < 1$, *learning rate*).

$$\forall p \in [A]_{-1} \quad p.MjeraNagrade(t+1) = p.MjeraNagrade(t) + \beta \cdot \big(P - p.MjeraNagrade(t)\big) \tag{4.14}$$

Nakon obnavljanja parametara pravilima iz prethodnog akcijskog skupa slijedi pokretanje genetskog algoritma nad prethodnim akcijskim skupom. Sustav koristi jednostavnu selekciju (*roullete-wheel selection*) roditelja (u odnosu na treću potenciju parametra kvalitete pravila). Roditelji se rekombiniraju križanjem s dvije točke prekida, a parametri kvalitete i mjere nagrade pravila djece se postavljaju na srednje vrijednosti odgovarajućih parametara pravila roditelja. Mutacija je generalizirana jednostavna mutacija: mijenjaju se samo specijalizirani atributi pravila [21]. Eliminacija se obavlja ukoliko je prethodni akcijski skup popunjen pa ne može primiti dva djeteta. Pri tome se popunjenost prethodnog akcijskog skupa određuje posebnim parametrom. Eliminiraju se dva pravila (nešto modificiranom) turnirskom

eliminacijom (u odnosu na parametar kvalitete pravila i specijaliziranost pravila) i ona se nadomještaju pravilima djecom [20],[21].
Nakon što završi rad genetskog algoritma odabire se akcija sustava. Za odabir akcije se najčešće koristi *ε-Greedy* selekcija (u odnosu na umnožak parametra kvalitete i parametra mjere nagrade). Nakon postavljanja akcijskog skupa i prethodnog akcijskog skupa te izvršavanja odabrane akcije sustav prima nagradu od okoline.
Za razliku od ZCS-a i XCS-a kojima GA služi kao metoda pretraživanja prostora stanja, ACS2 koristi GA kao metodu generalizacije pravila. Naime, ALP proces, zajedno sa svojim internim *cover* mehanizmom, ima tendenciju stvaranja specijaliziranih pravila [20]. Uzrok tome je nastojanje da se ALP procesom utvrdi što točnije predviđanje pojedinih pravila. S druge strane, GA sa križanjem i generaliziranom mutacijom nastoji stvoriti što općenitija pravila, a pri eliminaciji pravila se uzima u obzir i općenitost pravila. Osim GA-a, ulogu stvaranja općenitijih pravila imaju i metode obuhvaćanja (*subsumption methods*) koje se koriste i u nešto složenijim (poboljšanim) modelima XCS-a. Metode obuhvaćanja nastoje pri umetanju pravila u populaciju pronaći pravilo u populaciji koje ga može obuhvatiti: to je pravilo dovoljno dobro i dovoljno generalizirano. Ako pronađe pravilo koje ga obuhvaća, umjesto umetanja novog pravila, povećava se brojnost pravilu koje obuhvaća novostvoreno pravilo.
ALP proces je nastao na temelju Hoffmanove teorije *kontrole ponašanja očekivanjem* (*Anticpatory Behavioral Control, ABC*). Prema toj teoriji ponašanje nije uvjetovano direktnom nagradom okoline već očekivanjem posljedica koje će biti uzrokovane primijenjenom akcijom. Time sustav dolazi do sposobnosti latentnog učenja. Latentno učenje je učenje bez izravnih nagrada, a stečeno znanje se očituje tek kao odgovor na specifični podražaj [5]. Često takvog znanja biće nije ni svjesno. Sposobnost stjecanja znanja latentnim učenjem (uz korištenje učenja potkrjepljenjem (podražajem)) predstavlja prednost LCS sustava zasnovanih na očekivanju. Prednost je posebice izražena u ne-Markovljevim okolinama [19].

5. *Single step* i *Multi step* problemi

Dva tipa problema zanimljiva su za ispitivanje LCS sustava: *single step* (jednokoračni) problemi i *multi step* (višekoračni) problemi. Sa stanovišta LCS sustava, razlika između *single step* i *multi step* problema je ponajprije u povezanosti izvršene akcije sustava i stanja okoline u sljedećem koraku. Ovdje se opisuju *single step* i *multi step* problemi u kontekstu michiganskog stila LCS-a.

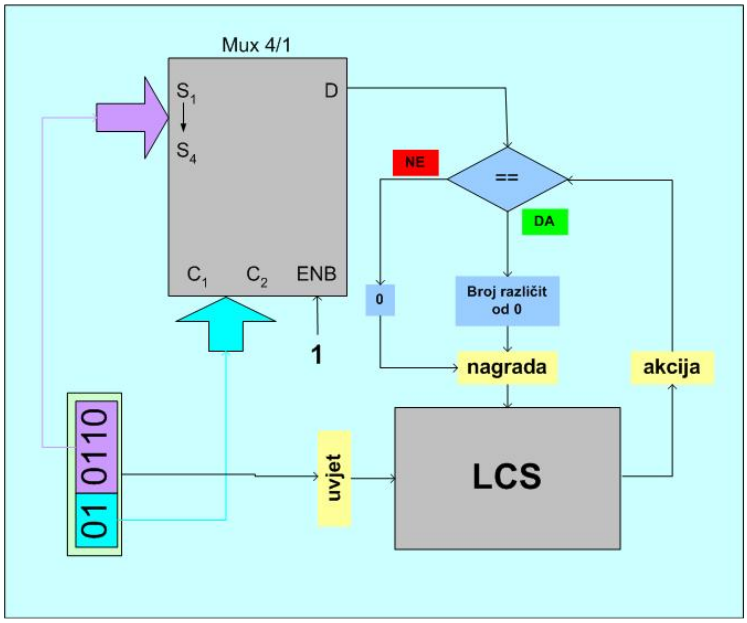
***Single step* problemi** su problemi gdje ne postoji veza između izvršene akcije sustava i sljedećeg stanja okoline. Drugim riječima, stanja okoline se sa stanovišta LCS sustava generiraju slučajno. To znači da rješenje problema traje jedan korak odakle i naziv jednokoračni problemi. Nema povezanosti između uzastopnih akcija što znači da sustav ne treba (i ne smije) graditi lanac akcija. Kako sustav ne bi gradio lanac akcija, u LCS-u je potrebno na kraju svakog koraka prazniti prethodni akcijski skup. Konstanta γ (*discount factor*) nema značaja pa se može postaviti na bilo koju vrijednost. *Single step* problemi su uglavnom problemi klasifikacije. Primjeri *single step* problema su problemi multipleksora i MONK problemi [22].

***Multi step* problemi** su problemi gdje postoji eksplicitna veza akcija sustava i sljedećeg stanja okoline. Drugim riječima, stanje okoline u sljedećem koraku ovisi o akciji sustava. Ovakvo definiranje *multi step* problema je neprecizno. Naime, ne misli se na postojanje isključivo beskonačnog slijeda ovisnosti *akcija→stanje*, već na postojanje barem jednog (konačnog) slijeda ovisnosti *akcija→stanje*. To znači da se traži rješenje u više koraka odakle i naziv višekoračni problemi. Kako bi mogao pronaći rješenje koje zahtijeva više koraka izvođenja, LCS sustav treba sagraditi lanac akcija. U LCS-u postoji implicitna veza između akcija sustava ostvarena sustavom nagrađivanja. Dakle, nije dovoljno da sustav nagradi samo akcije koje eksplicitno dovode do nagrade, nego da nagradi cijeli sustav akcija. Zbog toga su *multi step* problemi teži nego *single step* problemi. *Multi step* problemi spadaju u probleme učenja potkrjepljenjem. Primjeri *multi step* problema su problemi labirinta (*maze problems*), problemi svijeta blokova (*blocks world problems*) i problem automobila u udolini (*mountain car task*) [22].

5.1 Problemi multipleksora

Najčešće se ZCS i XCS sustavi ispituju na problemu multipleksora sa 6 ulaza i problemu multipleksora sa 11 ulaza pa se problemi multipleksora opisuju baš na tim instancama ovog problema.
Problem multipleksora sa 6 ulaza (*mux6*) 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 zahtijeva određivanje ispravnog izlaza iz multipleksora koji će se usporediti s akcijom sustava. Stanje okoline predstavlja slučajno generiran 6-bitni broj. Taj broj se dovodi na ulaz LCS-a, ali i na ulaze multipleksora (odgovarajući bit na odgovarajući ulaz). Sustav vraća akciju u obliku jednog bita. Ukoliko je akcija sustava jednaka izlazu iz

multipleksora, sustav se nagrađuje. Opisani proces prikazan je na slici (5.1). Treba primijetiti da akcije među koracima nisu povezane pa je potrebno na kraju svakog koraka isprazniti prethodni akcijski skup.



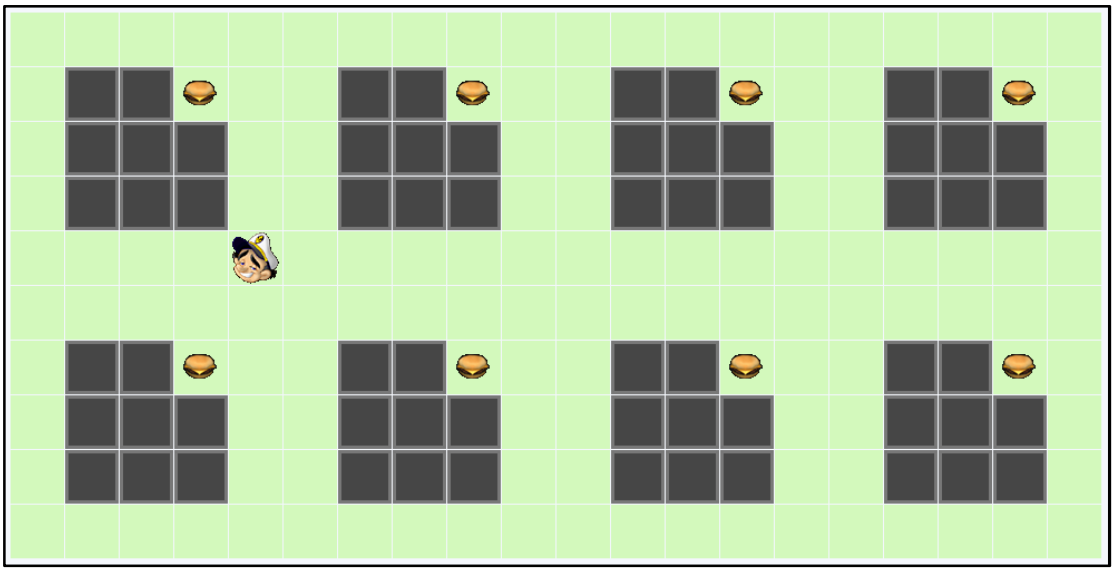
Slika 5.1 Problem multipleksora

Na sličan način se definira i problem multipleksora sa 11 ulaza (*mux11*): problem multipleksora sa 11 ulaza je klasifikacijski problem u kojem je cilj za bilo koji 11-bitni ulaz dati izlaznu vrijednost (0 ili 1) koju bi dao i multipleksor 8/1. Od problema multipleksora sa 6 ulaza se razlikuje jedino u broju bitova uvjeta: uvjet više nije 6-bitni, već 11-bitni broj.
Iako je problem multipleksora naizgled trivijalan, treba uzeti u obzir da je ovo klasifikacijski problem, a ne optimizacijski problem. To znači da sustav sam treba odrediti odnose u ulaznim bitovima da bi uspješno riješio problem. Težina problema multipleksora ovisi o broju ulaza u multipleksor: veći broj ulaza, veća težina klasifikacijskog problema. XCS, kao sustav zasnovan na preciznosti, pokazuje znatno bolja svojstva od ZCS-a, sustava zasnovanog na snazi, u rješavanju problema multipleksora. Posebice se to odnosi na problem multipleksora sa 11 ulaza gdje je ZCS-u potreban znatno veći broj pravila od uobičajenog (400-500 pravila) da bi se dosegla preciznost klasifikacije od 98% i više [25].

5.2 Problemi labirinta

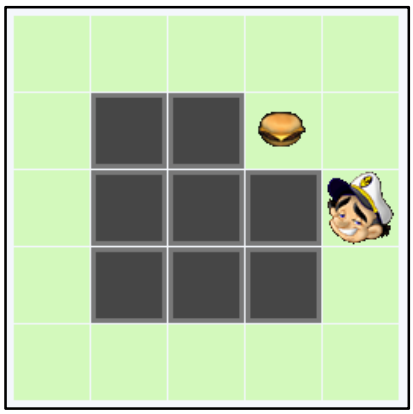
Problem labirinta (*maze problems, woods problems*) spada u skupinu problema učenja potkrjepljenjem (*reinforcement learning problems*). Bitnih razlika između *woods* i *maze* problema nema. Treba napomenuti da su okoline u kojima se testiraju LCS sustavi imenovani i kao *woods* okoline i kao *maze* okoline, a predstavljaju samo različite ploče po kojima se traži hrana. Problem labirinta se može definirati na više načina pri čemu se razlike u definicijama prvenstveno odnose na broj polja koje vidi *animat* (*artificial animal*) oko sebe i na dozvoljene akcije *animata*. Ovdje se koristi definicija problema u kojoj *animat* vidi sva polja oko sebe i u kojoj se može pomaknuti na bilo koje od polja oko sebe.

Srž problema labirinta je *animat* (*artificial animal*) koji je u potrazi za hranom, a upravljan je LCS-om [16]. *Animat* osjeća stanje oko sebe (stanje okoline), i na temelju LCS-a donosi odluku o smjeru kretanja, a cilj mu je u što manje koraka doći do hrane.
Primjer okoline problema labirinta je prikazan na slici (5.2). Okolina sadržava kamenja (prepreke), hranu, polja po kojima se može kretati i *animata.* Komponente okoline (stanja okoline) kodirana su sa 2 bita, primjerice: kamenja s 00, hrana s 11, polja za kretanja sa 10, a *animat* s 01.



Slika 5.2 Okolina woods1a

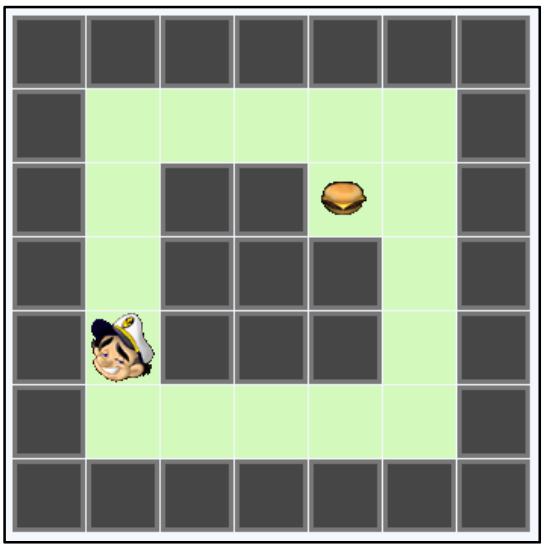
Stanje okoline koje prima *animat* predstavlja 8 polja oko njega: stanje je 16-bitni broj kodiran tako da su redom upisivana stanja okoline s lijeva na desno počevši od gornjeg (sjevernog)



Slika 5.3 Okolina woods1b

polja i nastavljajući u smjeru kazaljke na satu. Akcije *animata* su pomicanje u jedno od tih 8 polja. Akcije koje upućuju *animata* na kamenja se zanemaruju, tj. ako je izabrana takva akcija, *animat* se ne pomiče s mjesta. Akcije koje upućuju *animata* van ploče labirinta se izvode: okolina je *okrugla*. To znači, primjerice, da će pomak *animata* u lijevo s krajnje lijevog mjesta uzrokovati pojavu *animata* na krajnje desnom mjestu (ukoliko se tamo ne

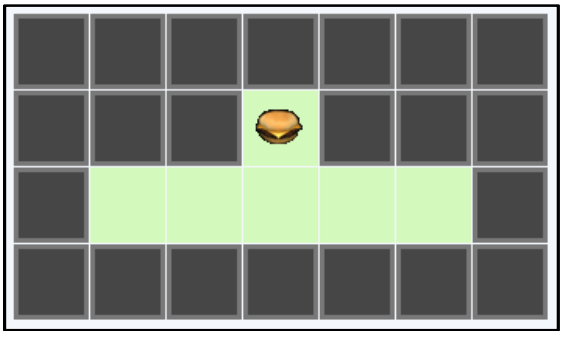
nalazi kamenje). Uzme li se u obzir da je okolina *okrugla*, okolina *woods1a* ekvivalentna je okolini *woods1b* predstavljenoj na slici (5.3), dok je okolina *woods1b* različita od okoline *woods1c* prikazane na slici (5.4).



Slika 5.4 Okolina woods1c

Kada se postavlja pitanje koliko se dobro LCS sustav snalazi na labirintu treba imati na umu da postoje dvije osnovne vrste okolina: Markovljeva okolina (*Markovian environment*) i ne-Markovljeva okolina (*non-Markovian environment*). Općenito je određivanje optimalne strategije do hrane, koristeći bezmemorijski sustav koji oko sebe vidi samo 8 okolnih polja, *NP-potpun* problem [17]. Složenost se smanjuje ako se zada Markovljeva okolina.

Markovljeve okoline su, u problemu labirinta, one okoline u kojima je stanje okoline koje vidi *animat* jednoznačno određeno pozicijom *animata* [16]. Primjerice, to su okoline prikazane na slikama (5.2), (5.3), (5.4) i (5.5).

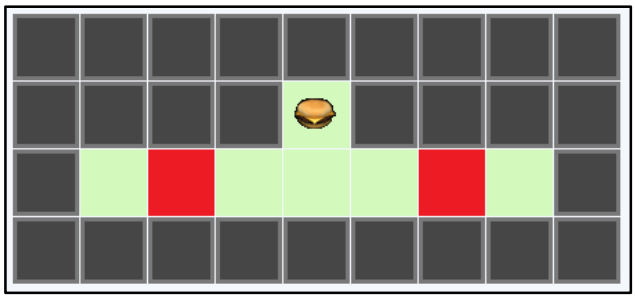


Slika 5.5 Okolina maze1

LCS sustav bolje uči u Markovljevim okolinama zato što se u tom slučaju može definirati funkcija (u strogom smislu) koja preslikava skup stanja sustava u skup akcija, a predstavlja optimalnu strategiju do hrane. To znači da se svako stanje okoline definiranom funkcijom preslikava u točno jednu akciju iz skupa mogućih akcija. Kako skup pravila određuje

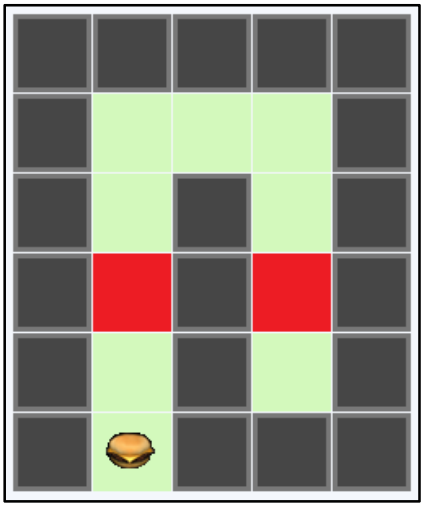
preslikavanje iz skupa stanja u skup akcija, "dovoljno" je pronaći takav skup pravila kojima će se oblikovati optimalna funkcija.

Za **ne-Markovljeve okoline** postoje različite pozicije u kojima *animat* vidi isto stanje okoline. Primjerice, to su okoline prikazane na slikama (5.6), (5.7) i (5.8).



Slika 5.6 Okolina maze2

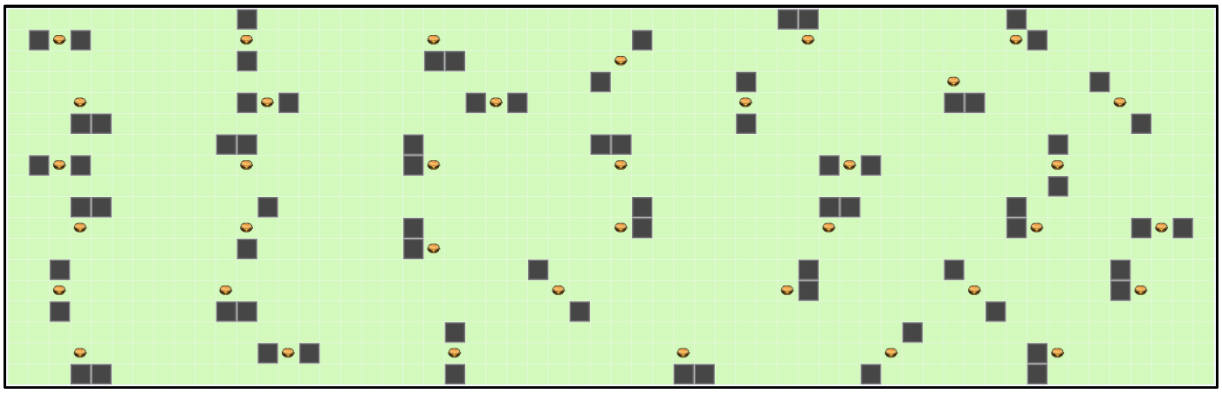
Okolina *maze2*, prikazan na slici (5.6), ima dvije pozicije koje imaju jednaka stanja okolnih polja. Te dvije pozicije su označene crvenom bojom. U lijevom crvenom polju najbolja akcija je kretanje prema desno, dok je za crveno desno polje najbolja akcija kretanje prema lijevo. Kako su obje akcije jednako vrijedne, sustav ne može sa sigurnošću odabrati optimalnu akciju kada se nalazi na crvenom polju.



Slika 5.7 Okolina maze7

Slična situacija se događa u okolini *maze7* prikazanoj na slici (5.7). Okolina *maze7* je nešto složenija od okoline *maze2*, a opisuje situaciju kada sustav treba odlučiti da li da ide gore ili dolje ako se nalazi na crvenom polju. Ipak, u ovom slučaju se sustav češće nalazi na lijevom crvenom polju nego na desnom pa je veća vjerojatnost da će odabrati akciju kretanja prema dolje. No zbog tog izbora on često zapinje desnom crvenom polju jer ga akcija vodi u krivom smjeru. Kako bi uspio izaći iz zamke crvenog desnog polja, sustav treba naučiti pravilo koje ga upućuje prema gore. To ga dovodi do problema kada ponovno dođe na lijevo crveno polje jer je pokvareno pravilo koje govori da sa crvenog polja treba krenuti prema dolje.

Okolina *woods7* prikazana na slici (5.8) je specifična. Iako se doima da LCS sustav nema što naučiti u ovoj okolini, ako se pažljivije pogleda, može se uočiti da je hrana smještena kraj



Slika 5.8 Okolina woods7

kamenja. Dakle, bitnu karakteristiku koju *animat* treba naučiti u *woods7* okolini je da se pokraj kamenja nalazi hrana. Općenito je optimalan prosječni broj koraka do hrane za *woods7* okolinu 2.2, dok je optimalan prosječni broj koraka do hrane za bezmemorijski sustav koji vidi 8 polja oko sebe 3.1 [17].
Ponašanje ZCS-a i XCS-a u ne-Markovljevim okolinama se može značajnije popraviti uvođenjem interne memorije [17], [23].

6. Uvid u implementirane klasifikatorske sustave

Sustavi koji su implementirani i na kojima su obavljena mjerenja se nešto razlikuju od osnovnih modela ZCS-a i XCS-a opisanih u četvrtom poglavlju. Implementirani sustavi se mogu promatrati kao nadopunjenja (proširenja) osnovnih modela ZCS-a i XCS-a pa je dovoljno dati kratak opis bitnih razlika. Treba spomenuti da se u oba implementirana sustava koriste makro-pravila. Sa stanovišta implementiranih sustava, može se smatrati da su makro-pravila sastavljena od mikro-pravila čiji je broj određen parametrom brojnosti u makro-pravilima.

6.1 Proširenje ZCS-a

Iako prošireni model ZCS-a uvelike prati osnovni model ZCS-a, postoje razlike i proširenja čija je svrha poboljšanje rada ZCS-a. Pojednostavljeni pseudokod ZCS-a je prikazan na slici (6.1).

```
Iteracija_ZCS{
     dohvaća se stanje okoline;
     smještaju se pravila (iz populacije pravila) koja
          odgovaraju stanju okoline u podudarni skup;
     ako je podudarni skup prazan ili pravila u podudarnom
          skupu nisu dovoljno dobra, poziva se cover algoritam;
     odabire se akcija sustava pri čemu se uzimaju u obzir
          snage pravila podudarnog skupa;
     pravila iz podudarnog skupa koja imaju akciju jednaku
          odabranoj akciji se smještaju u akcijski skup;
     izabrana akcija se izvršava;
     dohvaća se nagrada od okoline;
     obnovljaju se snage pravila sustava;
     uz određenu vjerojatnost se poziva GA nad populacijom
          pravila;
     pravila iz akcijskog skupa se smještaju u prethodni
          akcijski skup;
}
```

Slika 6.1 Pseudokod ZCS-a

Prva bitnija razlika u odnosu na osnovni model ZCS-a je korištenje makro-pravila umjesto mikro-pravila. Ova preinaka je vođena činjenicom da makro-pravila zahtijevaju manje memorijskog prostora, a trajanje pretraživanja po populaciji je kraće. Populacija nije konstantne veličine, već je veličina ograničena populacijskim limitom (*population limit*) i to u odnosu na ukupan broj mikro-pravila (zbroj brojnosti svih makro-pravila ne smije biti veći od populacijskog limita). Korištenje makro-pravila ponajprije utječe na smještanje pravila u populaciju, micanje pravila iz populacije, križanje pravila, obnavljanje snaga pravila i selekciju pravila.
Pri smještanju novog makro-pravila u populaciju provjerava se postoji li makro-pravilo s jednakim uvjetnim i akcijskim dijelom. Ukoliko postoji, parametar snage postojećeg makro-pravila se postavi na srednju vrijednost snaga mikro-pravila jednog i drugog makro-pravila, a brojnost postojećeg makro-pravila se postavi na ukupan broj mikro-pravila jednog i drugog

makro-pravila. Ako ne postoji pravilo sa jednakim uvjetom i akcijom, novo pravilo se nadodaje u populaciju.
Makro-pravila se eliminiraju tako da im se smanji parametar brojnosti za jedan, a ako brojnost poprimi vrijednost nula, makro-pravilo se miče iz populacije.
Križanje je izmijenjeno utoliko što se rekombinacija pravila vrši samo nad uvjetnim dijelom pravila, a akcijski dijelovi se samo zamjenjuju. Brojnost makro-pravila djece se postavlja na jedan, a snage djece su pomnožene sa koeficijentom manjim od jedan (a većim od nule) da se izbjegne dobivanje nepotrebnih pravila s velikim snagama (vidjeti podpoglavlje 4.1).
Obnavljanje snage pravila je nešto izmijenjeno jer sustav raspolaže s makro-pravilima. Kako je snaga makro-pravila jednaka prosjeku snaga mikro-pravila, formula (4.1) ostaje nepromijenjena. Formula (4.2) se prepravlja u formulu (6.1), a formula (4.3) se prepravlja

$$\forall p \in [A]_{-1} \quad p.Snaga(t+1) = p.Snaga(t) + \gamma \cdot \frac{\sum_{p' \in [A]} \beta \cdot p'.Snaga(t) \cdot p'.Brojnost}{\sum_{p'' \in [A]_{-1}} p''.Brojnost(t)} \tag{6.1}$$

u formulu (6.2). Značenja parametara ostaju nepromijenjena.

$$\forall p \in [A] \quad p.Snaga(t+1) = (1-\beta) \cdot p.Snaga(t) + \beta \cdot \frac{Nagrada(t)}{\sum_{p' \in [A]} p'.Brojnost(t)} \tag{6.2}$$

Prilikom selekcije (pravila u GA-u, pravila koji određuje akciju, pravila za eliminaciju) makro-pravila uzima se u obzir i brojnost pravila. Najčešće se to radi tako da se uobičajena vrijednost po kojoj se vrši selekcija, primjerice snaga pravila, pomnoži s brojnosti pravila.
Osim što su uvedena makro-pravila umjesto mikro-pravila, uvjeti pozivanja *cover* algoritma su nešto prošireni. *Cover* algoritam se i dalje poziva ukoliko je podudarni skup prazan ili je zbroj svih snaga manji za koeficijent Φ od prosjeka snaga, ali poziva se i u slučaju kada je ukupna snaga podudarnog skupa manja od broja koji je u osnovnom modelu predstavljao početnu snagu pravila (*Initial Strength*). Snaga novostvorenog pravila se postavlja na zadani broj (*Initial Strength*). Ovime se želi izbjeći mogući nedostatak promjenjive populacije kod ZCS-a. Naime, sustav može sadržavati mali broj nekvalitetnih pravila koji ne dovode do nagrade, a ispaljuju se u ciklusima. Kako se *cover* algoritam ne poziva (jer podudarni skupovi nisu prazni i zbroj svih snaga u njima nije manji za faktor Φ od prosjeka snaga), a GA ne mora dati bolja pravila (jer križa loša pravila), sustav ne dolazi do nagrade i loše se ponaša u danoj okolini. Osim što se *cover* algoritam poziva kada je ukupna snaga podudarnog skupa manja od zadanog broja, može se uključiti i pozivanje *sekundarnog cover* algoritma (*Secondary Cover*). Sekundarni *cover* algoritam je nadodan u osnovni model ZCS-a za potrebe završnog rada, a cilj algoritma je pretraživanje prostora pravila tako da se pokuša spriječiti zapinjanje u lokalnom optimumu. Sekundarni *cover* je istovjetan običnom *cover* algoritmu samo se poziva uz drugačije uvjete i snaga novostvorenog pravila poprima drugačiju vrijednost: poziva se uz određenu vjerojatnost (*Exploration Rate*), a snaga novostvorenog pravila se postavlja kako je to prikazano formulom (6.3) pri čemu *AvgS(Skup)*

$$p_{SecCov}.Snaga = \begin{cases} rand([0,1\rangle) \cdot AvgS([M]) \cdot \gamma \cdot (1-\beta) \cdot (1-\tau) & za\ AvgS([N]) \geq AvgS([M]) \\ rand([0,1\rangle) \cdot AvgS([N]) \cdot \gamma \cdot (1-\beta) \cdot (1-\tau) & za\ AvgS([N]) < AvgS([M]) \end{cases} \tag{6.3}$$

predstavlja srednju snagu *Skupa*, a *rand* funkcija računa slučajan broj iz zadanog intervala. Sekundarnim *cover* algoritmom se želi postići zamjena kratkih suboptimalnih lanaca akcija boljim (kraćim) lancima, a da se pri tome ne naruše duži optimalni lanci akcija.

Premda ne dolazi uvijek do optimalnog rješenja, ovako proširen ZCS dobro rješava problem labirinta koristeći mali broj makro-pravila.

6.2 Proširenje XCS-a

Proširenje osnovnog modela dodatnim heuristikama i poboljšanjima sustava nagrađivanja znatno unaprjeđuje preciznost klasifikacije XCS-om. Ovdje se daje tek sažet opis proširenja XCS-a, čiji je pojednostavljeni pseudokod prikazan na slici (6.2).

```
Iteracija_XCS{
     dohvaća se stanje okoline;
     smještaju se pravila (iz populacije pravila) koja
          odgovaraju stanju okoline u podudarni skup;
     ako je podudarni skup prazan ili nema dovoljno akcija u
          podudarnom skupu, poziva se cover algoritam;
     generira se polje predvidanja tako da za svaku akciju u
          podudarnom skupu odredi njeno predviđanje nagrade;
     odabire se akciju sustava pri čemu se uzimaju u obzir
          predvidanja akcija;
     pravila iz podudarnog skupa koja imaju akciju jednaku
          odabranoj akciji se smještaju u akcijski skup;
     izabrana akcija se izvršava;
     dohvaća se nagrada okoline i pamti se za sljedeću
          iteraciju;
     obnovljaju se parametri pravila prethodnog akcijskog
          skupa na temelju nagrade iz prethodnog koraka;
     ako je protekao određeni period poziva se GA  nad
          prethodnim akcijskim skupom;
     pravila iz akcijskog skupa se smještaju u prethodni
          akcijski skup;
}
```

Slika 6.2 Pseudokod XCS-a

Prošireni XCS koristi makro-pravila, a populacija je promjenjive veličine i ograničena je populacijskim limitom (kao kod proširenog modela ZCS-a). Pri stavljanju novog makro-pravila u populaciju gleda se postoji li pravilo u populaciji s jednakim uvjetnim i akcijskim dijelom. Ukoliko postoji, postojećem makro-pravilu se poveća brojnost za brojnost makro-pravila koje se dodaje u populaciju. Inače se novo pravilo dodaje u populaciju. Micanje makro-pravila je analogno micanju makro-pravila kod proširenog modela ZCS-a. Pri selekciji makro-pravila uzima se u obzir brojnost makro-pravila.
Osim brojnosti, prošireno XCS makro-pravilo sadržava i tri dodatna parametra: *iskustvo* (*Experience, exp*) koje označava broj pojavljivanja pravila u akcijskim skupovima, *srednju veličinu akcijskih skupova* kojima je pripadalo pravilo (*Action Set Size*, *as*) i *vremensku oznaku* (*Time Stamp, ts*) koja označava vrijeme zadnjeg prisustvovanja u populaciji nad kojom se vršio GA (u odnosu na starost sustava), a inicijalno je postavljena na vrijeme stvaranja pravila.
Iako se polje predviđanja može generirati po formuli (4.4), u implementiranom XCS-u je formula (4.4) modificirana na način da je dobrota makro-pravila pomnožena s njegovom brojnosti (kako u nazivniku, tako i u brojniku). Obnavljanje parametara pravila se obavlja

slično kao i u osnovnom XCS-u opisanom u podpoglavlju 4.2. Formule (4.5) , (4.8) i (4.10) ostaju nepromijenjene. Ako je parametar *iskustva* veći od recipročne vrijednosti parametra β (*p.Iskustvo >1/* β), formule (4.6) i (4.7) ostaju nepromijenjene. U suprotnome, parametar β se zamjenjuje vrijednošću *1/(p.Iskustvo)* kako je to prikazano formulama (6.4) i (6.5). U formuli

$$\forall p \in [A]_{-1} \quad p.\varepsilon(t+1) = \begin{cases} p.\varepsilon(t) + \dfrac{|P - p.Predviđanje(t)| - p.\varepsilon(t)}{p.Iskustvo} & za\ p.Iskustvo < \dfrac{1}{\beta} \\ p.\varepsilon(t) + \beta \cdot \left(|P - p.Predviđanje(t)| - p.\varepsilon(t)\right) & za\ p.Iskustvo \geq \dfrac{1}{\beta} \end{cases} \tag{6.4}$$

(6.5) je *p.Predviđanje(t)* označeno s *p.Pred(t)*. Formula (4.9) se mora prepraviti tako da uzima

$$\forall p \in [A]_{-1} \quad p.Pred(t+1) = \begin{cases} p.Pred(t) + \dfrac{P - p.Pred(t)}{p.Iskustvo} & za\ p.Iskustvo < \dfrac{1}{\beta} \\ p.Pred(t) + \beta \cdot \left(P - p.Pred(t)\right) & za\ p.Iskustvo \geq \dfrac{1}{\beta} \end{cases} \tag{6.5}$$

u obzir i brojnost makro-pravila. Preinaka je prikazana formulom (6.6). Parametri *iskustva* i

$$\forall p \in [A]_{-1} \qquad \kappa_r(p) = \frac{\kappa(p) \cdot p.Brojnost}{\sum_{p' \in [A]_{-1}} \kappa\,(p') \cdot p'.Brojnost} \tag{6.6}$$

srednje veličine akcijskih skupova se također obnavljaju pri evaluaciji pravila. Parametar iskustva se obnavlja na početku evaluacije pravila i to tako da mu se vrijednost poveća za 1.

$$\forall p \in [A]_{-1} \quad p.as(t+1) = \begin{cases} p.as(t) + \dfrac{\sum_{p' \in [A]_{-1}} p'.Brojnost - p.as(t)}{p.Iskustvo} & za\ p.Iskustvo < \dfrac{1}{\beta} \\ p.as(t) + \beta \cdot \left(\displaystyle\sum_{p' \in [A]_{-1}} p'.Brojnost - p.as(t) \right) & za\ p.Iskustvo \geq \dfrac{1}{\beta} \end{cases} \tag{6.7}$$

Parametar *srednje veličine akcijskih skupova* se obnavlja po formuli (6.7) pri čemu *as* označava navedeni parametar. Parametar vremenske oznake se obnavlja u GA-u i tada se postavlja na trenutno vrijeme sustava.

Prošireni XCS ima mogućnost korištenja triju heuristika: metode *gradijentnog spusta* (*gradient descent*), metoda *obuhvaćanja* (*subsumption methods*) i *deletion vote* tehnike.

Metoda *gradijentnog spusta* se koristi pri obnavljanju parametra predviđanja pravila. Parametar predviđanja se obnavlja računajući relativni doprinos pravila u ukupnome zbroju dobrota pravila prethodnog akcijskog skupa. U implementiranom XCS-u je metoda gradijenta izmijenjena tako da je uzeta u obzir i brojnost makro-pravila. Postupak koji se koristi opisan je u formuli (6.8) pri čemu *Pr(t)* predstavlja *Predviđanje(t)*, *F(t)* predstavlja *Dobrotu(t)*, *Br*

$$\forall p \in [A]_{-1} \ p.Pr(t+1) = \begin{cases} p.Pr(t) + \dfrac{P - p.Pr(t)}{p.exp} \cdot \dfrac{p.F(t) \cdot p.Br}{\sum_{p' \in [A]_{-1}} p'.F(t) \cdot p'.Br} & za\ p.exp < \dfrac{1}{\beta} \\ p.Pr(t) + \beta \cdot \left(P - p.Pr(t)\right) \cdot \dfrac{p.F(t) \cdot p.Br}{\sum_{p' \in [A]_{-1}} p'.F(t) \cdot p'.Br} & za\ p.exp \geq \dfrac{1}{\beta} \end{cases} \tag{6.8}$$

predstavlja *Brojnost*, a *exp* predstavlja *Iskustvo*.

Kao što je spomenuto u četvrtom poglavlju, metode obuhvaćanja (*subsumption methods*) se koriste u svrhu generalizacije pravila. Dvije su metode obuhvaćanja poznate u XCS-u: *GA Subsumption* i *Action Set Subsumption*. *GA Subsumption* metoda se poziva u GA-u nakon stvaranja djece. Cilj joj je obuhvatiti stvorenu djecu roditeljima tako da se djeca ne stavljaju u populaciju, već da se roditeljima poveća brojnost za jedan. Roditelj može obuhvatiti dijete samo ako ima dovoljno veliko *iskustvo* i općenitiji je od promatranog djeteta. *Action Set Subsumption* djeluje nad prethodnim akcijskim skupom tako da pronađe najopćenitije pravilo koje ima dovoljno veliko *iskustvo* i njime nastoji obuhvatiti što veći broj pravila prethodnog akcijskog skupa. Ukoliko se neko pravilo može obuhvatiti, to pravilo se izbacuje iz populacije, a povećava se brojnost pravila koje ga obuhvaća za iznos brojnosti izbačenog pravila.
Deletion vote tehnika se koristi pri eliminaciji pravila. *Deletion vote* tehnika određuje u kojoj mjeri je neko makro-pravilo loše, a pri izračunavanju te mjere se uzimaju u obzir brojnost makro-pravila i srednja veličina akcijskih skupova kojima je makro-pravilo pripadalo. Dobrota pravila se razmatra samo ako je pravilo dovoljno iskusno i ako mu je vrijednost dobrote za zadani koeficijent slabije od prosječne vrijednosti dobrote svih pravila u populaciji.
Za GA implementiranog modela XCS-a je bitno spomenuti da se rekombinacija obavlja s jednom točkom prekida, a parametar *pogreške* djece se postavlja na zbroj pogrešaka makro-pravila prethodnog akcijskog skupa, dok se parametri *procjene* i *dobrote* postavljaju kao u osnovnom modelu XCS-a. Brojnost djece poprima vrijednost 1, parametri *iskustva* i *srednje veličine akcijskog skupa* 0, a parametar *vremenske oznake* poprima vrijednost trenutnog vremena sustava.

6.3 Načini rada i vrste korištenih selekcija

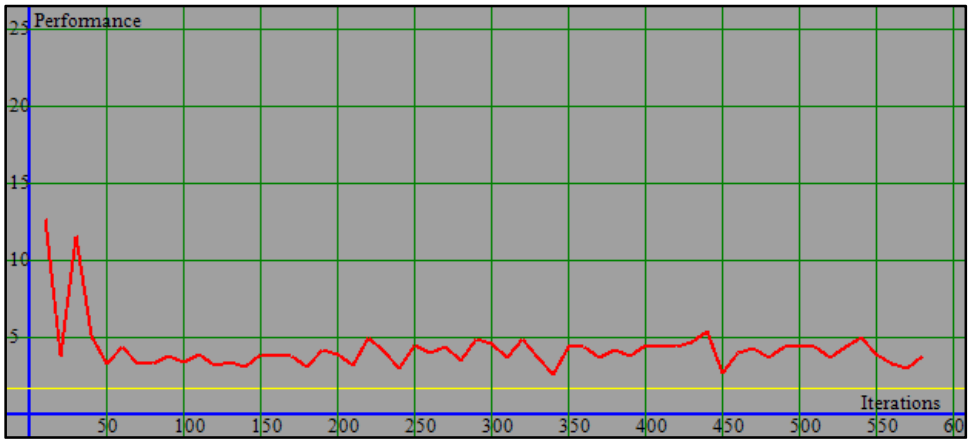
Tri su načina rada stvorenih sustava: *normalni način rada* (*Normal*), *iskorištavanje sustava* (*Exploit*) i *istraživanje mogućnosti sustava (Explore)*. U normalnom načinu rada sustavi rade kako je opisano u prethodna dva podpoglavlja. Kod istraživanja mogućnosti sustava, implementirani XCS odabire akciju ε-*Greedy* selekcijom pri čemu je parametar ε jednak 0.5, dok se u implementiranom ZCS-u akcija odabire jednostavnom selekcijom (*roullete-wheel selection*). Kod iskorištavanja sustava, implementirani XCS odabire najbolju akciju, a implementirani ZCS akciju bira jednostavnom selekcijom (*roullete-wheel selection*). Kada se sustav iskorištava ne poziva se GA, a *cover* algoritam se poziva samo ukoliko je to potrebno (podudarni skup je prazan).
Tri su vrste selekcije ugrađene u implementirane sustave: jednostavna selekcija (*roullete-wheel selection*), turnirska selekcija (*tournament selection*) i ε-*Greedy* selekcija. Sve tri selekcije se mogu koristiti prilikom selekcije roditelja u GA-u. Pri izboru pravila za eliminaciju mogu se koristiti jednostavna selekcija i turnirska selekcija, dok se pri izboru akcije koja će se izvršiti mogu koristiti jednostavna selekcija i ε-*Greedy* selekcija. Pravim odabirom selekcija mogu se znatno poboljšati svojstva klasifikatorskih sustava. Detaljan opis navedenih selekcija može se pronaći u [1], [6], [18] i [24].

7. Eksperimentalni rezultati

Rezultati mjerenja se odnose na implementirane ZCS i XCS klasifikatorske sustave. Prije nego što se prikažu rezultati mjerenja, potrebno je dati opis postupaka mjerenja (koji su implementirani). Nakon opisa postupaka mjerenja slijedi detaljan prikaz rezultata mjerenja.

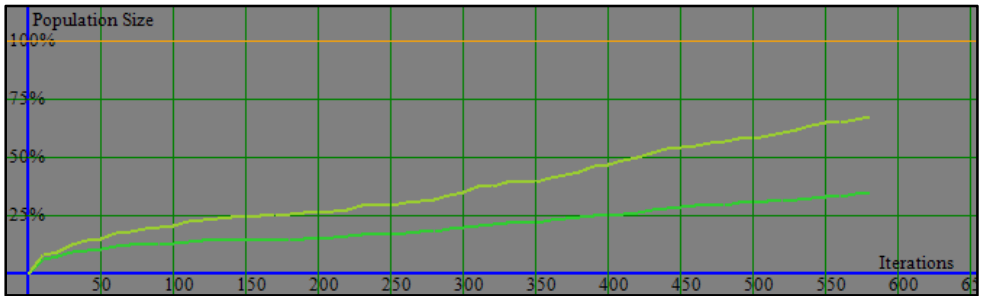
7.1 Postupci mjerenja

Prije nego se opišu postupci mjerenja, potrebno odgovoriti na pitanje što se točno mjeri. Mjere se performanse zadanog sustava, ali i napredovanje veličine populacije sustava. Na slici (7.1) je prikazan graf performansi sustava u odnosu na broj iteracija: crvena linija predstavlja potreban broj koraka do hrane, a žuta linija optimum koji se može postići takvim mjerenjem.



Slika 7.1 Mjerenje performansi klasifikatorskog sustava

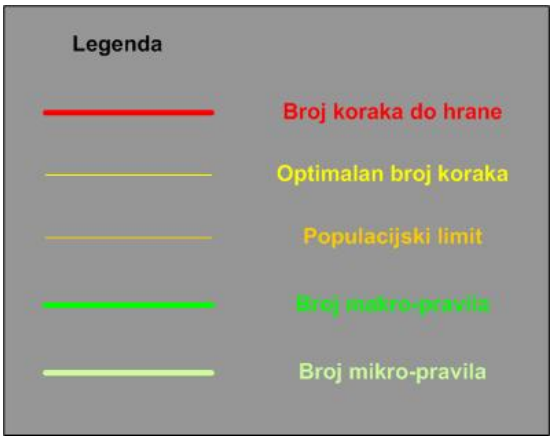
Na slici (7.2) je prikazan graf veličine populacije sustava u odnosu na broj iteracija: narančasta boja predstavlja populacijski limit, zelena broj makro-pravila, a žuto-zelena broj mikro-pravila. Broj mikro i makro pravila se promatra u odnosu na populacijski limit: nije



Slika 7.2 Mjerenje veličine populacije klasifikatorskog sustava

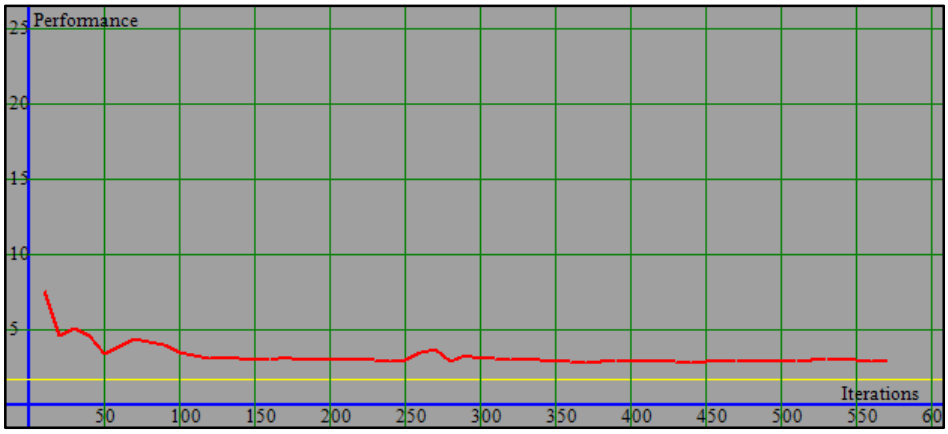
dan konkretan broj, već postotak u odnosu na populacijski limit. Tumač (legenda) pojedinih linija je prikazan na slici (7.3).

Postupak mjerenja se može provesti na više načina. Najjednostavniji način bi bilo smještati *animata* na slučajno odabrane pozicije i računati prosjek mjerenja nekoliko koraka. Ovakav



Slika 7.3 Tumač linija na grafovima

način ima izrazitih mana. Naime, generator slučajnih brojeva ne raspoređuje jednoliko *animata* po ploči pa se rezultati između dva koraka mjerenja znaju razlikovati. Primjerice, na slici (7.1) prikazan je izgled tako provedenog eksperimenta. Na slici se vide velike oscilacije iako se eksperiment vršio nad okolinom *woods1b* i iako se u jednoj iteraciji zapravo računa prosjek od 10 koraka. Povećanjem broja koraka se mogu donekle smanjiti oscilacije. Drugi način mjerenja se može ostvariti tako da se *animata* smješta na svako slobodno područje, a rezultat jednog mjerenja bi bio prosjek koraka od svake slobodne pozicija. Pri tome se redoslijed smještanja iz iteracije u iteraciju mijenja (da se *animatu* ne olakša učenje). Na slici

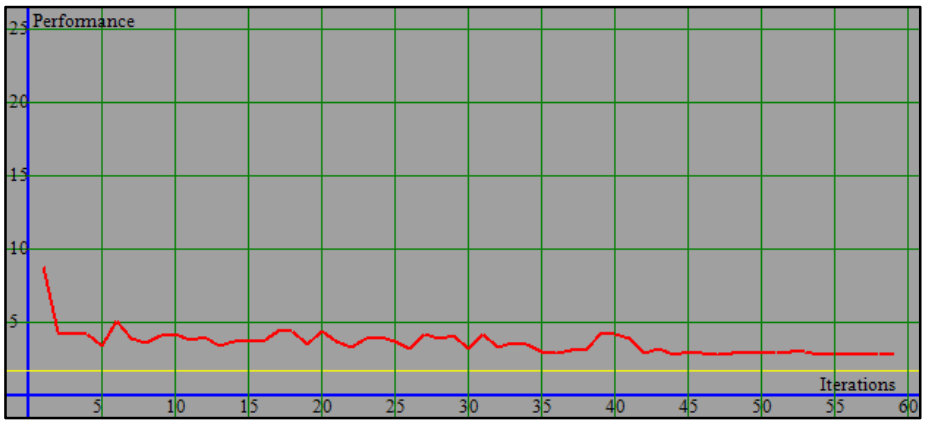


Slika 7.4 Prikaz grafa performansi kada se računa prosjek za sva slobodna polja (10 koraka)

(7.4) je prikazano kretanje grafa performansi za tako načinjen eksperiment: oscilacije se značajno smanjuju. Mjerenje prikazano na slici (7.4) je izvedeno uz prosjek od 10 koraka. Čak i ako se uzme samo jedan korak mjerenja oscilacije mjerenja se smanjuju kako je to prikazano na slici (7.5).

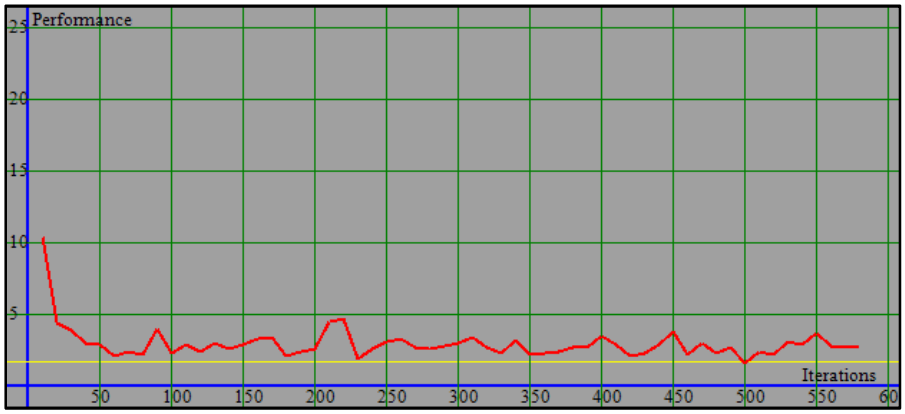
Treća vrsta mjerenje je nešto drugačija od prethodne dvije. Prilikom mjerenja se *animat* može stavljati slučajno na bilo koja polja kao u prvoj strategiji (postupku mjerenja) ili se može računati prosjek mjerenja pri stavljanju na sva prazna polja kao u drugoj strategiji (postupku

mjerenja). Mjerenje se obavlja tako da se u jednom koraku sustav istražuje (*explore*), a potom se *animat* stavlja na isto početno polje (kao u prethodnom koraku) i sustav se iskorištava (*exploit*). Prilikom iskorištavanja sustava se nastoji odabrati najbolje što sustav daje (tako se u



Slika 7.5 Prikaz grafa performansi kada se računa prosjek za sva slobodna polja (1 korak)

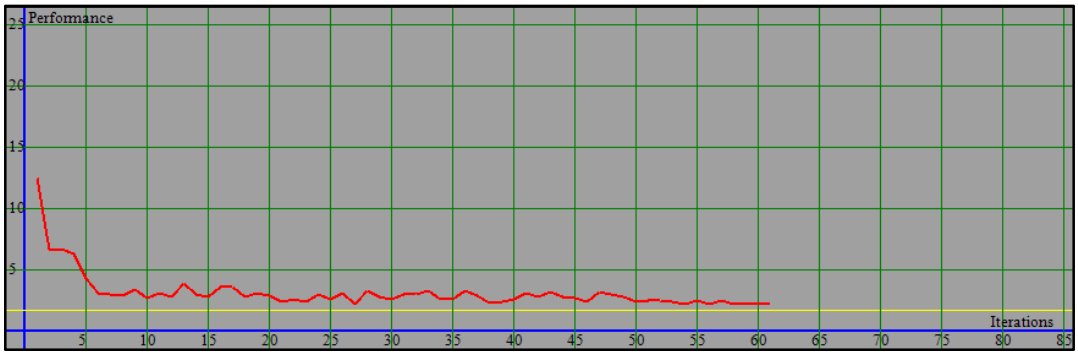
XCS-u odabire najbolja akcija u polju predviđanja), a u fazi istraživanja se teži da sustav što više oko sebe istraži (tako se u XCS-u koristi *ε-Greedy* strategija s ε parametrom 0.5). U fazi iskorištavanja sustava se ne poziva GA, a *cover* algoritam se poziva samo ako je podudarni skup prazan. Ovako definirana strategija predstavlja modifikaciju *explore-exploit* strategije. Na slici (7.6) su prikazani rezultati mjereni modificiranom *explore-exploit* strategije u kombinaciji s prvim načinom mjerenja. Iz slike (7.6) se vidi da je ovakva strategija bolja od prve strategije, ali oscilacije i dalje ostaju prevelike.



Slika 7.6 Explore-exploit strategija sa slučajnim postavljanjem animata (10 korak)

Ako se koristi kombinacija modificirane *explore-exploit* strategije s drugom strategijom (drugim načinom mjerenja), dobivaju se rezultati prikazani na slici (7.7). Naizgled su dobiveni rezultati jednaki onima koji su se dobili korištenjem druge strategije, no ako se malo bolje pogleda, korištenjem kombinacije modificirane *explore-exploit* strategije s drugom strategijom se dođe bliže optimalnom rješenju nego kada se druga strategija koristi zasebno. Svi gore navedeni rezultati prikazuju performanse ZCS sustava na *woods1b* okolini.
U eksperimentalnim mjerenjima prikazanim u nastavku najčešće korišten postupak mjerenja je kombinacija modificirane *explore-exploit* strategije s drugom strategijom. Ukoliko posebno

nije naglašen drugi način mjerenja, u nastavku teksta se podrazumijeva da je mjerenje obavljeno upravo ovom strategijom.



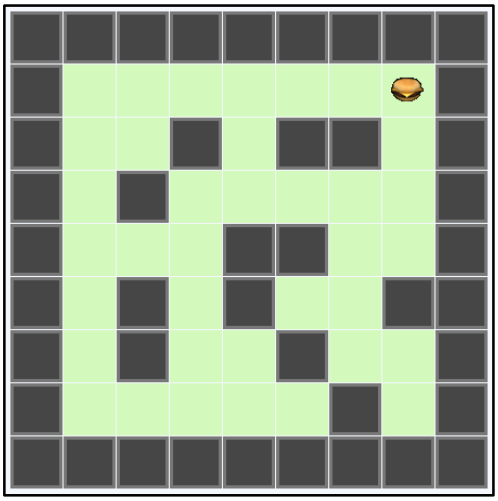
Slika 7.7 Explore-exploit strategija računajući prosjek svih slobodnih polja (1 korak)

7.2 Ponašanje ZCS-a u okolinama labirinta

Kako bi dobili potpunu sliku ponašanja ZCS-a u okolinama labirinta, potrebno je provesti mjerenja na nekoliko tipičnih okolina. Uz mjerenja učinkovitosti, potrebno je dati i mjerenja kretanja populacije ZCS-a. Vrijednosti parametara koje su korištene prilikom mjerenja uglavnom odgovaraju onima u tablici (B.1), a izuzeci su posebno naglašeni.

7.2.1 Ponašanje ZCS-a u Markovljevim okolinama

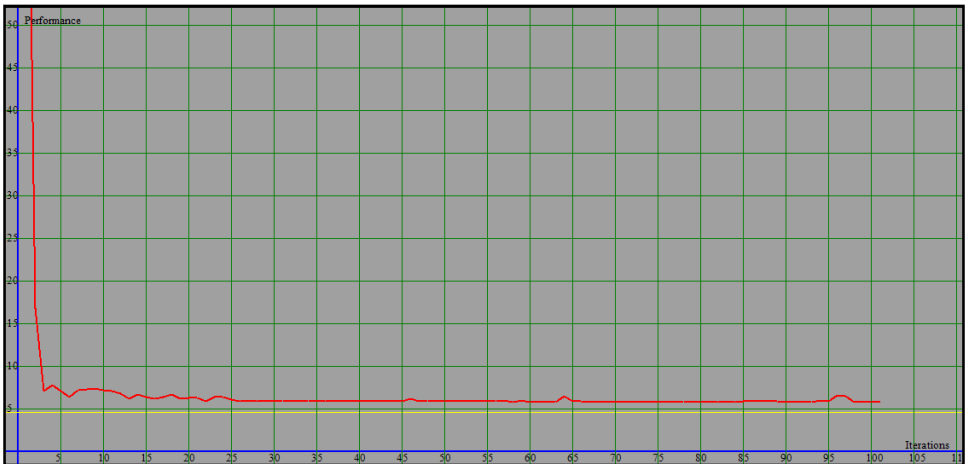
Kao primjer Markovljeve okoline uzeta je okolina *maze5* prikazana na slici (7.8). Optimalan



Slika 7.8 Okolina maze5

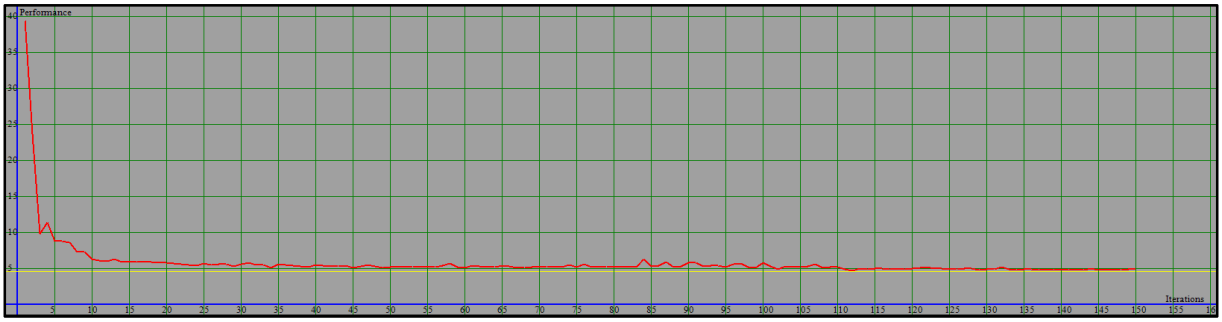
prosječni broj koraka je 4.611. Ukoliko se koristi drugi način mjerenja i ukoliko se ne koristi sekundarni *cover* algoritam, dobiju se rezultati prikazani na slici (7.9). Nakon 100 iteracija prosječan broj koraka do hrane je otprilike 6 i ne spušta se. Drugim riječima, ZCS je zaglavio

u lokalnom optimumu i ne može izaći iz njega. Kako bi se razbili lanci koji vode k suboptimalnom rješenju koristi se sekundarni *cover* algoritam i *explore-exploit* strategija u



Slika 7.9 Ponašanje ZCS-a u maze5 okolini bez sekundarnog cover algoritma (1 korak)

kombinaciji s drugim načinom mjerenja. Na slici (7.10) su prikazani rezultati ZCS-a uz tako načinjen eksperiment. Ono što se može primijetiti je da ZCS nakon 150 iteracija dolazi jako blizu optimuma. Mala titranja na grafu su posljedica stvaranja novih pravila čiji je cilj stvoriti nove lance akcija koji će zamijeniti suboptimalne lance akcija. Nažalost, u tom procesu se mogu i neke dobre akcija zamijeniti lošijima pa su moguća manja titranja kao što je vidljivo na slici (7.10). Korak mjerenja na slikama (7.9) i (7.10) je 1.



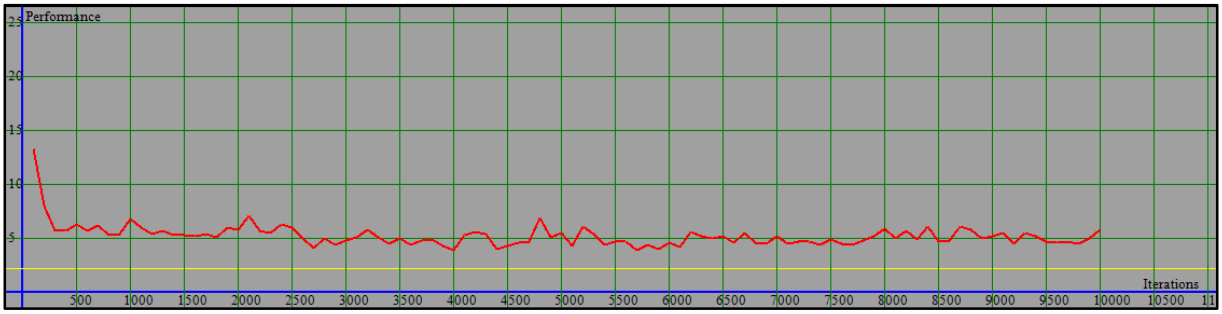
Slika 7.10 Ponašanje ZCS-a sa sekundarnim cover algoritmom u maze5 okolini (1 korak)

Dakle, ZCS se ponaša vrlo dobro u Markovljavim okolinama. Iako ne doseže optimalan broj koraka, dolazi vrlo blizu optimuma, a manja titranja su posljedica nedovoljnog broja parametara kojima se određuje korisnost (kvaliteta) pravila.

7.2.2 Ponašanje ZCS-a u ne-Markovljevim okolinama

Kao dobar primjer na kojem se može testirati ponašanje ZCS-a u ne-Markovljevim okolinama, uzima se okolina *woods7* prikazana na slici (5.8). Kao što je već spomenuto u petom poglavlju, optimalan prosječni broj koraka do hrane za bezmemorijski sustav koji vidi 8 polja oko sebe iznosi 3.1. Na slici (7.11) su prikazani rezultati mjerenja. Selekcija pravila za eliminaciju je obavljana jednostavnom selekcijom (*roullete-wheel selection*) u nastojanju da

se poveća omjer broja makro-pravila i populacijskog limita. Naime, turnirska eliminacija sa proporcijom turnira od 0.4 ima veću tendenciju smanjivanja navedenog omjera od jednostavne eliminacije (*roullete-wheel elimination*). Kako je okolina velika, za očekivati je da će veći broj makro-pravila doprinijeti boljem rješenju. Također, zbog veličine okoline, smještanje *animata* se obavlja slučajnim postupkom, tj. koristi se kombinacija prvog načina mjerenja i *explore-exploit* strategije. Korak mjerenja je 100.



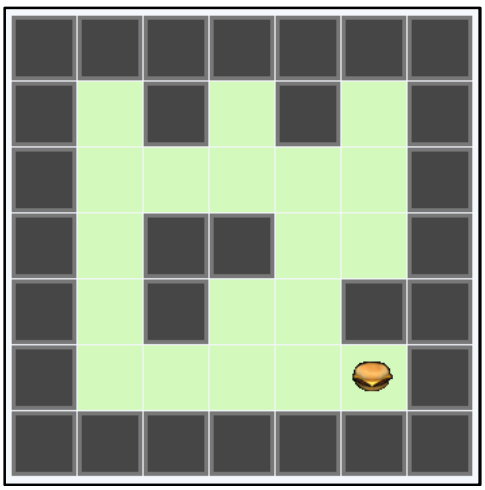
Slika 7.11 Ponašanje ZCS-a u woods7 okolini (100 koraka)

Iz slike (7.11) se vidi da ZCS-u treba prosječno oko 5 koraka do hrane. Titranja u prikazanom grafu su velika zbog velikog broja polja oko kojih se nalaze samo prazna polja. Općeniti optimum od 2.2 koraka (žuta linija) korištenjem (implementiranog) ZCS-a nije dosežljiv, no prosječan broj koraka ZCS-a do hrane se nije spustio ni na 3.1. Ipak, ponašanje koje bi trebalo biti zanimljivije, ne može se vidjeti na slici (7.11). Naime, pogleda li se kako se ZCS kreće po okolini *woods7*, može se uočiti da ZCS uspije zaključiti kako se hrana nalazi u okolici kamenja. U nastavku se ZCS testira isključivo na Markovljevim okolinama.

7.2.3 Prilagodba ZCS-a na promjenu položaja hrane

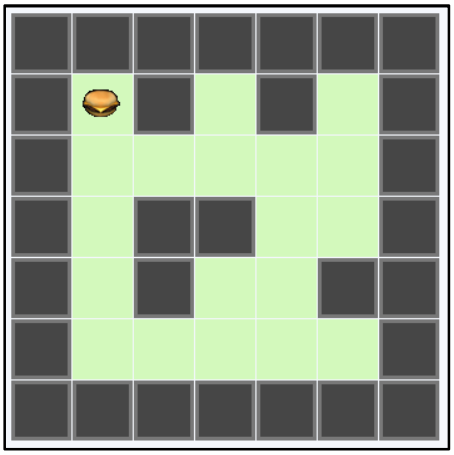
Zanimljivo je pogledati kako se ZCS sustav ponaša kada se promjeni položaj hrane. Treba razlikovati dva slučaja:

- novi položaj hrane je daleko od prvotnog položaja;
- novi položaj hrane je blizu prvotnog položaja hrane.



Slika 7.12 Okolina maze228

Na slici (7.12) je prikazana okolina *maze228* na kojoj se vrši ispitivanje ponašanja ZCS sustava. Prvotni položaj hrane je u donjem desnom kutu, a optimalan prosječni broj koraka do hrane je 3.16. Nakon što se izvrti 100 iteracija, promijeni se položaj hrane u gornji lijevi kut kako je prikazano na slici (7.13). Optimalan prosječni broj koraka do hrane sada iznosi 3.33.



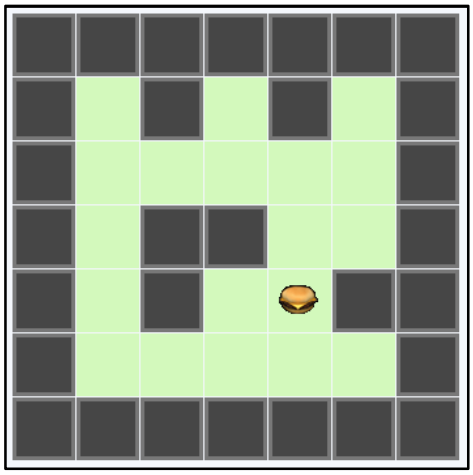
Slika 7.13 Okolina maze228 uz veliku promjenu položaja hrene

Nakon promjene položaja hrane, nastavlja se testiranje sustava sljedećih 100 iteracija. Rezultati mjerenja su prikazani na slici (7.14). Narančasta linija u grafu predstavlja optimum koji se mogao doseći u prvotnom položaju hrane, a žuta linija optimum koji se može doseći u trenutnom položaju hrane. Korak mjerenja je 1.

Slika 7.14 Ponašanje ZCS-a na veliku promjenu položaja hrane (1 korak)

Na slici (7.14) valja primjetiti veliki zaokruženi šiljak. Taj šiljak je uzrokovan upravo velikom promjenom položaja hrane. Hrana se smjestila daleko od prvotnog položaja, a optimalan prosječan broj koraka je porastao, pa je sustav trebao značajno promijeniti slijed akcija da bi dosegao optimalan prosječni broj koraka do hrane. Drugim riječima, sustav je trebao zamijeniti stara pravila novima. Zbog toga je zaokruženi šiljak viši od visine krivulje na početku cijelog procesa. Bitno je spomenuti da su se prvotni lanci akcija trebali u potpunosti promijeniti kako bi se došlo blizu optimalnog rješenja. Bez obzira što je trebao naučiti iznova sva pravila, ZCS je uspio promijeniti prvotno ponašanje i na neki način pokazao sposobnost adaptacije na nagle promjene u okolini.
Neka se sada ponovi eksperiment, ali neka se hrana stavi blizu prvotne pozicije. Neka je novi položaj hrane odmah do prvotnog položaja hrane kako je to prikazano na slici (7.15). Sada je

optimalan prosječni broj koraka do hrane 2.278 i manji je od prvotnog optimuma. Za očekivati je da će se smanjiti šiljak uzrokovan naglom promjenom položaja hrane.



Slika 7.15 Okolina maze228 uz malu promjenu položaja hrene

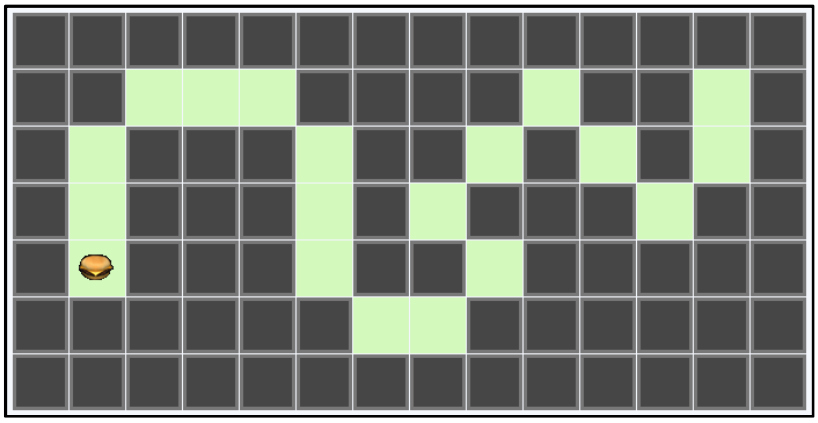
Rezultati mjerenja su prikazani na slici (7.16). Šiljak uzrokovan promjenom položaja hrane je jako mali. Treba napomenuti da šiljak može biti i nešto viši nego što je onaj prikazan na slici (7.16). Kako je promjena položaja hrane mala, ZCS se vrlo brzo uspio prilagoditi na promjenu položaja hrane. Razlog tomu je činjenica da je svega nekoliko pravila trebalo biti promijenjeno (nadodano) da bi se dobili lanci akcija koji vode k optimumu. Dakle u prvotnim lancima akcija su promijenjene samo krajnje akcije.

Slika 7.16 Ponašanje ZCS-a na malu promjenu položaja hrane (1 korak)

7.2.4 Ponašanje ZCS-a u okolinama koje zahtjevaju duge lance akcija

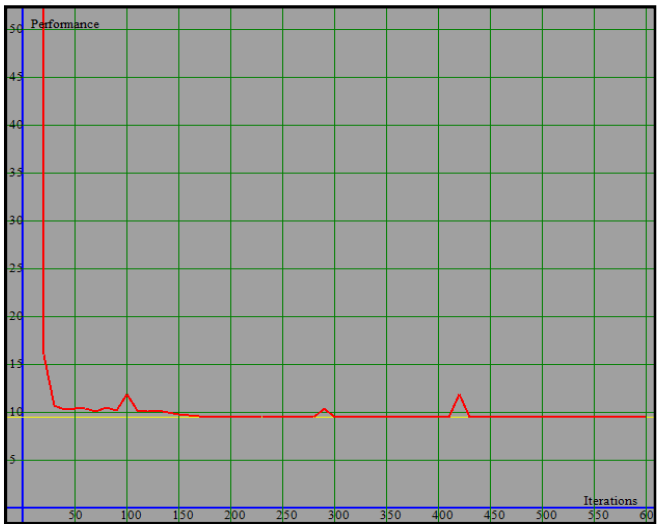
Kao primjer okoline koja zahtijeva duge lance akcija uzima se *woods14* okolina prikazana na slici (7.17). Prosječan optimalan broj koraka do hrane je 9.5, a najdulji lanac akcija koji se zahtijeva sadrži 18 akcija. Okolina *woods14* je specifična, jer, ako se malo bolje pogleda, postoji samo jedan dvosmjerni put. Ova činjenica znatno olakšava učenje.
Kako bi ZCS mogao dosegnuti optimalno rješenje potrebno je podesiti parametre. Koristi se drugi način mjerenja, sekundarni *cover* se ne koristi, a parametar γ je postavljen na 0.9. Vrijednost parametra γ uvelike utječe na povezanost slijeda akcija. Kako bi se povećala ovisnost slijeda akcija, γ se postavlja na relativno veliku vrijednost. ZCS sa ovako podešenim parametrima dobro funkcionira u okolini *woods14* ponajprije zbog specifičnosti *woods14*

okoline (postojanje samo jednog dvosmjernog puta). Potreba podešavanja parametara u cilju pronalaženja optimalnog rješenja kod nekih specifičnih okolina negativna je strana ZCS-a.



Slika 7.17 Okolina woods14

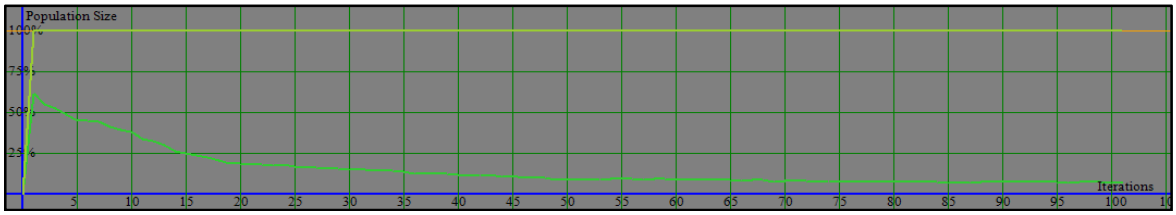
Rezultati mjerenja prikazani su na slici (7.18). Iz rezultata je vidljivo da ZCS već u 200 iteracija dosegne optimum, a pojava manjih šiljaka posljedica je pokušaja sustava da pronađe bolje pravilo (iako ono ne postoji). Naime koristi se jednostavna selekcija (*roulette wheel selection*) akcije pa postoji vjerojatnost izbora lošije akcije. Korak mjerenja je 10.



Slika 7.18 Ponašanje ZCS-a u woods14 okolini (10 koraka)

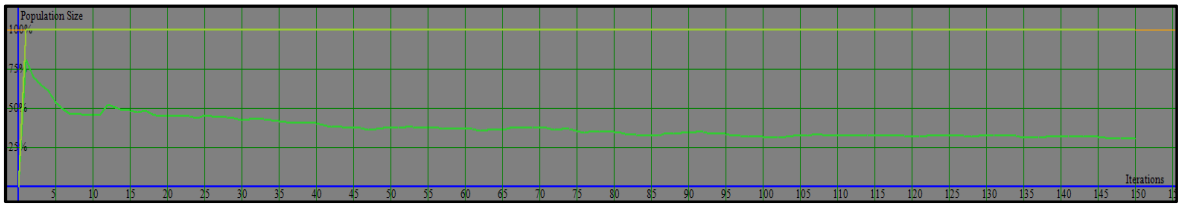
7.2.5 Kretanje populacije ZCS-a

Kako implementirani ZCS koristi makro-pravila, ima smisla pogledati kako se kreće broj makro-pravila i broj mikro-pravila u ovisnosti o broju iteracija. Kretanje populacije prvog mjerenja u *maze5* okolini je prikazano na slici (7.19). Iz slike se vidi da broj mikro-pravila naglo naraste na populacijski limit, a broj makro-pravila na početku raste do 60% vrijednosti populacijskog limita, a onda lagano pada do vrijednosti koja je između 5-10% populacijskog limita. Iako je ovakvo kretanje populacija vrlo dobro, populacija se pri kraju mjerenja sastoji



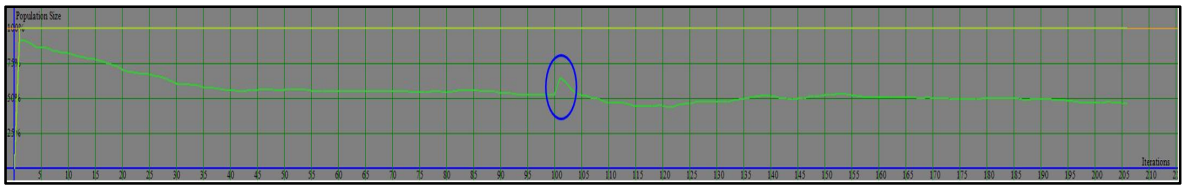
Slika 7.19 Kretanje populacije u maze5 okolini za prvi slučaj (1 korak)

od pravila koja stvaraju suboptimalne lance akcija. Kako bi se to sprječilo, u drugom mjerenju ponašanja ZCS-a u *maze5* okolini korišten je sekundarni *cover*. Njime se nastoje razbiti kratki suboptimalni lanci akcija dodavanjem novih makro-pravila u populaciju. Tako se na slici (7.20) vidi da je pri kraju mjerenja broj makro-pravila na 30% populacijskog limita. Nažalost, kretanje populacije više nije lijepo kao u prethodnom slučaju.



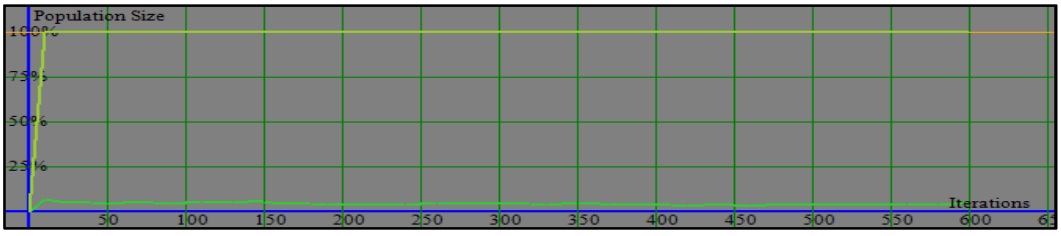
Slika 7.20 Kretanje populacije u maze5 okolini za drugi slučaj (1 korak)

Kod eksperimenata gdje se mijenja pozicija hrane, kretanje populacije ima jedan skok uzrokovan promjenom položaja hrane. Kretanje populacije kod promjene položaja hrane u *maze228* okolini (za prvi slučaj) je prikazano na slici (7.21). Karakteristični skok je zaokružen.



Slika 7.21 Kretanje populacije prilikom velike promjene položaja hrane (1 korak)

Zanimljivo je pogledati i kretanje populacije ZCS-a u woods14 okolini. Populacija se pred kraj mjerenja sastoji od malog broja makro-pravila (otprilike 5% populacijskog limita). Mali broj pravila je potreban jer okolina sadrži samo jedan (dvosmjerni) put.



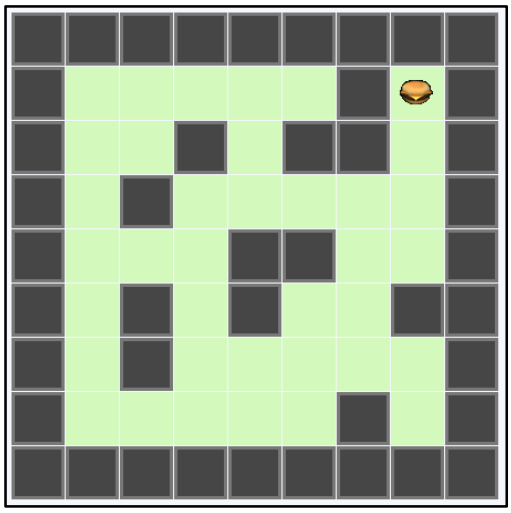
Slika 7.22 Kretanje populacije u woods14 okolini (10 koraka)

7.3 Ponašanje XCS-a u okolinama labirinta

Mjerenja XCS-om se provode na sličan način kao mjerenja ZCS-om, ali na nešto drugačijim okolinama labirinta. Tipične vrijednosti parametara su dane u tablici (B.1), a izuzeci su posebno naglašeni.

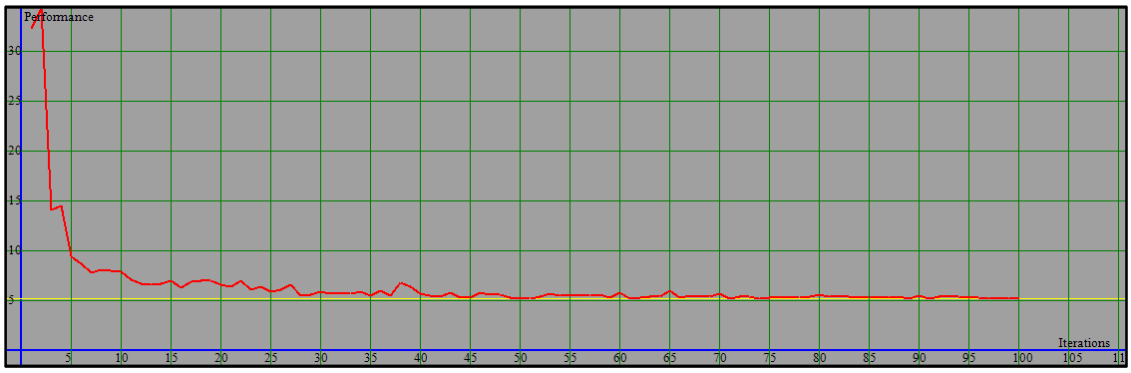
7.3.1 Ponašanje XCS-a u Markovljevim okolinama

Kao primjer Markovljeve okoline ovdje se uzima okolina *maze6* prikazana na slici (7.23). Okolina *maze6* je nešto teža od okolina *maze5* okoline kojom se testirao ZCS, a optimum koji



Slika 7.23 Okolina maze6

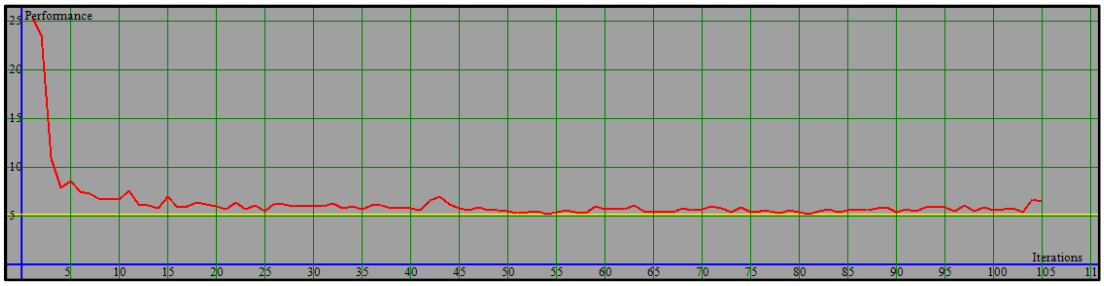
se može doseći je 5.194. Ukoliko se koriste vrijednosti parametara dane u tablici (12.1), dobivaju se rezultati prikazani na slici (7.24). Optimum je dosegnut nakon 100 iteracija. Iz slike (7.24) se vidi da jednom kada XCS dosegne optimalno rješenje, on se u njemu stabilizira. Preciznije bi bilo reći da su oscilacije koje se znaju događati, nakon što se pronađe



Slika 7.24 Ponašanje XCS-a u maze6 okolini uz korištenje metode gradijenta (1 korak)

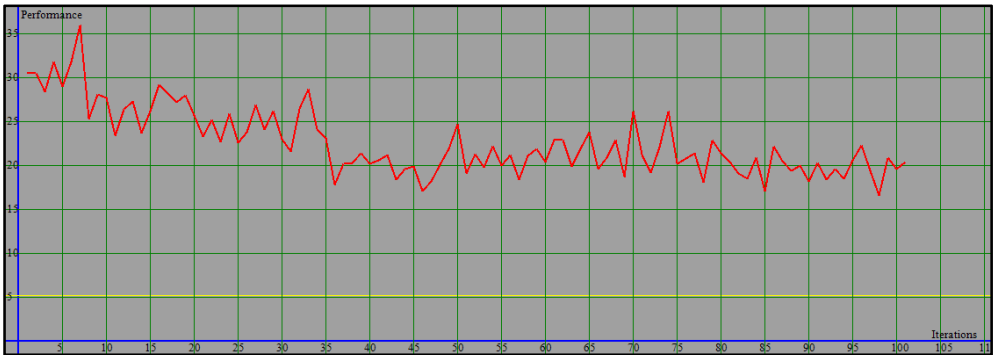
optimum, male. Ukoliko se ne koriste metoda gradijentnog spusta, metoda *GA Subsumption* i tehnika *deletion vote*, dobivaju se dvojaki rezultati. Povoljniji slučaj je prikazan na slici

(7.25a). U povoljnijem slučaju XCS je došao blizu optimuma i iznad njega počeo oscilirati. Drugi slučaj je prikazan na slici (7.25b). U nepovoljnijem slučaju oscilacije su velike, a XCS



Slika 7.25a Ponašanje XCS-a u maze6 okolini bez korištenja metode gradijenta (1 korak)

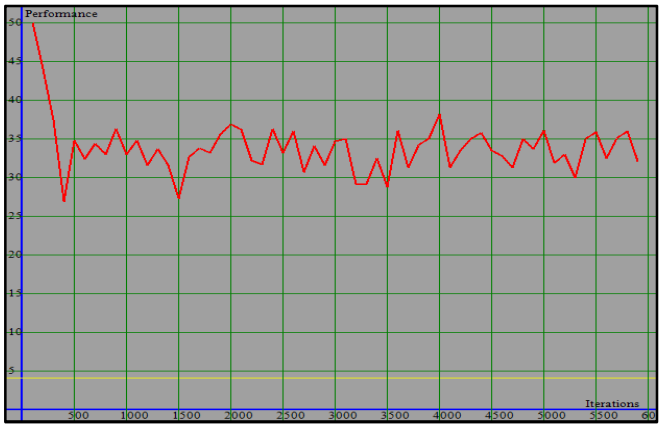
ne dolazi niti blizu optimuma. Dobivanje dvojakih rezultata je uzrokovano činjenicom da se ne koristi metoda gradijentnog spusta i tehnika *deletion vote*. Naime, metoda gradijentnog spusta nastoji što "pravednije" obnoviti parametar predviđanja, a *deletion vote* tehnika pri eliminaciji pravila ne uzima samo parametar dobrote pravila (vidjeti podpoglavlje 6.2).



Slika 7.25b Ponašanje XCS-a u maze6 okolini bez korištenja metode gradijenta (1 korak)

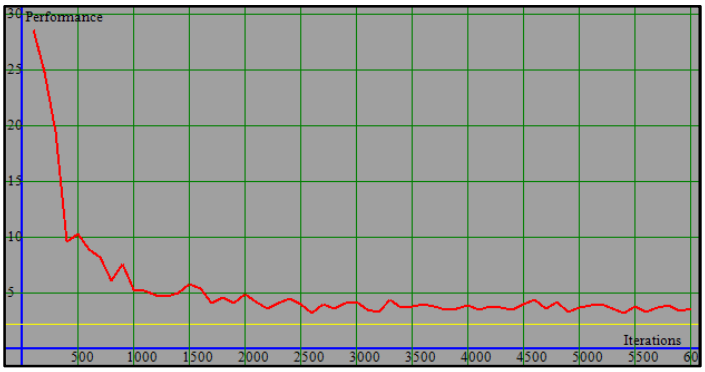
7.3.2 Ponašanje XCS-a u ne-Markovljevim okolinama

Kao reprezentativnu ne-Markovljevu okolinu uzima se okolina *maze7* prikazana na slici (5.7).



Slika 7.26 Ponašanje XCS-a u maze7 okolini (100 korak)

Rezultati mjerenja su prikazani na slici (7.26). Prilikom mjerenja se koristila kombinacija prvog načina mjerenja i *explore-exploit* strategije. Korak mjerenja je 100.
Iz slike (7.26) se vidi da se implementirani XCS jako loše ponaša u ne-Markovljevim okolinama. Kako XCS teži što preciznijem preslikavanju skupa stanja u skup akcija, a u *maze7* okolini postoje dvije pozicije za koje bezmemorijski sustavi ne mogu predvidjeti nagradu okolina, dobiveni rezultati nisu neočekivani. Izlaz iz ovoga problema se može pronaći u ugradnji interne memorije.

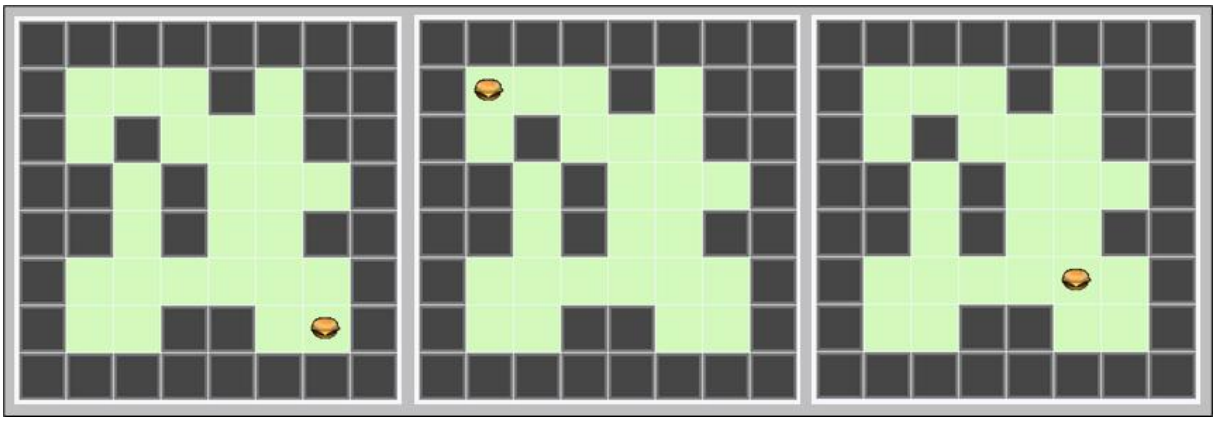


Slika 7.27 Ponašanje XCS-a u woods7 okolini (100 korak)

U okolini *woods7* XCS pokazuje znatno bolje rezultate nego u okolini *maze7*. Kao i ZCS, XCS uspije *shvatiti* da se pored kamenja nalazi hrana, a broj koraka do hrane je bliži optimumu od 3.1 koraka koji se bezmemorijskim sustavom može postići . U nastavku se XCS ispituje isključivo na Markovljevim okolinama.

7.3.3 Prilagodba XCS-a na promjenu položaja hrane

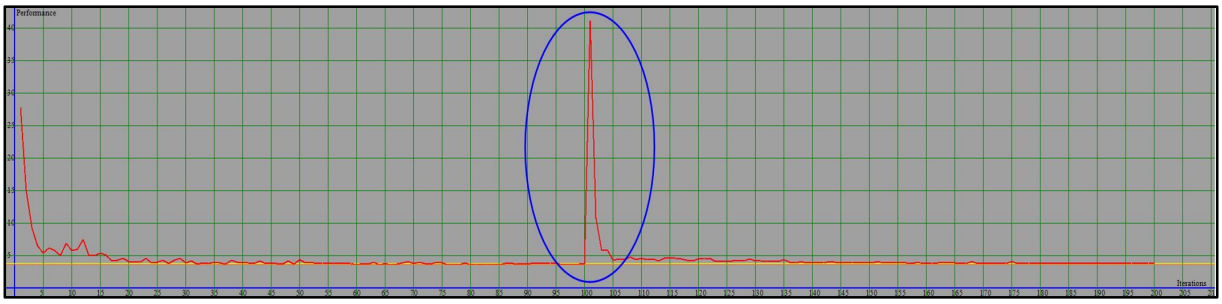
Kao što je rečeno kod mjerenja prilagodbe ZCS-a, razlikujemo dva bitna slučaja kod promjene položaja hrane: kada je nova pozicija hrane daleko od prvotne pozicije i kada je nova pozicija hrane blizu prvotnoj poziciji hrane. Ispitivanje prilagodbe se obavlja na okolini



Slika 7.28 Okolina maze288 s različitim pozicijama hrane

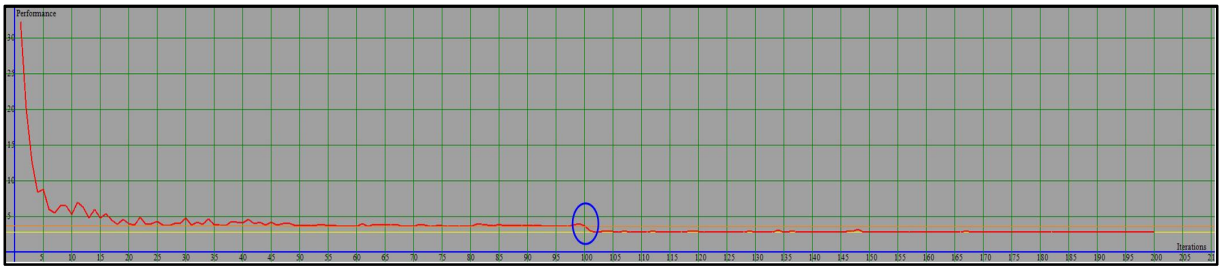
maze288 koja je prikazana na slici (7.28). Prvotni položaj hrane je u donjem desnom kutu, a hrana se još smješta u gornji lijevi kut i na poziciju do prvotne pozicije. Kada se hrana nalazi

u prvotnom položaju optimalan prosječan broj koraka do hrane je 3.63. Kada se hrana smjesti na udaljeniji položaj, optimalan prosječan broj koraka do hrane naraste na 3.79, a kada se smjesti na položaj do prvotnog položaja, optimalan prosječan broj koraka do hrane iznosi 2.75.
Provedena mjerenja su prikazana na slikama (7.29) i (7.30). Korak mjerenja je 1. Rezultati



Slika 7.29 Ponašanje XCS-a na veliku promjenu položaja hrane (1 korak)

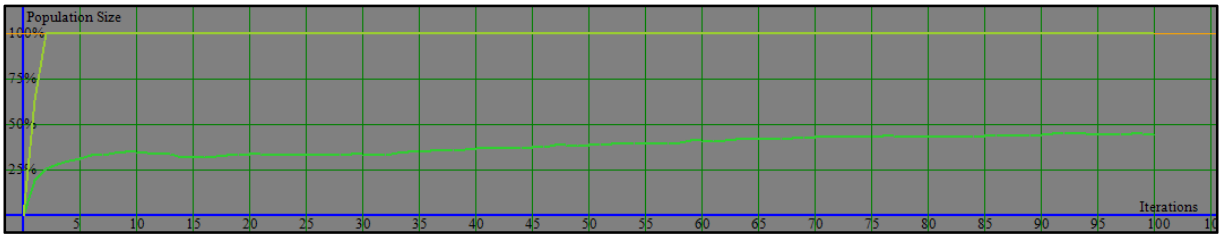
mjerenja su jednaki onima kod ZCS-a: kada je promjena položaja hrane velika pojavljuje se visoki šiljak u grafu; kada je promjena položaja hrane mala šiljka nema (ili je mali). Kako zaključci napravljeni za prilagodbu ZCS-a na promjenu položaja hrane vrijede i za XCS sustav, ovdje se ti zaključci neće posebno iznositi.



Slika 7.30 Ponašanje XCS-a na malu promjenu položaja hrane (1 korak)

7.3.4 Kretanje populacije XCS-a

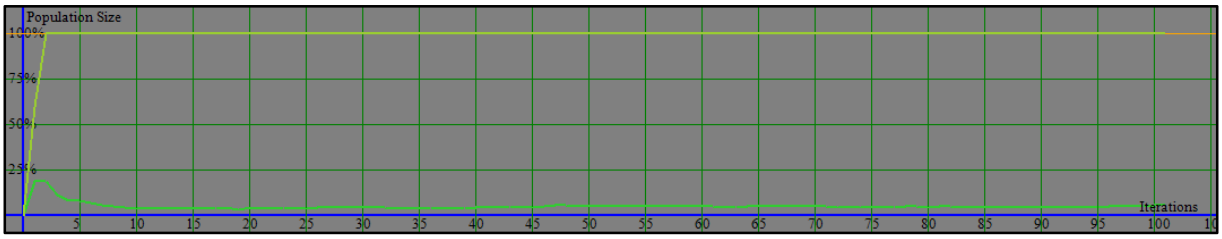
Kretanje populacije XCS sustava se pokazuje u ovisnosti o dva slučaja: koriste li se metoda gradijentnog spusta, metoda *GA Subsumption* i tehnika *deletion vote* ili se ne koriste. Na slici



Slika 7.31a Kretanje populacije XCS-a u maze6 okolini (sa deletion vote-om) (1 korak)

(7.31a) je prikazano kretanje populacije XCS-a u *maze6* okolini uz korištenje navedenih heuristika. Iako je XCS korištenjem navedenih heuristika došao do optimuma (pogledati

podpoglavlje 7.3.1), kretanje njegove populacije nije u skladu s očekivanjem. Naime, za očekivati je da će populacija makro-pravila rasti, a zatim postupno padati tako da se na kraju populacija sastoji od malog broja makro-pravila. Uzrok se može pronaći u *deletion vote* tehnici koja *čuva* pravila sa manjim iskustvom (jer se dobrota pravila uzima u obzir samo ako je pravilo dovoljno iskusno). Na slici (7.31b) je prikazano kretanje populacije XCS-a u *maze6* okolini bez korištenje navedenih heuristika. Kretanje populacije u tom slučaju je znatno bolje, ali ponašanje sustava unutar okoline nije dovoljno dobro (pogledati podpoglavlje 7.3.1).

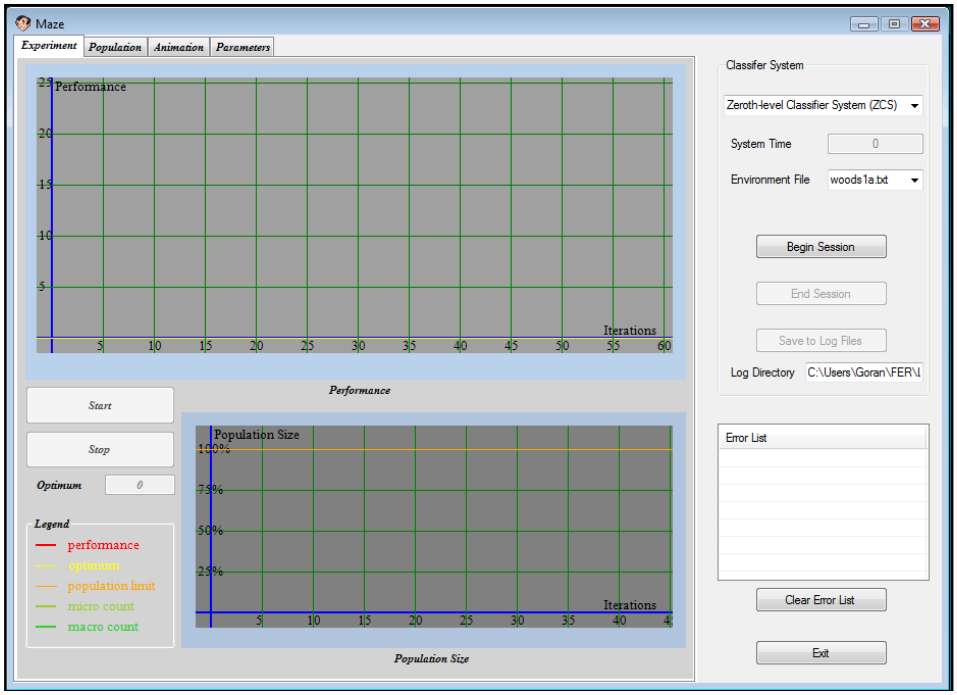


Slika 7.31b Kretanje populacije XCS-a u maze6 okolini (bez deletion vote-a) (1 korak)

Veći broj makro-pravila u populaciji omogućava potpunije preslikavanje (a time i kvalitetnije ponašanje sustava), ali je sustav znatno usporen (jer je dulja pretraga po populaciji makro-pravila). Kako postoji mogućnost da se potkrala pogreška pri implementaciji XCS-a, ovdje se ne mogu dati potpuni zaključci o vezi između *deletion vote* tehnike i kretanja populacije prikazanog na slici (7.31a).

8. Opis ostvarene aplikacije

U ovom poglavlju se ukratko opisuje programski proizvod ostvaren u sklopu završnog rada. Nakon pokretanja aplikacije otvara se prozor prikazan na slici (8.1). Prozor aplikacije je



Slika 8.1 Prozor nakon pokretanja aplikacije

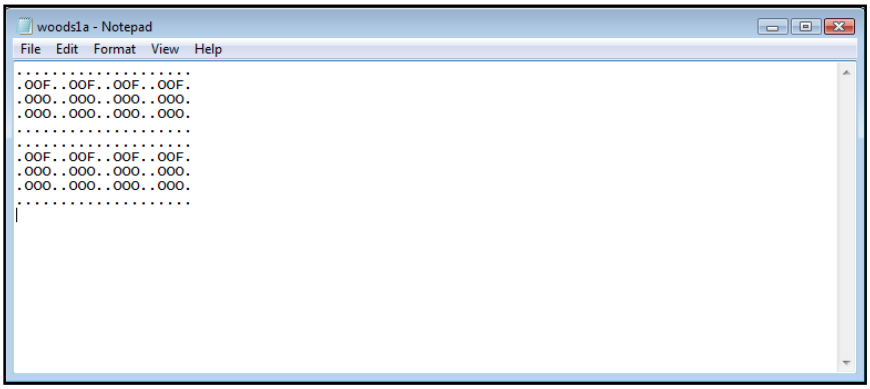
podijeljen na dva dijela: desni dio koji određuje koji se sustav koristi i koja se okolina labirinta učitava, i lijevi dio u kojem se nalaze kartice (*tabs*) vezane uz eksperiment i animaciju.

8.1 Učitavanje okoline i sustava

Na desnoj strani aplikacije se nalaze dva padajuća izbornika. U prvome (gornjem) se nalaze sustavi koji se mogu izabrati; može se izabrati samo jedan od dva ponuđena sustava. Ponuđeni sustavi su ZCS i XCS. Drugi padajući izbornik određuje okolinu koja se želi učitati. Osim ponuđenih okolina, može se učitati i vlastita stvorena okolina. Pravilno će se učitati samo okolina stvorena u tekstualnom formatu i koja zadovoljava sljedeće uvjete:

- prazno mjesto je kodirano znakom . ;
- hrana je kodirana znakom F ;
- prepreka (kamen) je predstavljen ostalim znakovima;
- sustav stvara uvijek pravokutnu okolinu (određenu najduljim retkom u datoteci) pa je svakako dobro odmah napraviti okolinu pravokutnih dimenzija;
- hrana mora postojati i sa svake se pozicije mora moći doći do hrane.

Primjerice okolina prikazana na slici (5.2) u tekstualnom formatu izgleda kao na slici (8.2).

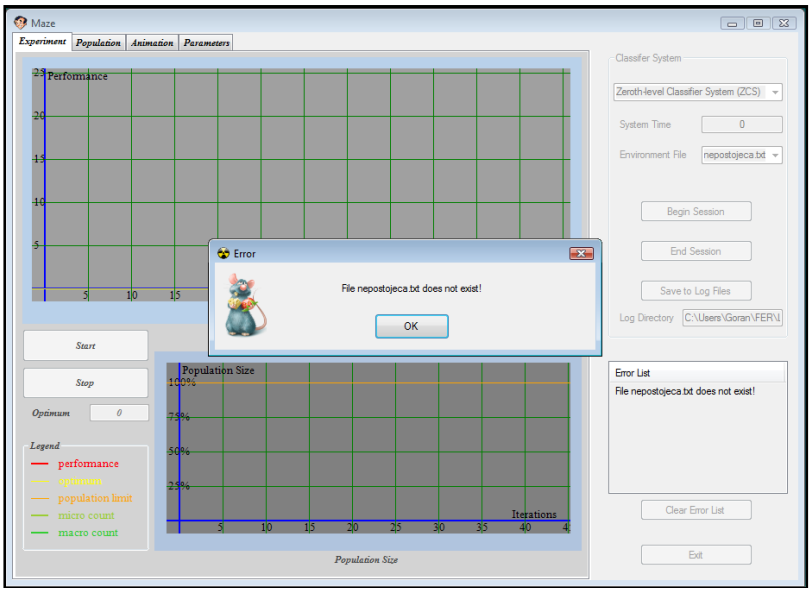


Slika 8.2 Okolina woods1 u tekstualnom formatu

Nakon što se pravilno izgradi okolina, u padajući izbornik se upiše put do datoteke. Učitavanje okoline i sustava se napravi klikom na *Begin Session*. Ako se želi učitati ponuđeni sustav sa željenim parametrima, potrebno je prije klika unijeti parametre. Za unos parametara pogledati podpoglavlje *Unos parametara*.

8.2 Dojava pogreške

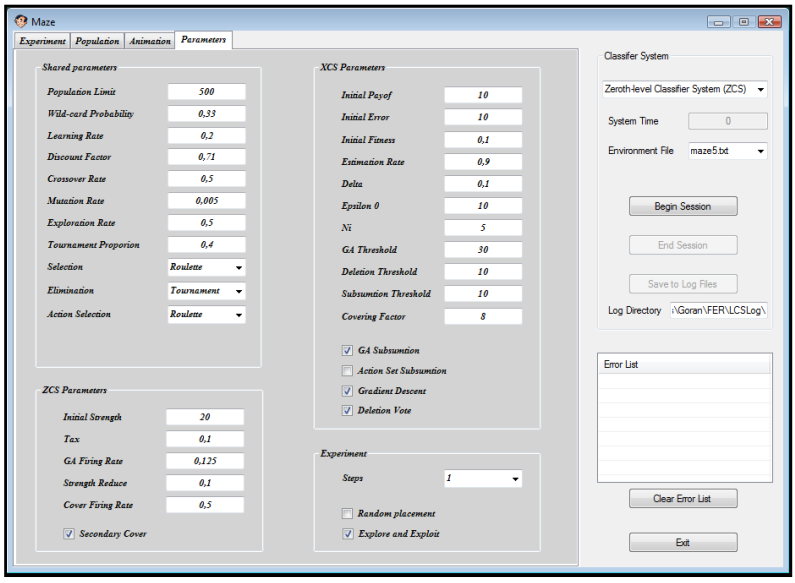
Aplikacija dojavljuje korisniku pogreške koje nastaju kao pogreške unosa. U prvom redu se to odnosi na nepostojeći sustav, krivo zadanu datoteku (ili put do datoteke) okoline, krivi unos parametara i krivi unos puta direktorija u koji se vrši spremanje slika, opisa populacije i rezultata eksperimenta (vidjeti podpoglavlje spremanje slika, opisa populacije i rezultata eksperimenta). Pogreške se dojavljuju u novoj formi kako je to prikazano na slici (8.3). U formu se stavlja opis pogreške, a opis pogreške moguće je pronaći i u listi pogrešaka (*error list*). Lista pogrešaka sadrži opise svih pogrešaka od kada je zadnji put ispražnjena. Lista pogrešaka se prazni klikom na *Clear Error List*.



Slika 8.3 Dojava pogreške

8.3 Postavljanje parametara

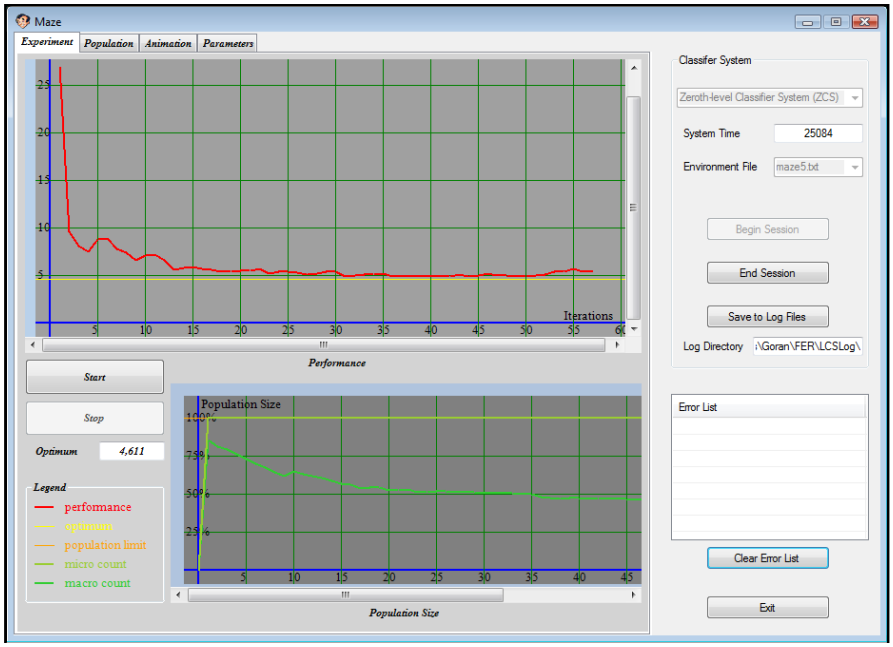
Lijevi dio aplikacije se sastoji od četiri kartice (*tabs*): eksperimenta, populacije, animacije i parametara. Izborom kartice *Parameters* se mogu postaviti parametri željenog sustava. Parametri se postavljaju prije nego što se učita sustav i okolina.



Slika 8.4 Unos parametara

8.4 Pokretanje eksperimenta

Nakon što se učita sustav i okolina, može se pokrenuti eksperiment. Eksperiment se nalazi u

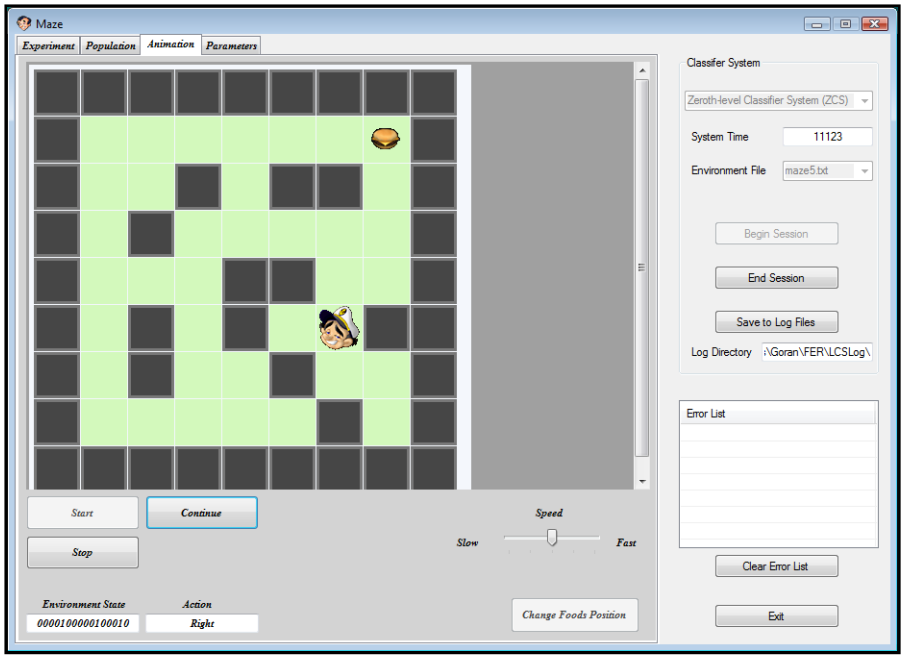


Slika 8.5 Pokretanje eksperimenta

kartici *Experiment*, a pokreće se klikom na *Start* i zaustavlja klikom na *Stop* (ili *End Session*). Za vrijeme trajanja eksperimenta, prikazuju se performanse sustava (prosječan broj koraka do hrane, gornji graf) i veličina populacije (donji graf). Prikazan je i optimalan prosječni broj koraka do hrane.

8.5 Pokretanje animacije

Animacija se pokreće na sličan načina kao i eksperiment. Nakon što se učita sustav i okolina može se pokrenuti animacija, pri čemu se animacija i eksperiment ne mogu istodobno pokrenuti. Animacija se nalazi kartici *Animation*, a pokreće se klikom na *Start* i zaustavlja klikom na *Stop* (ili *End Session*). Klikom na *Pause* animacija se pauzira, dok se klikom na *Continue* animacija nastavlja. Animacija prikazuje kretanje *animata* po okolini labirinta. Kada je kliknut *Pause* mogu se u kartici *Population* iščitati skupovi pravila (bez da se animacija prekine), a omogućeno je i spremanje slika, opisa populacije i rezultata eksperimenta. U kartici *Animation* se prikazuje i binarno kodirano stanje te zadnja izvršena akcija, a omogućeno je i mijenjanje pozicije hrane klikom na *Change Foods Position* (hrana se smjesti na nasumično odabranu poziciju). Brzina animacije se može postaviti na pet različitih vrijednosti.

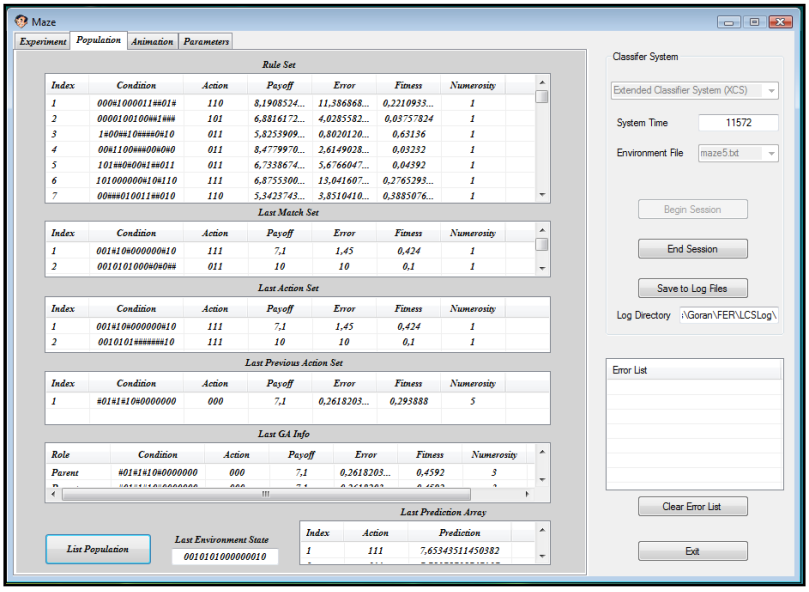


Slika 8.6 Pokretanje animacije

8.6 Prikaz populacije

Populacija se izlistava klikom na *List Population* u kartici *Population*. Populacija se ne može izlistati dok se odvija eksperiment ili animacija. Klikom na *List Population* se dobije prikaz skupa pravila (populacije), zadnjeg podudarnog skupa, zadnjeg akcijskog skupa, zadnjeg prethodnog akcijskog skupa, prikaz zadnjeg pokretanja GA-a i prikaz zadnjeg polja predviđanja ukoliko se trenutno ispituje XCS sustav. Za svaki navedeni skup je moguće

obaviti sortiranje po bilo kojem stupcu prikaza (ukoliko sortiranje po zadanom stupcu ima smisla) i to uzlazno i silazno. U kartici *Population* se nalazi i stanje okoline koje je detektirao sustav u svom zadnjem punom ciklusu. Izlistavanje populacije je prikazano na slici (8.7).



Slika 8.7 Izlistavanje populacije

8.7 Spremanje slika, opisa populacije i rezultata eksperimenta

Spremanje slika, opisa populacije i rezultata omogućeno je nakon klika na *Begin Session*. Slike, opis populacije i rezultati eksperimenta ne mogu se spremati tijekom izvođenja eksperimenta ili animacije. Spremanje slika, opisa populacije i rezultata eksperimenta se vrši klikom na *Save to Log Files* pri čemu zadani direktorij *Log Directory* mora postojati (inače se javlja greška). Slike koje se spremaju su slike grafova u eksperimentu i slika (zadnje prikazane) okoline. Opis populacije (opis svih skupova pravila) i rezultati eksperimenta se smještaju u tekstualne datoteke. Rezultati eksperimenta se spremaju u obliku: [broj koraka do hrane] [broj makro-pravila] [broj mikro-pravila]. Imena datoteka koje se koriste za spremanje slika, opisa populacije i rezultata mjerenja su: *CFSPerformance.bmp*, *PopulationSize.bmp*, *Environment.bmp*, *Population.txt*, *Results.txt*. Ukoliko navedene datoteke postoje, naziv se modificira tako da se dodaje broj prije ekstenzija .bmp i .txt. Broj koji se dodaje jednak je najmanjem prirodnom broju za koji modificirano ime datoteke ne postoji u zadanom direktoriju.

8.8 Mijenjanje sustava i okoline i izlaz iz aplikacije

Vraćanje u stanje prije učitavanja sustava i okoline se postiže klikom na *End Session*. Nakon toga se mogu promijeniti okolina, sustav i parametri te ponovno započeti ispitivanje sustava.
Izlaz iz aplikacije se postiže klikom na *Exit* ili na križić u gornjem desnom kutu prozora aplikacije.

9. Ukratko o primjeni LCS-a

LCS se uglavnom upotrebljava za rješavanje jednog od ova četiri tipa problema: klasifikacijski problemi (*classification problems*), problemi učenja potkrjepljenjem (*reinforcement learning problems*), problemi aproksimacije funkcija (*function approximation problems*), problemi predviđanja (*prediction problems*) [11].
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 *dubinske analiza podataka* (*data mining*).
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, woods problem*).
Cilj LCS-a koji se primjenjuje na problemima aproksimacije funkcije je što točnija aproksimacija funkcije skupom pravila koja se djelomično preklapaju. Primjer takvog problema je problem aproksimacije 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 svakako treba spomenuti: *dubinsku analizu podataka* (*data mining*), *upravljanje procesima* i *modeliranje i optimizaciju* [3]. U tablici (9.1) su za spomenuta područja dani primjeri aplikacija koje upotrebljavaju LCS.

Tablica 9.1 Područja primjene LCS-a

Područja primjene LCS-a	**Primjeri aplikacija**
Dubinska analiza 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

10. Zaključak

Izvedena mjerenja na različitim okolinama labirinta pokazuju da su prošireni modeli ZCS i XCS klasifikatorskih sustava pogodni za rješavanje problema učenja potkrjepljenjem (podražajem). Prvenstveno se treba obratiti pažnja na *multi step* probleme s odgovarajućim Markovljevim okolinama u kojima dobra svojstva ZCS-a i XCS-a izlaze na vidjelo. Mogućnost adaptacije na promjene u okolini dodatno je svojstvo koje krasi LCS sustave, a lošije ponašanje u ne-Markovljevim okolinama može se značajno popraviti uvođenjem interne memorije. Uvođenje metode gradijentnog spusta te korištenje turnirske selekcije značajno poboljšavaju performanse sustava u *multi step* problemima, a modeli ZCS-a i XCS-a su otvoreni za dodatne heuristike koje bi mogle ubrzati proces učenja i preciznost klasifikacije. Ipak, treba pripaziti da se heuristikama uvodi što manje parametara jer opisani LCS sustavi već sadrže velik broj parametara čije vrijednosti variraju ovisno o problemu koji se njima rješava. Upravo veliki broj parametara predstavlja značajan nedostatak LCS sustava. Premda iza metoda kojima se koriste navedeni sustavi stoje već dobro poznati eksperimentalni rezultati, potpuni matematički opis ponašanja ZCS i XCS sustava još uvijek nije poznat.

Iako su zasjenjeni mnogo poznatijim genetskim algoritmima, sve je veći interes istraživanja u području klasifikatorskih sustava s mogućnošću učenja. Razlog tome su dobra svojstva XCS-a, kao i dobre karakteristike LCS sustava zasnovanih na očekivanju (posebice se to odnosi na MACS sustav). Ne treba izostaviti ni LCS sustave pittsburškog stila i stila iterativnog učenja za koje također vlada interes u istraživačkim krugovima.

Premda LCS predstavlja opširno područje koje se primjenjuje i u rješavanju stvarnih problema, to je područje još uvijek nedovoljno istraženo. Posebna pažnja se usmjerava na istraživanje novih te analizu i poboljšavanje postojećih LCS sustava. Buduća istraživanja uključuju identifikaciju karakteristika koje određuju može li se problem riješiti LCS-om i određivanje koji modeli LCS-a su najprikladniji za rješavanje određenih problema. Također, uz poboljšavanje postojećih i stvaranja boljih modela LCS-a, potrebno je poboljšati i teorijsku podlogu LCS-a koja je općenito nedovoljno istražena u području evolucijskog računanja.

Literatura

[1] Eiben A. E., Smith J. E., *Introduction to Evolutionary 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/lcsg/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*, *Latent learning*, *Mersenne Twister*, dostupno na: http://en.wikipedia.org/wiki/Main_Page , 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.V., *Rule-Based Evolutionary Online Learning Systems: A Principled Approach to LCS Analysis and Design*, Berlin: Springer-Verlang, 2006.
[19] Metivier M., Lattaut C., *Anticipatory Classifier System using Behavioral Sequences in Non-Markov Environment*, IWLCS 2002 : international workshop on learning classifier systems No5, Granada, rujan 2002.
[20] Kusiak A., *Anticipatory Learning Classifier Systems: An Introduction*, dostupno na: http://www.icaen.uiowa.edu/~ie238/Lecture/LCS_2.pdf , studeni 2007.
[21] Butz M., Stolzmann W. , *An Algorithmic Description of ACS2*, dostupno na: http://74.125.39.104/search?q=cache:a4roRMdc7zwJ:ftp://ftp-illigal.ge.uiuc.edu/pub/papers/Publications/2002/iwlcs01-butz2.ps.Z+Butz+ACS2+An+algorithmic+description+of+ACS2&hl=hr&ct=clnk&cd=9&gl=hr , lipanj 2008.
[22] Bull L., Kovacs T., *Foundations of Learning Classifier Systems*, 6. izdanje, Berlin: Springer-Verlang, 2005.
[23] Lanzi P.L., Wilson S.W., *Optimal Classifier System Performance in Non Markov Environments*, IlliGAL Report No. 99022, prosinac 1999.
[24] Butz M.V., Wilson S.W., *An Algorithmic Description of XCS*, IlliGAL Report No. 2000017, travanj 2004.
[25] Lučić M., Radanović G., *Evolucijski algoritmi: Klasifikatoski sustavi*, projekt, Fakultet elektrotehnike i računarstva, Zagreb, siječanj 2008.

Dodatak A: Oznake i parametri

Tablica A.1 Korištene oznake

Oznaka	Objašnjenje
CS, CFS	Klasifikatorski sustav (*Classifier System*)
LCS	Klasifikatoski sustav s mogućnošću učenja (*Learning Classifier System*)
n-LCS	Klasifikatorski sustav bez mogućnosti učenja (*non-Learning Classifier System*)
ZCS	*Zeroth-level Classifier System*
XCS	*Extended Classifier System*
ACS	*Anticipatory Classifier System*
ACS2	*Anticipatory Classifier System 2*
YACS	*Yet Another Classifier System*
MACS	*Modular Anticipatory Classifier System*
RL	Učenje potkrjepljenjem (podražajem) (*Reinforcement Learning*)
SL	Učenje pod nadzorom (*Supervised Learning*)
GA	Genetski algoritam (*Genetic Algorithm*)
[N]	Populacija pravila (*Population, Rule Set*)
[M]	Podudarni skup (*Match Set*)
[A]	Akcijski skup (*Action Set*)
$[A]_{-1}$	Prethodni akcijski skup (*Previous Action Set*)
PA	Polje predviđanja (*Prediction Array*)
ALP	Proces učenja na očekivanju (*Anticipatory Learning Process*)

Tablica A.2 Tipične oznake i parametri koji se pojavljuju u LCS sustavima

Oznaka	Objašnjenje
ρ	Nagrada okoline (*Reward*)
σ	Stanje okoline (*State*)
N	Veličina populacije (*Population Size*), populacijski limit (*Population Limit*)
$p_{\#}$	Vjerojatnost pojave *don't care* simbola (*Wild-Card Probability*)
β	Brzina učenja (*Learning Rate*)
γ	Koeficijent smanjivanja nagrade (*Discount Factor*)
λ, P_c	Vjerojatnost križanja (*Crossover Rate*)
μ, P_m	Vjerojatnost mutacije (*Mutation Rate*)
ε	Parametar ε *-Greedy strategije* (*Exploration Rate*)
tour	Proporcija turnira u turnirskoj selekciji (*Tournament Proportion*)
τ	Udio snage koje se uzima pravilima iz skupa [M]\[A] u ZCS-u (*Tax*)
Φ	Vjerojatnost pokretanja *cover* algoritma kod ZCS-a (*Cover Firing Rate*)
P_{ga}	Vjerojatnost pokretanja GA u ZCS sustavu (*GA Firing Rate*)
S_I	Vrijednost snage pri stvaranju ZCS pravila (*Initial Strength*)
S_r	Faktor kojim se množe snage djece u ZCS-u (*Strength Reduce Factor*)

p_I, ε_I, F_I	Početne vrijednosti parametara predviđanja, pogreške i dobrote XCS pravila
α, ν, ε_0	Vrijednosti koeficijenata koji se koriste pri evaluaciji XCS pravila
κ	Preciznost XCS pravila
κ_r	Relativna preciznost XCS pravila
Θ_{GA}	Granica koja kontrolira pokretanje GA-a (*GA Threshold*)
Θ_{sub}	Minimalno iskustvo XCS pravila potrebno da bi pravilo moglo obuhvaćati manje općenita pravila (*Subsumption Threshold*)
Θ_{del}	Minimalno iskustvo XCS pravila potrebno da se prilikom eliminacije uzme u obzir njegova dobrota (*Deletion Threshold*)
δ	Koeficijent koji određuje koliko mala treba biti dobrota XCS pravila u odnosu na prosječnu dobrotu svih pravila da se prilikom eliminacije uzme u obzir njegova dobrota

Tablica A.3 Parametri pravila koji se pojavljuju u opisanim LCS sustavima

Parametar	**Objašnjenje**
C	Uvjet (*Condition*)
A	Akcija (*Action*)
N	Brojnost makro-pravila (*Numerosity*)
s	Snaga ZCS pravila (*Strength*)
p	Predviđanje nagrade (*Predicted Payoff*) XCS pravila
ε	Procjena pogreške predviđanja (*Estimated Error*) XCS pravila
F	Dobrota XCS pravila (*Fitness*)
E	Očekivanje, efekt (*Effect*) sustava zasnovanih na očekivanju
M	Parametar oznake (*Mark*) kod ACS2 pravila
q	Kvaliteta (*Quality*) ACS2 pravila
r	Mjera nagrade (predviđanje nagrade, *Reward Measure*) ACS2 pravila
Dodatni parametri za XCS pravilo	
exp	Iskustvo XCS pravila (*Experience*)
as	Srednja vrijednost veličine akcijskih skupova kojima je XCS pravilo pripadalo (*Action Set Size*)
ts	Vremenska oznaka (*Time Stamp*), određuje vrijeme zadnjeg sudjelovanja XCS pravila u GA-u

Dodatak B: Vrijednosti parametara korištene pri mjerenju

U ovom dodatku se daje tablični prikaz vrijednosti parametara ZCS-a i XCS-a korištene prilikom ispitivanja sustava na problemu labirinta. U svakom provedenom mjerenju su dane vrijednosti onih parametara koji imaju različitu vrijednost od one navedene u tablici.

Tablica B.1 Vrijednosti parametara korištene prilikom mjerenja

Parametar sustava	**ZCS**	**XCS**
ρ *(konst.)*	*1000*	*1000*
N	*500*	*3000*
$p_{\#}$	*0.33*	*0.33*
β	*0.2*	*0.2*
γ	*0.71*	*0.71*
λ, P_c	*0.5*	*0.5*
μ, P_m	*0.005*	*0.005*
ε	*0.5*	*0.01*
tour	*0.4*	*0.4*
Selekcija pravila u GA-u	*Jednostavna (Roulette)*	*Turnirska*
Eliminacija pravila	*Turnirska*	*Jednostavna (Roulette)*
Selekcija akcije	*Jednostavna (Roulette)*	ε*-Greedy*
τ	0.1	-
Φ	0.5	-
P_{ga}	0.125	-
S_I	20.0	-
S_r	0.1	-
Secondary Cover	Koristi se	-
p_I	-	10
ε_I	-	10
F_I	-	0.01
α	-	0.1
ν	-	5
ε_0	-	10
Θ_{GA}	-	25
Θ_{sub}	-	10
Θ_{del}	-	10
δ	-	0.1
GA subsumption	-	Koristi se
Action Set Subsumption	-	Ne koristi se
Gradient Descent	-	Koristi se
Deletion Vote	-	Koristi se