SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1357

Ugradnja klasifikatorskih sustava u okruženje za evolucijsko računanje

Danko Komlen

Zagreb, lipanj 2010.

Sadržaj

1. Uvod

Mnogi problemi sa kojima se susrećemo u računarstvu zahtijevaju specijalizirano znanje iz neke problemske domene. Njihovim rješavanjem bave se ekspertni sustavi (eng. *expert system*) kao posebna grana umjetne inteligencije. To su sustavi temeljeni na pravilima kod kojih su u posao razvoja baze znanja uključeni istovremeno poznavatelji domene te inženjeri baze znanja. Tako se glavni uvjet za razvoj uspješnog sustava sastoji od učinkovitog prikupljanja i integracije znanja. Rješavanju zahtjevnog problema prikupljanja znanja moguće je pristupiti pomoću klasifikatorskih sustava s mogućnošću učenja (eng. *learning classifier system*, LCS), koje je 1976. predstavio John Holland. Kod njih se pravila razvijaju kombinacijom tehnike podržanog učenja (eng. *reinforcement learning*, RL) te tehnikama za evolucijsko računanje (eng. *evolutionary computation*). LCS sustav kroz iterativnu komunikaciju sa okolinom koja predstavlja određeni problem pokušava maksimizirati dobivenu nagradu i time ponuditi što bolje rješenje. Jedna od popularnijih vrsta LCS-a je klasifikatorski sustav XCS, koji je u radu detaljanije razmatran.

XCS sustav implementiran je kao vrsta alogoritma u sklopu okruženja za evolucijsko računanje (eng. *evolutionary computation framework*, ECF). Pomoću dobivene komponente izgrađena je simulacijska okolina za rješavanje višekriterijskog problema snalaženja u 2D labirintu.

U prvom poglavlju dan je pregled Q-učenja te genetičkih algoritama. Njihovo poznavanje potrebno je za razumijevanje rada LCS-a čije su osnovne vrste i način rada opisani u trećem poglavlju, dok je u četvrtom poglavlju detaljnije obrađen XCS sustav. U drugom poglavlju dan je opis vrsta problema koji se mogu rješavati klasifikatorskim sustavima, sa naglaskom na višekoračne probleme podržanog učenja čiji je primjer obrađen u implementaciji. U petom poglavlju opisan je način predstavljanja višezadaćnih problema za rješavanje XCS-om, a u šestom njegova programska implementacija kroz simulacijsku okolinu te opis XCS komponente u sklopu ECF-a. U posljednjem poglavlju dani su eksperimentalni rezultati i njihova usporedba sa postojećim klasifikatorskim sustavom.

2. Podržano učenje i genetički algoritam

Za proučavanje klasifikatorskih sustava s mogućnošću učenja potrebno je poznavanje algoritama podržanog učenja te određene heurističke metode pretraživanja prostora rješenja. U sljedeća dva potpoglavlja dano je uvodno razmatranje algoritma Q-učenja (eng. *Q-learning*), te genetičkih algoritama (eng. *genetic algorithm*, GA) kao tehnike evolucijskog računanja.

2.1 Podržano učenje

Podržano učenje je tehnika strojnog učenja (eng. *machine learning*) kojom se rješavaju problemi modelirani Markovljevim procesima (eng. *Markov decision proces*, MDP). Kod njih je virtualni agent u interakciji sa okolinom koja predstavlja određeni problem, a interakcija je vođena MDP procesom. Više detalja o problemima podržanog učenja dano je u zasebnom poglavlju 3.2. Osnovni elementi svakog deterministčkog[1] MDP procesa su sljedeći [10]:

- konačan skup stanja S;
- konačan skup akcija A;
- funkcija prijelaza p: $S \times A \rightarrow S$;
- funkcija nagrade r: $S \times A \rightarrow \mathbb{R}$, koja za par (s,a) daje skalarnu vrijednost nagrade koju agent dobiva ako u stanju *s* izvrši akciju *a*.

Opisani formalizam ne govori ništa o ponašanju agenta već samo o sljedećim stanjima i nagradama koje će dobiti ovisno o odabranom nizu akcija. Ponašanje agenta definirano je *strategijom ponašanja* (eng. *behavioral policy*) $\pi : S \rightarrow A$. Cilj je pronaći optimalnu strategiju π^*, koja će rezultirati u maksimalnoj ukupno dobivenoj nagradi, a da pri tome nisu poznate funkcije *p* i *r*. Iz tog razloga definiramo *funkciju ukupne nagrade* (eng. *cumulative discounted reward*):

$$V^{\pi}(s_t) = \sum_{i=0}^{\infty} \gamma^i r(s_{i+t}, a_{i+t}), \tag{2.1}$$

koja predstavlja zbroj budućih nagrada od trenutka *t*, pomnoženih sa *faktorom odbitka* (eng. *discount factor*) $\gamma \in [0,1]$.

[1] Uz deterministčke MDP procese postoje i nedeterministički kod kojih je prijelaz u sljedeće stanje definiran određenom vjerojatnošću

Stanja s_{i+t} su buduća stanja okoline u kojima agent obavlja akcije a_{i+t}, prema strategiji ponašanja π. Faktor odbitka služi kako bi se prednost dala nagradama koje se dobiju ranije. Pomoću funkcije ukupne nagrade možemo definirati željenu optimalnu strategiju ponašanja:

$$\pi^* \equiv \arg\max_{\pi} V^{\pi}(s_t) . \tag{2.2}$$

Korištenjem strategije π^* u određenom stanju *s*, optimalnu akciju možemo dobiti pomoću sljedećeg izraza:

$$\pi^*(s) = \arg\max_{a}(r(s,a) + \gamma V^{\pi^*}(p(s,a))) . \tag{2.3}$$

2.2 Q-učenje

Prema relaciji 2.3 za odabir optimalne akcije potrebno je doznati optimalnu funkciju ukupne nagrade V^{π^*}. Međutim za nju je potrebno poznavanje funkcije prijelaza p. Rješenje je moguće korištenjem Q-učenja kod kojeg se definira Q-vrijednost:

$$Q(s,a) = r(s,a) + \gamma V^{\pi^*}(p(s,a)) , \tag{2.4}$$

koja određuje ukupnu buduću nagradu nakon što se u stanju *s* izvrši akcija *a*. Stoga izraz za V^{π^*} možemo zapisati kao:

$$V^{\pi^*}(s) = \max_{a} Q(s,a) , \tag{2.5}$$

što omogućuje rekurzivnu definiciju Q-vrijednosti:

$$Q(s,a) = r(s,a) + \gamma \max_{a'} Q(p(s,a), a') . \tag{2.6}$$

Dobiveni izraz 2.6 poseban je slučaj sljedećeg općenitijeg načina osvježavanja Q-vrijednosti [1]:

$$Q(s,a) \leftarrow Q(s,a) + \beta(r(s,a) + \gamma \max_{a'} Q(p(s,a),a') - Q(s,a)) . \tag{2.7}$$

Parametar β ϵ [0,1] je *stopa učenja* (eng. *learning rate*), koji označava kolika se važnost daje novoj informaciji u odnosu na staru. Za veću vrijednost stope učenja prednost se daje novom znanju, tako se za krajnji slučaj β=1 dobiva relacija 2.6 gdje se prethodna Q-vrijednost ne uzima u obzir prilikom izračuna.

Izbor optimalne akcije u stanju *s* korištenjem Q-vrijednosti dobiva se iz izraza 2.3 i 2.4:

$$\pi^*(s) = \arg\max_a (Q(s,a)). \tag{2.8}$$

Algoritam Q-učenja sastoji se od odabira optimalne akcije *a* u određenom stanju *s* prema 2.8 te njenim izvršavanjem. Zatim se uz dobiveno novo stanje s' = p(s,a) i nagradu r' = r(s,a) korištenjem relacije 2.7 osvježava Q-vrijednost Q(s,a). Koraci se ponavljaju iterativno i Q-vrijednosti sa kojima algoritam radi postepeno konvergiraju ispravnim vrijednostima [10].

2.3 Genetički algoritam

Genetički algoritam je heuristička metoda optimiranja koja imitira prirodni evolucijski proces, robustan proces pretraživanja prostora rješenja. Svrstavamo ga u stohastičke metode, kod kojih, za razliku od deterministčkih, ne možemo sa sigurnošću tvrditi koji će biti rezultat izvođenja.

Kod genetičkih algoritama za opis svojstava jedinke najčešće se koristi binarni zapis nad kojim se vrše operacije po uzoru na evoluciju iz prirode (selekcija, križanje i mutacija).

Princip rada genetičkog algoritma

Genetički algoritmi rade sa populacijom potencijalnih rješenja koje nazivamo *jedinkama*. Mjera kvalitete rješenja naziva se *dobrota* (eng. *fitness*), a funkcija kojom se ona određuje, *funkcija dobrote*.

Glavne faze procesa izvođenja GA nazivaju se *generacije*. Prilikom prijelaza iz jedne generacije u drugu, određenim postupkom izdvajaju se bolje jedinke iz populacije, od kojih se neke podvrgavaju genetičkim operatorima i time stvaraju nove jedinke.

Proces se zaustavlja ukoliko se postigne željena vrijednost funkcije dobrote ili se dosegne određen broj iteracija. Rješenje najbliže optimalnom predstavlja najbolja jedinka posljednje generacije.

Proces stvaranja nove generacije podijeljen je na sljedeće korake:

- izračunavanje funkcije dobrote za svaku jedinku;
- selekcija (odabir najboljih jedinki);
- reprodukcija (stvaranje novih jedinki križanjem i primjena mutacije).

Početna populacija može se dobiti nasumičnim odabirom rješenja ili korištenjem neke druge optimizacijske metode.

Postupci selekcije

Kao što je već ranije spomenuto, uloga selekcije je prenošenje dobrih svojstava na sljedeću generaciju. Genetičke algoritme prema vrsti selekcije dijelimo na generacijske i eliminacijske. U generacijske spadaju algoritmi s *jednostavnom selekcijom* (eng. *roulette wheel selection*). Veličina populacije kod jednostavne selekcije je konstantna, a vjerojatnost odabira jedinke (p_n) je proporcionalna njezinoj dobroti, pri čemu se p_n računa po formuli:

$$p_n = \frac{dobrota(n)}{\sum_i dobrota(i)}, \text{ gdje je } n \text{ oznaka jedinke.} \tag{2.9}$$

Uz jednostavnu u generacijske selekcije spada i *turnirska selekcija* (eng. *tournament selection*). Kod nje se s jednakom vjerojatnošću odabire *k* jedinki, od kojih se uzima najbolja i stavlja u bazen za reprodukciju. Postupak se nastavlja dok se ne odabere cijela populacija nad kojom se dalje djeluje genetičkim operatorima.

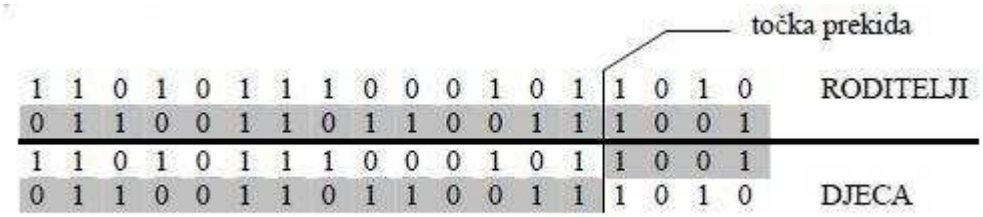
Kod eliminacijskih algoritama koristi se *eliminacijska selekcija* (eng. *steady-state selection*). Ona za razliku od jednostavne, bira loše jedinke koje će se eliminirati i zamijeniti novima.

Genetički operatori

Nakon postupka selekcije dobrih jedinki započinje postupak njihove reprodukcije, kako bi se nadomjestili izbačeni kromosomi i održala veličina populacije. Reprodukcija se ostvaruje pomoću genetičkog *operatora križanja* (eng. *crossover*).

Procesom križanja iz dvije jedinke roditelja stvara se jedna ili dvije nove jedinke djece, koje nasljeđuju svojstva roditelja. Kod binarnog zapisa vrši se zamjena bitova koji se nalaze između *točaka prekida* (slika 2.1). Bitovi koji će se mijenjati određuju se maskom *K*, koja je dobivena slučajnim odabirom bitova prema masci s vjerojatnostima. Sama operacija križanja nad kromosomima roditelja R1 i R2, te dobivanje djeteta D, vrši se prema sljedećoj formuli:

$$D = R1R2 + K(R1 \otimes R2). \tag{2.10}$$



Slika 2.1. Križanje s jednom točkom prekida [9]

Sljedeći operator karakterističan za genetičke algoritme je unarni *operator mutacije*. Za svaki bit u kromosomu određuje se vjerojatnost da dođe do njegove promjene. Kod jednostavne mutacije ta je vjerojatnost jednaka za sve bitove. Mutacija je odgovorna za slučajno pretraživanje prostora rješenja, kao i za izlazak iz lokalnog minimuma ukoliko cijela populacija u njemu završi

3. Problemi prikladni za LCS

Domena primjene strojnog učenja je vrlo široka i postoji više vrsta problema za koje su prikladne takve tehnike. Kod klasifikatorskih sustava najčešće se radi o interakciji sa okolinom koja predstavlja određeni problem, a LCS ima ulogu sustava za odlučivanje. U nastavku su dani opisi tri glavne vrste problema prikladnih za rješavanje klasifikatorskim sustavom, uz naglasak na višekoračne probleme podržanog učenja.

3.1 Vrste problema

Vrste problema koji se mogu rješavati sa klasifikatorskim sustavima su:

- optimizacijski,
- klasifikacijski i
- problemi podržanog učenja [5].

Kod optimizacijskih problema cilj je pronaći najbolje rješenje iz prostora svih mogućih rješenja. Sustav kao odgovor na dano rješenje dobiva ocjenu njegove kvalitete koju mu je u konačnici cilj maksimizirati. Klasifikacijski problemi odnose se na traženje koncepta koji svaki dani problem svrstava u određeni razred. Kod sustava za rješavanje takvih problema poželjna svojstva su *preciznost* i *generalizacija*. Preciznost se može mjeriti kao postotak uspješno klasificiranih problema dok generalizacija označava da je rješenje moguće primjeniti na probleme s kojima se sustav još nije susreo.

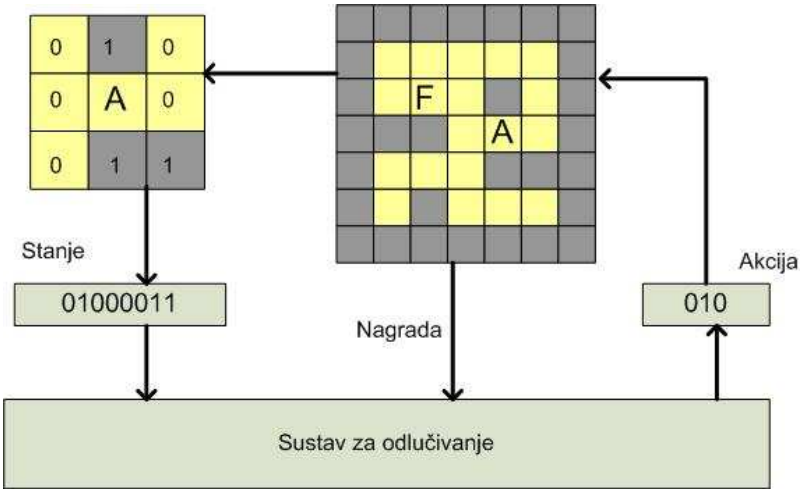
Treća vrsta problema su problemi podržanog učenja (eng. *reinforcement learning problems*) kod kojih je zadaća sustava pronaći optimalnu strategiju ponašanja u interakciji sa okolinom koja predstavlja određeni problem i služi kao izvor podataka za LCS.

3.2 Jednokoračni i višekoračni problemi

Problemi podržanog učenja specifični su po tome da se povratna informacija o kvaliteti rješenja dobiva sa odgodom. Također, stanje okoline uvjetovano je prethodno odabranom nizu akcija. Takvu vrstu problema kod kojih vrijede ovi uvjeti nazivamo još i *višekoračnim problemima* (eng. *multistep problem*). Višekoračni problemi modelirani su ranije opisanim Markovljevim procesima. Druga vrsta problema podržanog učenja su *jednokoračni* (eng. *singlestep problem*), kod kojih se povratna informacija od okoline dobiva bez odgode, a svako njezino stanje predstavlja problem za sebe.

Klasifikacijski problemi mogu se poistovjetiti s jednokoračnima, gdje okolina šalje nagradu ovisno o tome da li je zadani problem točno klasificiran. Primjer jednokoračnog problema je problem 6-multipleksora [14].

Nešto složenija podvrsta višekoračnih problema su oni modelirani djelomično vidljivim Markovljevim procesima (eng. *partially observable Markov decision proces*). Kod njih za razliku od MDP procesa, agent opaža okolinu pomoću funkcije opažanja koja preslikava stvarno stanje okoline u opažanje agenta [5].



Slika 3.1 Rad okoline koja predstavlja labirint

Primjer višekoračnog problema je problem snalaženja u 2D labirintu (slika 3.1). Okolina je labirint kroz koji se kreće agent (označen sa A) ovisno o akcijama sustava za odlučivanje. Takvu vrstu agenta nazivamo još i *animat* [3]. Cilj animata je u što manjem broju koraka pronaći cilj ("hrana" označena sa F), pri čemu se može kretati u osam mogućih smjerova.

Prije svake odluke o pomaku u određenom smjeru, sustav dobiva stanje labirinta u obliku binarnog zapisa osam susjednih polja oko animata. Zapisi polja u stanju određuju se ovisno o tome da li se na njima nalazi hrana, zid ili je polje prazno. Sljedeći korak je donošenje odluke o akciji, prema strategiji ponašanja koju sustav koristi. Ukoliko je moguće animat se pomiče, a zatim okolina šalje nagradu u skladu sa kvalitetom akcije.

4. LCS klasifikatorski sustavi

Klasifikatorski sustavi s mogućnošću učenja su vrsta sustava temeljenih na pravilima (eng. *rule-based system*) koji se koriste kao tehnika u području strojnog učenja. Preciznije, radi se o genetički temeljenim tehnikama strojnog učenja (eng. *genetic-based machine learning*, GBML) [8]. Pravila sustava nalaze se u uobičajenom AKO-ONDA produkcijskom obliku. Za pretraživanje prostora pravila odgovorna je komponenta evolucijskog računanja, čija najčešće korištena tehnika su genetički algoritmi. Vrednovanje pravila koje vodi proces učenja prepušteno je tehnici podržanog učenja. Za uspjeh LCS-a ključna je suradnja tih dviju komponenti. Tako se komponenta za podržano učenje oslanja na strukturu klasifikatora koju su razvili GA. S druge strane za uspješan rad GA komponente potrebno je točno vrednovanje klasifikatora koju obavlja RL komponenta u interakciji s okolinom. Osnovna struktura LCS sustava prikazana je na slici 4.1.

4.1 Rad klasifikatorskog sustava

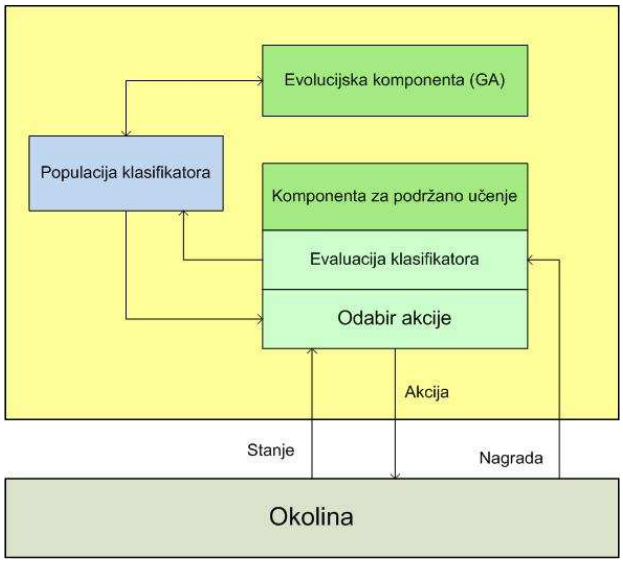
Rad LCS-a temelji se na populaciji klasifikatora koja predstavlja bazu znanja i okosnica je strategije ponašanja sustava. Svaki klasifikator (cl) sastoji se od uvjeta (C), akcije (A) i skupa parametara. Radi lakšeg opisivanja u nastavku se svaka komponenta koja se odnosi na točno određeni klasifikator zapisuje odvojena točkom. Primjerice akcija za klasifikator cl1 označavat će se cl1.A.

Uvjet svakog klasifikatora uspoređuje se sa stanjem okoline, a klasifikatori čiji su uvjeti zadovoljeni za određeno stanje definiraju moguće akcije. Za prikaz uvjeta moguće je koristiti binarni, realni ili neki složeniji način zapisivanja, ovisno o vrsti problema koji se rješava. U nastavku rada koristi se binarni prikaz koji je najčešće korišten.

Kod binarnog zapisa vrijedi $C \epsilon \{0,1,\#\}^l$, gdje je *l* duljina zapisa koja ovisi o broju bitova kojim se kodira stanje okoline. Simboli 0 i 1 su *konkretni simboli* i odgovaraju bitovima kojima se zapisuje stanje. Kako bi se uvjet podudarao sa stanjem moraju se podudarati bitovi stanja i konkretni simboli uvjeta. Simbol # je *slobodni simbol* (eng. *don't care*) i označava da na odgovarajućem mjestu u zapisu stanja može biti bilo koji bit kako bi se uvjet podudarao.

Svaki uvjet pravila predstavlja podprostor problema. Klasifikatori čiji se podprostori problema preklapaju natječu se za nagradu dobivenu od okoline.

Veličina podprostora problema koju klasifikator pokriva određena je *faktorom specifičnosti* k/l, gdje *k* označava broj konkretnih simbola [5]. Klasifikatore sa manjim faktorom specifičnosti nazivamo *generalizacijama* i oni pokrivaju veći prostor problema od specifičnih klasifikatora. Upravo je mogućnost generalizacije glavna razlika između tehnike Q-učenja i LCS-a, a omogućuje kompaktniji zapis klasifikatora za razliku od Q-učenja gdje veličina tablice sa pohranjenim Q-vrijednostima može jako narasti [1].



Slika 4.1 Struktura LCS sustava

Za prikaz akcije također su moguće razne varijante, a budući da se u problemima koji će se kasnije razmatrati koristi manji broj akcija, u nastavku se koristi obični binarni zapis (samo konkretni simboli 0 i 1). Parametri klasifikatora ovise o vrsti klasifikatorskog sustava koji ih koristi. Dobrota je zajednički parametar za sve vrste, a određuje mjeru uspješnosti klasifikatora. Njezina uloga je ključna kod rada genetičkih algoritama prilikom razvoja populacije klasifikatora te kod odabira akcije koju će sustav poslati kao odgovor okolini. Odabir akcije obavlja se u skladu sa strategijom ponašanja koju razvija komponenta za podržano učenje.

4.2 Vrste LCS-a

LCS sustave uveo je i formalizirao John Holland. Problem sa kojim je započeo svoje istraživanje bila je interakcija sustava sa okolinom, kojeg je kasnije opisao Stewart Wilson kao problem animata. Radi se o ranije opisanom snalaženju agenta unutar labirinta u kojem je potrebno pronaći određeni cilj.

U samim počecima razvoja klasifikatorskih sustava pojavila su se dva odvojena pristupa [2]. Holland i njegovi studenti sa sveučilišta u Michiganu začetnici su Michiganskog pristupa, kod kojeg jedan LCS pomoću genetičkog algoritma i podržanog učenja, razvija populaciju klasifikatora kao rješenje problema. Drugi pristup je Pittsburški, kojeg je započeo Stephen Frederick Smith sa sveučilišta u Pittsburghu. Njegova vrsta sustava radi sa više populacija klasifikatora, između kojih se također pomoću GA razvija ona koja predstavlja najbolji LCS za dani problem. Razlika između Pittsburškog i Michiganskog pristupa je također u načinu rada. Pittsburški LCS sustavi rade iterativno nad skupovima problema, što je vrsta *off-line* načina rada, dok su Michiganski građeni za *on-line* način rada [15].

Prvu konkretnu implementaciju klasifikatorskog sustava u Michiganskom stilu, objavili su Holland i Reitman 1978., pod nazivom "CS1" (Cognitive System One) [1]. Sustav od okoline dobiva kodirano stanje koje smješta na *listu poruka*. Po primitku stanja iz populacijskog skupa klasifikatora (eng. *population set*, [P]) izdvajaju se oni čiji su uvjeti zadovoljeni za bilo koju poruku na listi te se smještaju u podudarni skup (eng. *match set*, [M]). Pomoću mehanizma glasanja odabire se klasifikator čija će akcija predstavljati akciju sustava. Glas pojedinog klasifikatora računa se po formuli [2]:

$$glas(cl.C) = \beta \cdot specificnost(cl.C) \cdot cl.dobrota \quad \forall cl \in [P] \tag{4.1}$$

Zapis akcije sustava dopuštao je slobodne simbole, koji su u tom slučaju označavali propuštanje simbola iz dobivenog zapisa stanja [2]. Lista poruka se briše te se na nju postavlja akcija od odabranog klasifikatora. Dalje se nastavlja sa popunjavanjem liste po istom principu glasanja, dok ima slobodnog mjesta. Glas svakog pobjednika pohranjuje se u "kantu" (eng. *bucket*) čiji se sadržaj podjednako dijeli prethodnim pobjednicima u korist njihove dobrote. Navedena tehnika podržanog učenja poznata je pod nazivom *živi lanac* (eng. *bucket brigade*) [1]. GA se primjenjuju nad cijelim skupom [P] u svrhu otkrivanja novih klasifikatora.

Klasifikatorski sustavi u Michiganskom stilu prema načinu određivanja uspješnosti klasifikatora dijele se na sustave:

- temeljene na snazi (eng. *strenght-based*),
- temeljne na preciznosti (eng. *accuracy-based*) te
- temeljene na očekivanju (eng. *anticipatory-based*).

Kod sustava temeljenih na snazi kao mjera uspješnosti uzima se samo predviđena nagrada (parametar klasifikatora). Primjer takvog sustava je ZCS (eng. *zeroth level classifier system*). Kod njih se međutim javlja problem pretjerane generalizacije, gdje klasifikatori sa manjim faktorom specifičnosti imaju veću dobrotu [5]. Iz tog razloga oni postepeno nadjačavaju specifične klasifikatore čija bi akcija u određenim stanjima eventualno dala bolju nagradu. Razlog tome je što su uvjeti generalnih klasifikatora često zadovoljeni i predviđena nagrada se povećava zahvaljujući različitim stanjima u kojima se akcija izvršava.

Poboljšanje po pitanju pretjerane generalizacije donose klasifikatorski sustavi temeljni na preciznosti. Kod njih se traži da klasifikator bude točan, tj. da točno predviđa nagradu bez većih odstupanja. Primjer takvog sustava je XCS čiji je detaljniji opis dan u sljedećem poglavlju.

Klasifikatorski sustavi temeljeni na očekivanju razlikuju se od prethodno navedenih po tome što klasifikatori osim uvjeta i akcije sadrže očekivanje (eng. *effect*) koje predstavlja očekivano buduće stanje okoline. Očekivanje sadrži dodatne simbol ? (eng. *don't know*), koji označava da se odgovarjući bit stanja ne može predvijeti te simbol = (eng. *don't change*), koji označava da se bit u budućem stanju neće promijeniti. Dobrota klasifikatora određuje se prema uspješnosti predviđanja sljedećeg stanja okoline [1].

5. Klasifikatorski sustav XCS

XCS klasifikatorski sustav, sustav je temeljen na preciznosti u Michiganskom stilu. Kod njega se za razliku od ostalih vrsta LCS-a dobrota klasifikatora ne bazira na predviđenoj nagradi već na preciznosti tog predviđanja. Želja je pomoću učinkovitih generalizacija ostvariti precizno klasificiranje prostora problema [2]. S tim načinom vrednovanja rješen je i ranije spomenuti problem pretjerane generalizacije koji se javlja kod LCS-a temeljenih na snazi.

5.1 Parametri klasifikatora

Kao i kod ostalih LCS-a Michiganskog stila, XCS razvija populaciju klasifikatora. Osnovne komponente klasifikatora su pravilo, akcija i parametri. Pravilo i akcija su ranije opisani, a osnovni parametri su:

P - predviđanje nagrade (eng. *reward prediction*)

F - dobrota

ε- pogreška predviđanja nagrade (eng. *reward prediction error*)

Parametar cl.P ϵ $\mathbb{R}$ označava predviđenu vrijednost nagrade, ako je zadovoljen uvjet cl.C i izvršena je akcija cl.A. Računanje nove vrijednosti parametra odvija se iterativno nakon svake dobivene nagrade od okoline. Rezultat je pomična prosječna vrijednost[2] (eng. *moving average*) dobivene nagrade za određeni klasifikator. Slično pogreška predviđanja ε predstavlja pomičnu prosječnu vrijednost apsolutne pogrešku predviđanja nagrade. Dobrota se također računa kao pomična prosječna vrijednost preciznosti klasifikatora u odnosu na ostale klasifikatore koji se mogu primijeniti u istom trenutku.

Uz navedene osnovne, svaki klasifikator posjeduje i sljedeće dodatne parametre:

as - procjena veličine akcijskog skupa (eng. *action set size estimate*)

ts - vremenska oznaka (eng. *time stamp*)

exp - iskustvo (eng. *experience*)

n - brojnost (eng. *numerosity*)

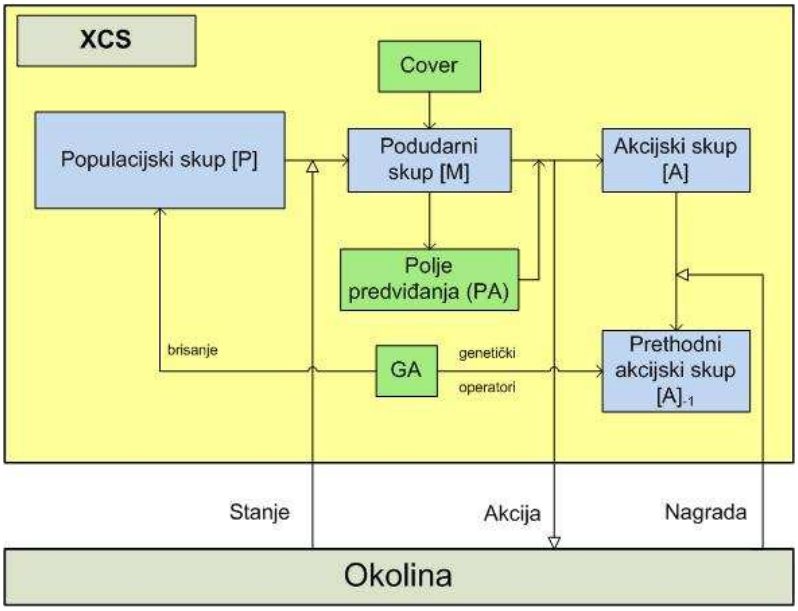
[2] Sa svakom novom primljenom vrijednosti, dobiva se prosječna vrijednost svih do tada primljenih

Parametar *as* predstavlja prosječnu veličinu akcijskih skupova u kojima se klasifikator nalazio. Služi prilikom donošenja odluke o brisanju pojedinog klasifikatora iz populacije, čime se pokušava održati jednolike veličine akcijskih skupova [6]. Vremenska oznaka služi za donošenje odluke o pokretanju GA, a predstavlja trenutak stvaranja klasifikatora ili zadnjeg izvršavanja GA. Iskustvo je broj koliko puta se klasifikator nalazio u podudarnom skupu ([M]). Parametar brojnosti predstavlja broj mikro-klasifikatora koje obuhvaća makro-klasifikator. Time je smanjen broj nepotrebnih klasifikatora u populaciji, jer je uloga mikro-klasifikatora obuhvaćena makro-klasifikatorom. Mehanizam obuhvaćanja detaljnije je objašnjen u poglavlju o implementaciji.

5.2 Algoritamski opis sustava

Prije početka rada sustava obavljaju se svi pripremni postupci poput inicijalizacije početne populacije klasifikatora te inicijalizacije okoline. Rad XCS klasifikatorskog sustava odvija se kroz iterativnu interakciju sa okolinom (slika 5.1). Glavni koraci svake iteracije su:

- dohvaćanje stanja okoline
- korištenje klasifikatora za odabir akcije
- slanje akcije okolini
- dohvaćanje nagrade od okoline
- obnavljanje parametara klasifikatora
- moguće pokretanje GA



Slika 5.1 Struktura XCS sustava

Skup svih klasifikatora naziva se *populacijski skup* (eng. *population set*, [P]), a maksimalan broj klasifikatora koje može sadržavati je parametar sustava *N*. Nakon dobivenog stanja okoline iz skupa [P] izdvajaju se svi klasifikatori čiji se uvjet podudara sa stanjem i tvori se *podudarni skup* (eng. *match set*, [M]). Prilikom stvaranja skupa parametrom θ_{mna} moguće je odrediti minimalni broj različitih akcija koje klasifikatori iz skupa trebaju imati. Ukoliko se ne može zadovoljiti uvjet koristi se postupak *pokrivanja* (eng. *cover*) kojim se stvaraju novi klasifikatori s potrebnim akcijama. Iz dobivenog skupa [M] na temelju parametara klasifikatora odabire se akcija koju će sustav poslati kao odgovor. Za odabir akcije stvara se *polje predviđanja* (eng. *prediction array*, PA). PA određuje predviđenu nagradu za pojedinu akciju A, koju je moguće odabrati koristeći klasifikatore iz skupa [M]. Predviđena nagrada računa se kao težinska srednja vrijednost predviđanja, gdje je težinski faktor dobrota klasifikatora:

$$PA(cl.A) = \frac{\sum_{c.A=A, c\in[M]} c.P \cdot c.F}{\sum_{c.A=A, c\in[M]} c.F}, \quad \forall cl \in [M]. \tag{4.1}$$

Za odabir akcije odgovorna je komponenta podržanog učenja uz primjenu određenog mehanizma, među kojima je često korišten ε *–greedy mehanizam*:

$$\pi_{XCS}(s) = \begin{cases} \arg\max_{A}(PA(A)), & 1 - p_{\exp lore} \\ rand([M]), & p_{\exp lore} \end{cases}. \tag{4.2}$$

Nakon odabira akcije A gradi se akcijski skup (eng. *action set*, [A]), koji sadrži sve klasifikatore iz [M] sa akcijom A. Akcija se šalje okolini na evaluaciju. Interno stanje okoline se mijenja u skladu s akcijom te se kao povratna vrijednost sustavu šalje nagrada *r*. Na temelju nagrade računa se vrijednost ukupne nagrade *R* pomoću koje se obnavljaju parametri klasifikatora iz skupa [A]:

$$R = r + \gamma \cdot \max_{A}(PA^{t+1}(A)). \tag{4.3}$$

```
Eksperiment:
       nagrada-1 = 0;

       ponavljaj:
            stanje = okolina.stanje();
            [M] = stvoriPodudarniSkup([P], stanje);
            PA = stvoriPA ([M]);
            akcija = odaberiAkciju(PA);
            [A] = stvoriAkcijskiSkup([M], akcija);

            nagrada = okolina.evaluiraj(akcija);

            ako ([A]-1 nije prazan):
                 R = nagrada-1 + γ * max(PA);
                 obnoviParametre([A]-1, R, [P]);
                 pokreniGA([A]-1, stanje-1, [P]);

            ako (okolina.krajProblema()):
                 obnoviParametre([A], nagrada, [P]);
                 pokreniGA([A], stanje, [P]);
                 obrisi([A]-1);
            inače:
                 [A]-1 = [A];
                 stanje-1 = stanje;
                 nagrada-1 = nagrada;

       dok (uvjet zaustavljanja nije zadovoljen);
```

Slika 5.2 Pseudokod glavnog algoritma XCS-a

Polje predviđanja PA^{t+1} označava PA u sljedećem trenutku nakon dobivanja nagrade, a parametar γ je faktor odbitka sa istom ulogom kao i kod Q-učenja. Konkretna algoritamska implementacija računanja vrijednosti *R* i obnavljanja parametara radi se pomoću trenutnog polja predviđanja nad prethodnim akcijskim skupom $[A]_{-1}$, a prikazana je na slici 5.2. Ukoliko je u trenutku *t* kraj problema, odnosno u svakoj iteraciji kod jednokoračnih problema, vrijedi R = r.

Novi parametri klasifikatora iz akcijskog skupa [A] računaju se korištenjem *Widrow-Hoff delta pravila* [11], sljedećim redoslijedom:

1. Pogreška predviđanja ε za svaki klasifikator obnavlja se prema izrazu:

$$cl.\varepsilon \leftarrow cl.\varepsilon + \beta(|\,\mathrm{R} - \mathrm{cl.P}\,| - \mathrm{cl.}\varepsilon), \quad \forall \mathrm{cl} \in [\mathrm{A}]. \tag{4.4}$$

Parametar β je stopa učenja sa istim značenjem kao u izrazu 2.7 kod Q-učenja.

2. Predviđanje nagrade *P* za svaki klasifikator obnavlja se prema sljedećem izrazu:

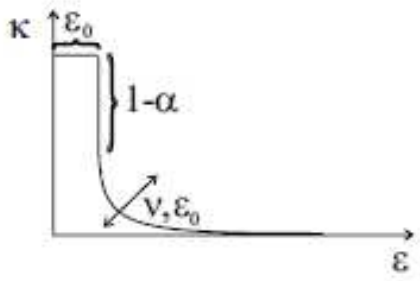
$$cl.P \leftarrow cl.P + \beta(\mathrm{cl.P} - \mathrm{R}), \quad \forall \mathrm{cl} \in [\mathrm{A}]. \tag{4.5}$$

XCS rabi isti mehanizam obnove predviđanja kao i Q-učenje uz razliku što kod njega Q-vrijednosti nisu izražene tablično za svaku akciju i stanje već se nalaze unutar polja predviđanja PA.

3. Za svaki klasifikator pomoću vrijednosti ε računa se preciznost κ, prema sljedećem izrazu:

$$\kappa(cl) = \begin{cases} 1, & cl.\varepsilon < \varepsilon_0 \\ \alpha\left(\frac{cl.\varepsilon}{\varepsilon_0}\right)^{-\nu}, & cl.\varepsilon \geq \varepsilon_0 \end{cases}, \quad \forall cl \in [A]. \tag{4.6}$$

Parametar sustava ε_0 predstavlja prag tolerancije na pogrešku predviđanja. Klasifikatori za koje vrijedi $\varepsilon < \varepsilon_0$ smatraju se preciznima. Odnos preciznosti i pogreške predviđanja u ovisnosti o parametrima ε_0, α i ν prikazan je na slici 5.3.



Slika 5.3 Ovisnost preciznosti o pogrešci predviđanja [5]

4. Za svaki klasifikator iz skupa [A] računa se relativna preciznost κ' u odnosu na ostale klasifikatore iz skupa. Ona predstavlja udio preciznosti pojedinog klasifikatora u ukupnoj preciznosti svih klasifikatora iz akcijskog skupa te se računa prema sljedećem izrazu:

$$\kappa'(cl) = \frac{\kappa(cl) \cdot cl.n}{\sum_{c \in [A]} \kappa(c) \cdot c.n}, \quad \forall cl \in [A]. \tag{4.7}$$

5. Korištenjem dobivene relativne preciznosti dobrota F obnavlja se prema izrazu:

$$cl.F \leftarrow cl.F + \beta(\kappa' - F), \quad \forall cl \in [A]. \tag{4.8}$$

6. Procjena veličine akcijskog skupa *as* obnavlja se prema izrazu:

$$cl.as \leftarrow cl.as + \beta(|[A]| - as), \quad \forall cl \in [A]. \tag{4.9}$$

5.3 Evolucijska komponenta

Evolucijska komponenta korištena u XCS sustavu je genetički algoritam, koji se provodi nad akcijskim skupom. Razlog tome je želja da klasifikatori roditelji budu primjenjivi na istom podprostoru problema [5]. Također, primjena GA nad akcijskim skupom služi za razvoj generalizacija, jer generalizirani klasifikatori su često u skupu [A] te imaju veću mogućnost reprodukcije [13].

GA koji se primjenjuje kod XCS-a je najčešće eliminacijski [1]. Određenim selekcijskim postupkom (najčešće jednostavna selekcija) ovisno o dobroti iz skupa [A] biraju se dva roditelja [6]. Njihovim križanjem dobivaju se dva potomka, koji zamjenjuju lošije klasifikatore, također odabrane jednostavnom selekcijom.

Križanje klasifikatora obavlja se nad uvjetima roditelja prema slici 2.1, uz uzimanje u obzir slobodnih simbola. Mutacija nad uvjetom se obavlja tako da se pojedini konkretni simboli pretvaraju u slobodne i obratno. Zbog toga je potrebno poznavanje trenutnog stanja okoline pomoću kojeg je stvoren skup [M], kako bi se uvjet i nakon mutacije podudarao sa stanjem.

Kako bi se GA primjenjivao jednoliko nad svim akcijskim skupovima te u jednakim intervalima, koristi se vremenska oznaka (ts) [11]. Prilikom stvaranja klasifikatora parametru *ts* pridružuje se vrijeme kada je stvoren. Kod stvaranja skupa [A] za sve klasifikatore gleda se prosječna vrijednost oznake te ukoliko je razlika u odnosu na trenutno vrijeme veća od praga tolerancije θ_{GA}, nad skupom se pokreće GA. Također, ukoliko se GA izvršio, svim klasifikatorima vremenska oznaka se postavlja na trenutno vrijeme sustava.

6. Primjena XCS-a na višekriterijske probleme

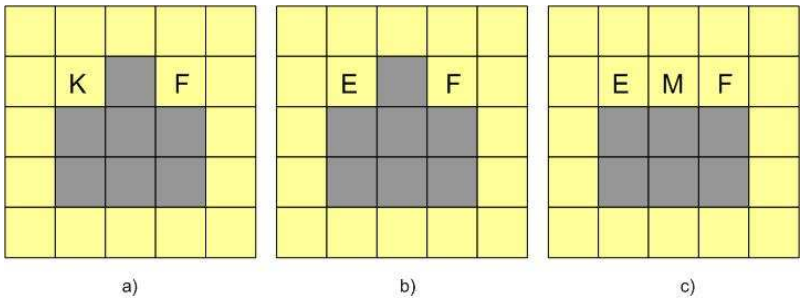
Svaki živi organizam posjeduje svojstvo homeostaze, što mu omogućava da uz promjenjivu okolinu zadrži relativno konstantno unutarnje stanje [7]. Pri tome se neprestano suočava sa raznim oblicima ciljeva koje mora ostvariti, poput nabavljanja hrane, traženja zaklona i slično. Odabir akcija vrši se prema određenim kriterijima i dinamički se mijenja ovisno o promjenama u okolini.

Navedena vrsta problema, kod kojih postoji više ciljeva i određeni kriteriji njihovog izvršavanja nazivaju se *višekriterijski problemi*, a predstavljaju posebnu vrstu višekoračnih problema. Kao problem za primjenu XCS sustava odabran je ranije objašnjen problem labirinta uz proširenja sa više traženih ciljeva. Informacija o ostvarenju pojedinih ciljeva sustavu se predaje preko zapisa stanja.

6.1 Slijedni višekriterijski problem labirinta

Jedan od načina da se problem labirinta proširi na višekriterijski je dodavanje uvjetnog cilja (ključ), čije ostvarenje omogućuje ostvarenje glavnog cilja (skupljanje hrane). Ukoliko animat pronađe hranu bez sakupljenog ključa dobivena nagrada iznosi 0, a u suprotnom 1000 [7].

Okolina *Maze1k* (slika 6.1 a) predstavlja kvadratni 2D labirint sa 25 polja. Rubovi labirinta povezani su na način da pomak izvan granica labirinta prebacuje animata na suprotnu stranu. Kretanje je dozvoljeno po praznim poljima (označeno sivo) i to u 8 mogućih smjerova oko trenutne pozicije. Svaki nedozvoljeni pomak se ignorira i rezultira sa nagradom 0. Ispitivanje započinje postavljanjem animata na nasumično prazno polje, a završava kada je pronađena hrana uz sakupljeni ključ.



Slika 6.1 Tri višekriterijske okoline (K – ključ, E –izvor energije, M – održavanje)

Za zapis pojedinog polja u stanju okoline koriste se 3 bita, a na kraju se dodaje dodatni bit koji označava da li je posjećeno polje sa ključem.

6.2 Paralelni višekriterijski problem labirinta

Osim korištenja uvjetnog cilja, problemu labirinta moguće je dodati više ciljeva koje je potrebno istovremeno zadovoljiti. Ovisno o visini mogućih nagrada, animat će sam morati usvojiti određene kriterije za redoslijed ostvarivanja ciljeva.

Prvi primjer je okolina *Maze1e* (slika 6.1 b), gdje je dodano polje za prikupljanje energije (oznaka E) [7]. Animat posjeduje unutarnji spremnik sa energijom, maksimalnog kapaciteta 1. Na početku svakog ispitivanja količina energije pohranjena u spremniku je nasumičan realni broj iz intervala [0,1]. Kod opisa stanja okoline uz opise polja nalazi se dodatni bit koji označava da li je količina pohranjene energije veća od 0.5. Ukoliko animat posjeti polje *E* uz manje od 0.5 količine energije, dobiva nagradu u iznosu 1000, a u suprotnom 1. Također, kada se nađe na polju *F* i ima više od 0.5 količine energije, dobiva nagradu 1000, a u suprotnom 1.

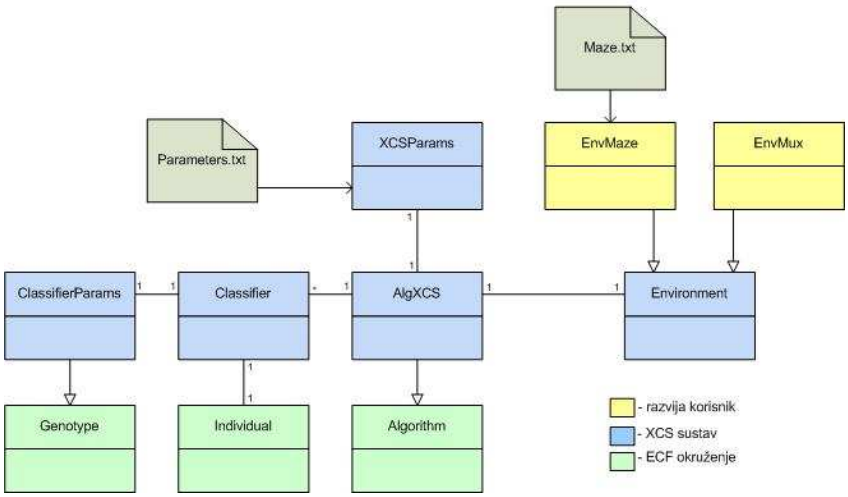
Prethodni primjer proširen za još jedan cilj predstavlja okolinu *Maze1em* (slika 6.1 c), gdje je dodano polje za "održavanje" (oznaka M) [7]. Potreba za održavanjem se također označava realnim brojem iz intervala [0,1] i uz dodatan bit kod zapisa stanja koji označava da li je ona veća od 0.5. Prioritet održavanja veći je od prethodna dva cilja, pa će tako ukoliko je njegova vrijednost veća od 0.5, animat dobiti nagradu 1 bez obzira koji od njih obavi. Međutim ukoliko u tom slučaju posjeti polje označeno sa *M* dobiva nagradu od 1000, a 1 ukoliko je potreba za održavanjem manja od 0.5. Kada je održavanje manje od 0.5 ostala dva cilja se zadovoljavaju kao u prethodnom primjeru. Na početku simulacije vrijednost održavanja je nasumičan broj iz intervala [0,1].

7. Programsko ostvarenje XCS sustava

Programsko ostvarenje klasifikatorskog sustava XCS sastoji se od implementacije XCS komponente u sklopu okruženja ECF te izgradnje simulacijske okoline koja se njome služi. XCS je u ECF-u implementiran kao algoritam i podržava rad sa jednokoračnim i višekoračnim problemima što se ostvaruje implementacijom odgovarajuće okoline. Kod simulacijskog okruženja navedena komponenta je iskorištena za rješavanje višekoračnih problema labirinta uz odgovarajuću grafičku simulaciju, interaktivni prikaz populacije klasifikatora te uređivanje parametara sustava.

7.1 XCS komponenta u sklopu ECF-a

XCS klasifikatorski sustav implementiran je u sklopu ECF okruženja za evolucijsko računanje, kao zasebna algoritamska komponenta *AlgXCS*. Za njeno ostvarenje razvijene su dodatne komponente *XCSParams*, *Classifier*, *ClassifierParams* i *Environment* (Slika 7.1). *XCSParams* služi za enkapsulaciju parametara sustava i omogućuje njihovo učitavanje i pohranjivanje u formatu koji koristi ECF okruženje za parametre svakog algoritma. Komponenta *Classifier* predstavlja jedan klasifikator u populaciji. Njoj je prepuštena sva odgovornost oko pravila i akcije, čime je omogućena kasnija nadogradnja uz korištenje drukčijeg načina njihovog zapisivanja. O ostalim parametrima klasifikatora odgovorna je komponenta *ClassifierParams*. Implementacija je, kao i okruženje, napisana u programskom jeziku C++.



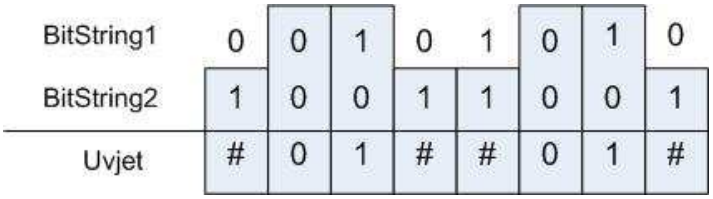
Slika 7.1 Organizacija komponenti XCS sustava u sklopu ECF okruženja

Komponenta *Environment* predstavlja okolinu sa kojom je sustav u interakciji. Poput uvjeta klasifikatora, stanje okoline je također predstavljeno genotipom. Time je omogućena lakša usporedba podudarnosti stanja i uvjeta te potencijalno kasnije proširivanje na neki drugačiji zapis od binarnog.

Pohrana komponenti klasifikatora

Razred *Classifier* predstavlja jedan klasifikator iz populacije. On u sebi sadrži sve komponente klasifikatora, koji su: uvjet, akcija i parametri. Za pohranu uvjeta i akcije koriste se genotipovi (komponenta *Genotype*) unutar jedinke sa kojom radi ECF (komponenta *Individual*). Ovisno o željenom načinu zapisivanja, za uvjet se može koristiti proizvoljan broj genotipa, a za akciju samo jedan.

Implementacija klasifikatora sa binarnim zapisom uvjeta i akcije koristi tri BitString genotipa, dva za uvjet i jedan za akciju. Način pohrane uvjeta prikazan je na slici 7.2.



Slika 7.2 Zapisivanje uvjeta pomoću BitString genotipova

Jedinka ECF-a sadrži još i parametar dobrote klasifikatora što omogućava primjenu gotovih selekcijskih postupaka okruženja, kod rada genetičkog algoritma. Ostali parametri pohranjeni su u zasebnom genotipu *ClassifierParams* koji se brine za njihovu pohranu u datoteku prilikom zapisivanja populacije.

Komponenta *Classifier* zadužena je za obavljanje genetičkih operacija nad klasifikatorom, jer se križanje i mutacija obavlja specifično za vrstu genotipa koji se koristi za zapis uvjeta i akcije. Iako ECF podržava vlastite operatore za svaku vrstu genotipa, ovakav pristup korišten je zbog specifičnosti mutacije uvjeta.

Komponenta *Environment* predstavlja sučelje prema okolini sa kojom sustav radi. Tako su za problem labirinta implementirani razredi *MazeEnv*, *SingleObjMazeEnv*, *SeqObjMazeEnv*, *TwoObjMazeEnv* te *ThreeObjMazeEnv*. Okolina se na početku pokretanja sustava inicijalizira, što uključuje učitavanje labirinta iz datoteke.

Stanje labirinta šalje se XCS sustavu kao *BitString*, a općenito se može koristiti bilo koji oblik genotipa. Usporedbu uvjeta klasifikatora sa stanjem vrši komponenta *Classifier*, što omogućuje kasniju nadogradnju za drukčije načine zapisivanja stanja okoline. Ispitivanje sustava moguće je pomoću metode *isExploit*. Ako se želi da eksperiment služi za ispitivanje, pomoću navedene metode određuje se strategija iskorištavanja te će sustav obavljati najbolje akcije bez obnavljanja parametara klasifikatora.

Implementacija algoritma

Svaki eksperiment obavlja se pokretanjem algoritma sustava za što je odgovorna ECF komponenta *State*. Prije pokretanja algoritma obavlja se inicijalizacija koja uključuje učitavanje parametara iz konfiguracijske datoteke, inicijalizaciju operatora selekcije, incijalizaciju okoline te stvaranje početne populacije klasifikatora. Inicijalizacija okoline je specifična za problem koji ona predstavlja. Tako se kod komplonente *MazeEnv* za vrijeme inicijalizacije učitava labirint iz datoteke.

Svaki eksperiment sastoji se od više manjih provjera čiji je broj ograničen ECF parametrom *term.maxgen*. Jedna provjera označava rješavanje jedne instance problema, primjerice jedan prolazak kroz labirint pri čemu je pronađen cilj. Svaka provjera vrši se pokretanjem metode *advanceGeneration*, razreda *AlgXCS*. Nakon njenog izvršavanja, okolina se priprema za novu provjeru pomoću metode *reset*.

Svaka provjera predstavlja više iteracija algoritma XCS sustava dok se ne zadovolji određeni uvjet u okolini, primjerice pronađen je cilj u labirintu. Provjera da li je taj uvjet zadovoljen ostvaruje se metodom *isOver*, razreda *Environment*.

Algoritam sustava ostvaren je prema pseudokodu na slici 5.2. Kod obnavljanja parametara klasifikatora korištena je MAM tehnika (eng. *moyenne adaptive modifiee*) [11], kojom se vrijednost parametara postavlja na srednju vrijednost ukoliko je iskustvo klasifikatora manje od 1/β. Primjer obnavljanja parametra predviđanja korištenjem MAM tehnike je sljedeći izraz:

$$cl.p \leftarrow \begin{cases} cl.p + \dfrac{(R - cl.p)}{cl.\exp}, & cl.\exp < \dfrac{1}{\beta} \\ cl.p + \beta(R - cl.p), & cl.\exp \geq \dfrac{1}{\beta} \end{cases}. \tag{6.1}$$

Obnavljanjem paramatera kroz dvije faze omogućuje da početne vrijednosti parametara brzo konvergiraju ka njihovim prosječnim vrijednostima [11].

Algoritam podržava i mehanizam *obuhvaćanja* (eng. *subsumption*), kojim je moguće izbaciti klasifikatore čiji je uvjet obuhvaćen uvjetom drugog klasifikatora sa manjim faktorom specifičnosti, a klasifikatori imaju iste akcije [6]. Na slici 7.3 prikazan je pseudokod funkcije koja određuje da li klasifikator cl_{gen} obuhvaća cl_{spec}. Operaciju te funkcije obavlja metoda *doesSubsume* u razredu *Classifier*. Ukoliko je klasifikator cl_{spec} obuhvaćen nekim generaliziranijim cl_{gen}, on se izbacuje iz populacije, a parametar cl_{gen}.n uvećava se za brojnost cl_{spec}.n. Navedeni mehanizam primjenjuje se nad akcijskim skupom nakon obnove parametara klasifikatora, pomoću metode *actionSetSubsumption*. Pri tome se za cl_{gen} uzima klasifikator sa najmanjim faktorom specifičnosti i uspoređuje se sa ostalim klasifikatorima iz skupa [A]. Također kod pokretanja GA provjerava se da li roditelji podrazumijevaju dobivene potomke te se u tom slučaju samo povećava brojnost roditelja, bez uključivanja potomaka u skup [P].

```
Obuvaćа(cl_gen, cl_spec):

       ako (cl_gen.A = cl_spec.A &&
              cl_gen.exp > θ_sub && cl_gen.ε < ε_0 &&
            broj # u cl_gen.C > broj # u cl_spec.C):

                     i = 0;
                     ponavljaj:
                            ako (cl_gen.C[i] ≠ #
                                 && cl_gen.C[i] ≠ cl_spec.C[i]):
                                    vrati NE;
                            i++;
                     dok (i < duljina cl_gen.C);
                     vrati DA;

       vrati NE;
```

Slika7.3 Pseudokod mehanizma za obuhvaćanje

Jedna od zadaća algoritma je i održavanje veličine populacije u granicama parametra N. Mogući načini povećanja broja klasifikatora u populaciji su slijedeći:

- prilikom inicijalizacije algoritma;
- pokretanjem *cover* mehanizma;
- pokretanjem GA.

Prvi način koristit parametar algoritma N_I, kojim je određena početna veličina populacije i mora vrijediti $N_I \leq N$.

Međutim, kod druga dva načina moguće je da veličina populacije prekorači granicu N, te je potreban mehanizam za izbacivanje nepotrebnih klasifikatora [6]. To se u algoritmu obavlja metodom *deleteFromPopulation*, koja obavlja jednostavnu selekciju nad cijelom populacijom, te odabrani klasifikator uklanja iz skupa [P]. Selekcija se vrši u odnosu na visinu glasova koju pojedini klasifikator dobije za izbacivanje. Glas za izbacivanje se računa prema pseudokodu na slici 7.4.

```
GlasZaBrisanje (cl, prosjecnaDobrota):
       glas = cl.as * cl.n;
       ako (cl.exp > θdel && cl.F / cl.n < δ *
prosjecnaDobrota):
              glas = glas * prosjecnaDobrota / (cl.F / cl.n);
       vrati glas;
```

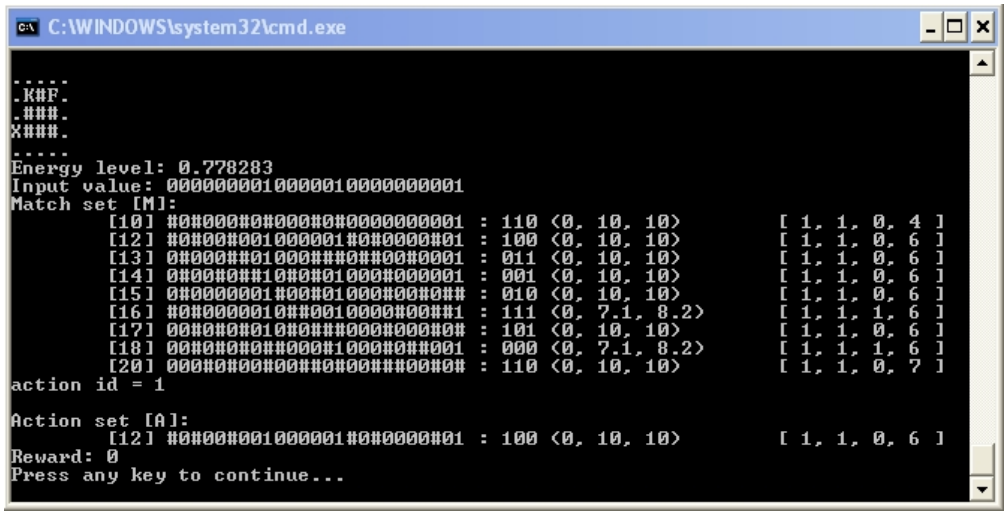
Slika7.4 Pseudokod za izračuavanje glasa za brisanje iz populacije

GA komponenta dijelom je ostvarena korištenjem postojećih komponenti ECF-a, a dio odgovornosti je prepušten komponenti *Classifier*. Za selekciju roditelja koristi se razred *SelFitnessProportionalOp*. Križanje se obavlja dvostrukim pozivanjem metode *mate* nad odabranim roditeljima kako bi se dobila dva potomka. Tu se algoritam razlikuje od uobičajenog postupka, gdje se to obavlja jednim postupkom križanja. Mutaciju vrši razred *Classifier* pomoću metoda *mutateRule* i *mutateAction*. Razlog za izdvajanjem mutacije pravila od postojećih postupaka ECF-a je da se nakon mutacije pravilo mora podudarati sa stanjem okoline. Mutacija akcije je izdvojena jer je poželjno da nova akcija bude postojeća.

Primjena XCS komponente

Rješavanje određenog problema sa klasifikatorskim sustavom svodi se na implementaciju konkretnog razreda koji nasljeđuje *Environment* te uređivanje konfiguracijske datoteke. U njoj se nalaze parametri XCS sustava, početne vrijednosti parametara klasifikatora, postavke genotipa koji se koriste za pravila i akcije te općenite ECF postavke. Prilikom definiranja postavki genotipa potrebno je obratiti pažnju da su one u skladu sa postavkama genotipa koji se koristi za opis stanja okoline. Kod binarnog zapisa to znači da uvjet i opis stanja ne smiju imati različit broj simbola.

Kako bi se spriječio takav slučaj, za vrijeme inicijalizacije sustava šalje se njegovo trenutno stanje komponentama *Classifier* i *Environment* da provjere odgovaraju li postavke njihovom načinu rada.



Slika 7.5 Rad sustava uz detaljan ispis u konzolu

Implementacijom konkretnog razreda okoline moguće je modelirati jednokoračne i višekoračne okoline. Također, prilikom prevođenja XCS komponente pomoću pretprocesorske *define* direktive moguće je definirati simbole XCS_DEBUG i XCS_STEP_DEBUG, čime se omogućuje detaljan ispis prilikom rada sustava kao i rad u koracima (slika 7.5).

7.2 Ostvarenje simulacijske okoline

Za lakše prikazivanje rada XCS sustava, napravljena je dodatna simulacijska okolina koja koristi XCS komponentu ECF okruženja i pomoću grafičkog sučelja prikazuje njen rad. Sučelje je ostvareno korištenjem Windows Forms komponenti u programskom alatu Visual C++, koji se još naziva i upravljani C++ (eng. *managed*).

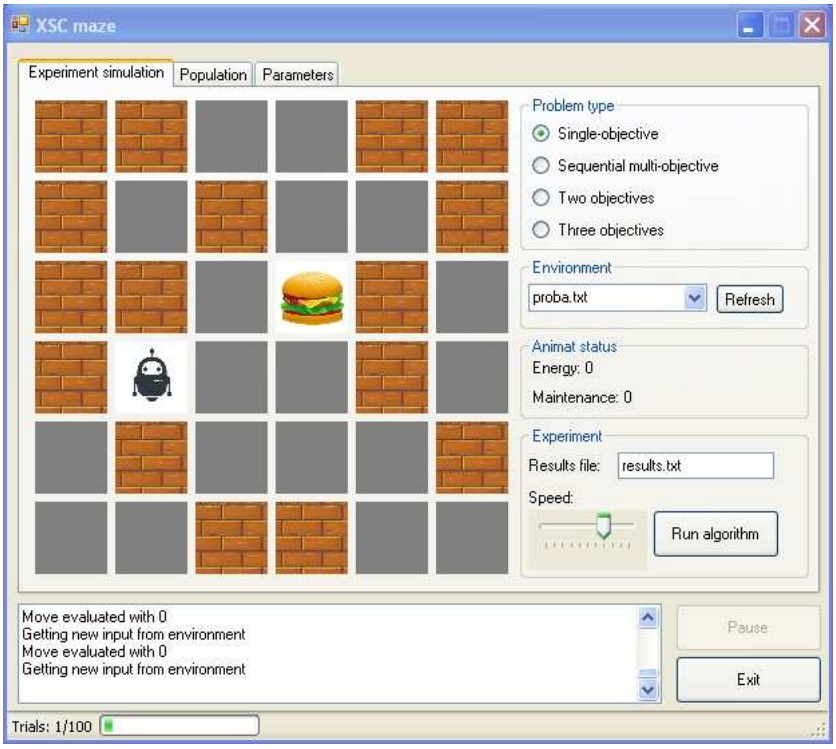
Sučelje programa sastoji se od tri glavna dijela odvojena zasebnim karticama:

- simulacija eksperimenta,
- tablični prikaz populacije klasifikatora i
- parametri algoritma.

Uz donji rub prozora nalazi se prostor za ispis poruka o radu sustava, gumb za pauziranje eksperimenta te gumb za izlaz iz programa.

Kartica sa prikazom simulacije sadrži grafički prikaz labirinta, odabir vrste problema, odabir labirinta, kontrolu za određivanje brzine simulacije, prikaz statusa animata te gumb za pokretanje eksperimenta (slika 7.6). Ponuđene vrste problema su jednokriterijski te tri vrste objašnjene u 5. poglavlju. Za svaku vrstu, moguće je odabrati više labirinata koji se nalaze u odgovarajućem određenom pod-direktoriju unutar direktorija *environments*. Labirinti se opisuju u tekstualnim datotekama uz korištenje slijedećih znakova:

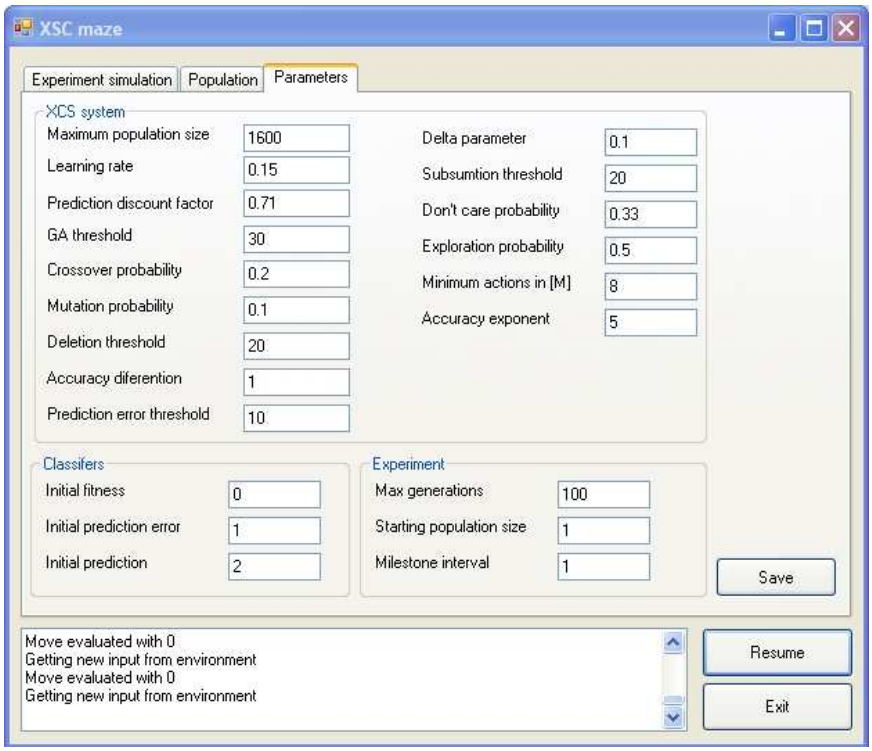
- . – prazno polje,
- # – zid,
- F – polje sa hranom,
- K – polje sa ključem,
- E – polje sa energijom,
- M – polje za održavanje.



Slika 7.6 Prikaz simulacije eksperimenta

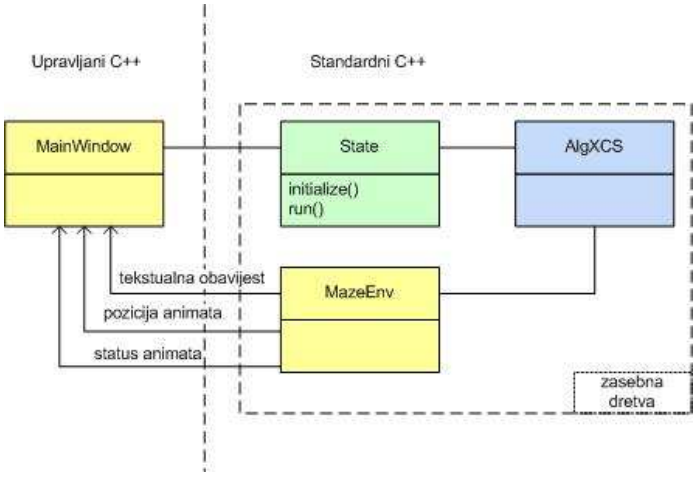
Osvježavanje liste s labirintima moguće je pomoću gumba *Refresh list*. Nakon pokretanja eksperimenta započinje simulacija kretanja animata u skladu sa akcijama koje daje sustav, a na području za poruke ispisuju se vrijednosti nagrada, broj koraka koliko je bilo potrebno za ostvarenje cilja te ostale informacije. Tablica sa prikazom populacije osvježava se svaki određen broj provjera i prikazuje važnije komponente klasifikatora.

Na trećoj kartici su parametri sustava (slika 7.7), koji se pritiskom na gumb *Save* pohranjuju u konfiguracijsku datoteku i specifični su za svaku vrstu problema.



Slika 7.7 Uređivanje parametara sustava

Kod pokretanja algoritma sustava stvara se zasebna dretva koja ga izvršava, kako bi se omogućio nesmetani rad grafičkog sučelja za vrijeme izvršavanja eksperimenta. Za prikaz informacija o stanju eksperimenta potrebno je bilo ostvariti komunikaciju između sučelja (komponenta *MainWindow*) i komponente *MazeEnv* koja predstavlja roditeljski razred za sve četiri vrste problema. Važno je uzeti u obzir da se ti dijelovi izvršavaju u zasebnim dretvama te da su pisani u različitim jezicima (slika 7.8).



Slika 7.8 Komunikacija grafičkog sučelja i XCS komponente

Komunikacija je ostvarena korištenjem funkcija povratnog poziva (eng. *callback function*). *MainWindow* kod pokretanja algoritma razredu *MazeEnv* šalje pokazivače na odgovarajuće vlastite metode koje se pozivaju u odgovarajućim trenucima eksperimenta. Na slici 7.9 je isječak koda kojim se ostvaruje navedeno povezivanje.

```
System::IntPtr d2 =
System::Runtime::InteropServices::Marshal::GetFunctionPointerForDelegate
(delegateChangePos);
mazeEnv->changePositionEvent =
      ( void(__stdcall *)(std::pair<int, int>))d2.ToPointer();
```

Slika 7.9 Povezivanje sučelja sa okolinom pomoću funkcije povratnog poziva

U dijelu sa parametrima podešavaju se parametri XCS sustava, početne vrijednosti parametara klasifikatora te ECF postavke. Pomoću ECF postavki moguće je odrediti broj provjera koje sadrži eksperiment, nakon koliko provjera se osvježava prikaz populacije te veličina početne populacije.

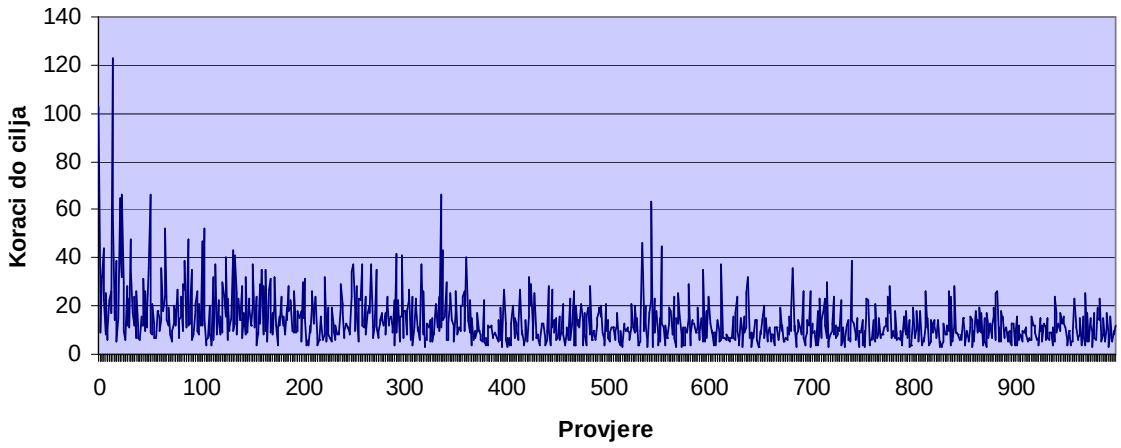
8. Eksperimentalni rezultati

Kod problema labirinta mjeri se broj koraka algoritma potrebnih za ostvarenje cilja. Za slijedni višekriterijski problem to je pronalazak hrane nakon sakupljenog ključa, a kod paralelnog, dolazak na bilo koje ciljno polje (energija, održavanje ili hrana). U sljedeća dva poglavlja prikazano je više načina na koje možemo mjeriti uspješnost sustava i dana je usporedba dobivenih rezultata sa rezultatima postojećeg XCS sustava.

8.1 Načini mjerenja

Za svaku poziciju u labirintu moguće je odrediti najmanji broj pomaka do ostvarenja cilja s najvećom nagradom. Pomoću tog podatka od svih slobodnih polja, moguće je izračunati optimalan prosječni broj koraka za pojedini labirint. Cilj je da sustav nakon određenog broja pokušaja dostigne tu vrijednost i daljnjim izvršavanjem na njoj se zadrži.

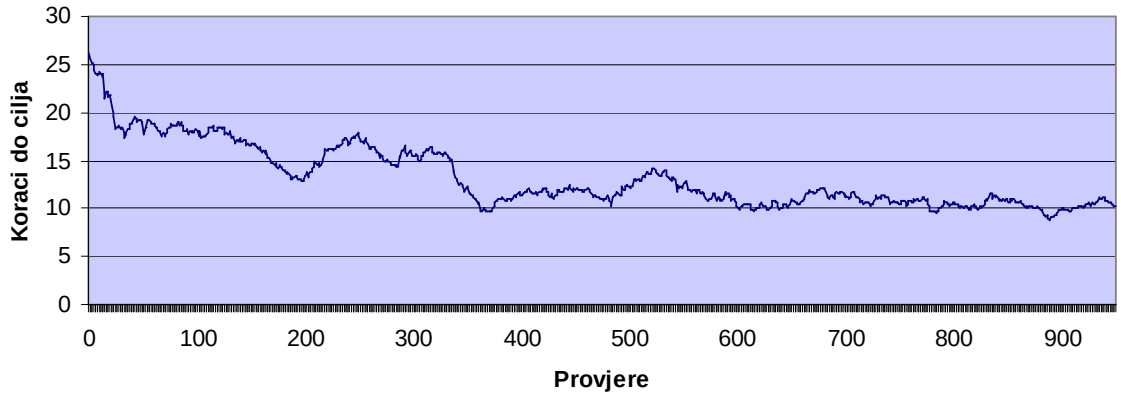
Najjednostavniji način mjerenja je da nakon svakog ostvarenog cilja bilježimo broj potrebnih koraka. Međutim, budući da svaka provjera započinje sa nasumično odabranog polja, iz prikaza takvog mjerenja teško je zaključiti stvarni napredak prema optimalnoj vrijednosti što se vidi iz slike 8.1.



Slika 8.1 Maze1k - mjerenje broja koraka do cilja

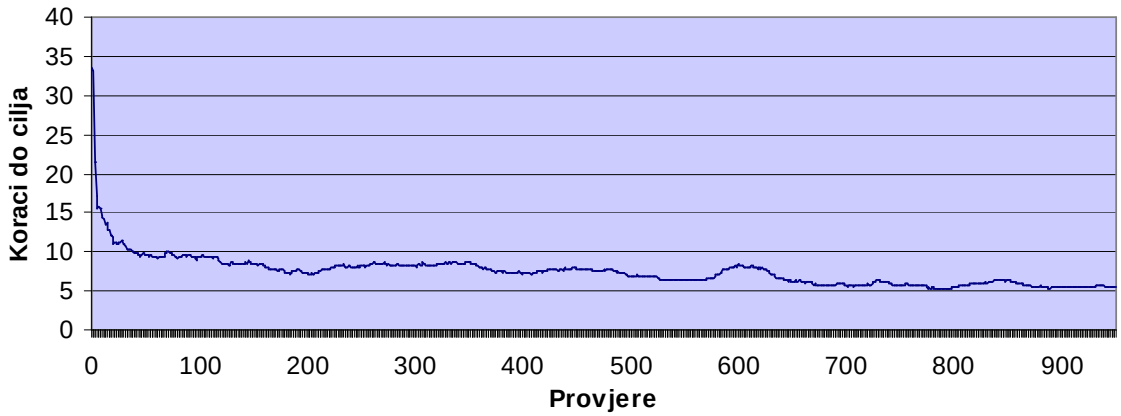
Poboljšanje je moguće ukoliko se uzima pomična prosječna vrijednost prethodnih mjerenja za određen broj koraka (slika 8.2). Krivulja performansi se izglađuje, međutim još postoje primjetne oscilacije. Uzrok tome je u načinu rada sustava.

Budući da se za odabir akcije koristi ε-greedy mehanizam, sustav povremeno odabire akcije koje nisu optimalne prema vrijednosti u polju PA. Iz tog razloga potrebno je razdvojiti provjere koje će služiti za učenje, od onih na kojima se sustav ispituje i kada se mjere performanse.



Slika 8.2 Maze1k - prosječna vrijednost uz korak od 50 provjera

Kod ispitivanja sustav odabire akcije s najvećom vrijednosti u polju PA. Također u toj fazi ne pokreće se GA komponenta i ne obnavljaju se parametri u posljednjem koraku provjere. U svim ostalim koracima obnavljaju se parametri kako bi sustav izašao iz petlji do kojih može doći u ranoj fazi ispitivanja [11]. Rad sustava proširuje se na način da nakon svake uobičajene provjere slijedi ispitivanje iz iste pozicije na kojoj je animat bio u prethodnoj provjeri. Tako su uklonjene oscilacije u mjerenjima, a uz primjenu pomične prosječne vrijednosti dobiva se krivulja performansi prikazana na slici 7.3.

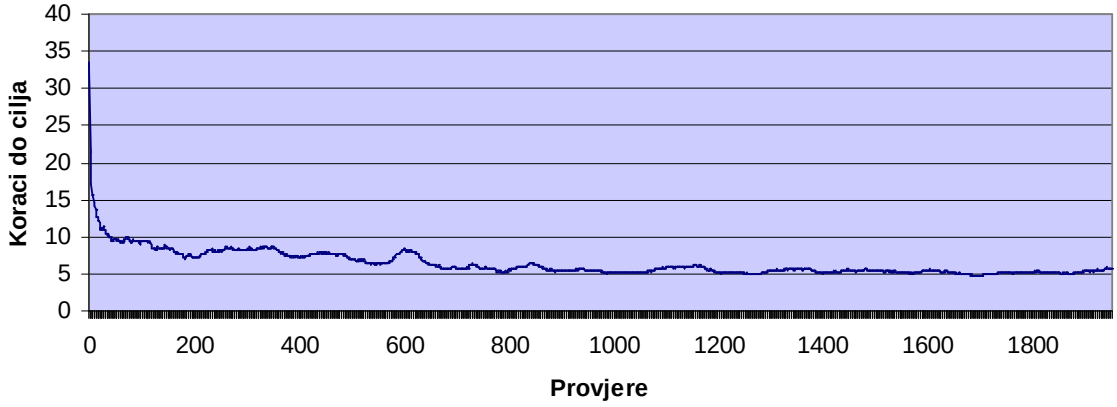


Slika 8.3 Maze1k - prosječna vrijednost uz strategiju iskorištavanja

8.2 Rezultati mjerenja

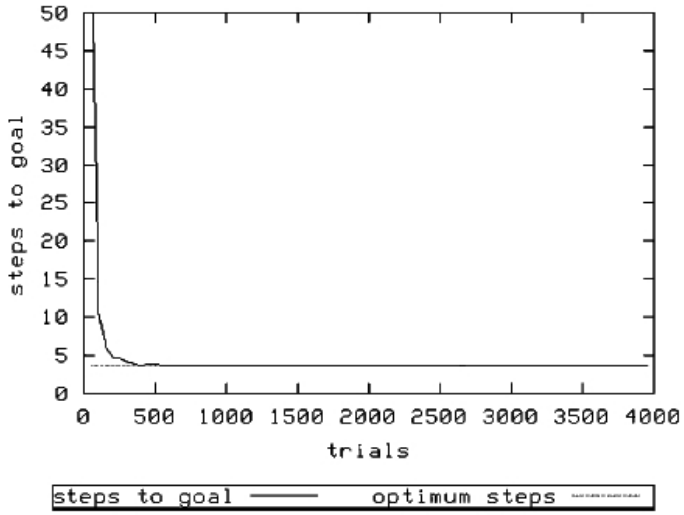
Parametri sustava korišteni za rješavanje problema su [7]: $\alpha = 0.1$, $\beta = 0.2$, $\gamma = 0.71$, $\delta = 0.1$, $p_{explore} = 0.5$, $\theta_{GA} = 20$, $\theta_{del} = 20$, $\theta_{sub} = 20$, $\varepsilon_0 = 0.01$, $\nu = 5$, $p_{\#} = 0.33$, $N = 1600$, $\theta_{mna} = 8$, $\mu = 0.01$, $\chi = 0.8$.

Početne vrijednosti parametara klasifikatora: $p_I = 10$, $\varepsilon_I = 0$, $F_I = 10$, $as_I = 0$, $ts_I = 0$, $n_I = 0$, $exp_I = 0$. Početna populacija je prazna ($N_I = 0$).

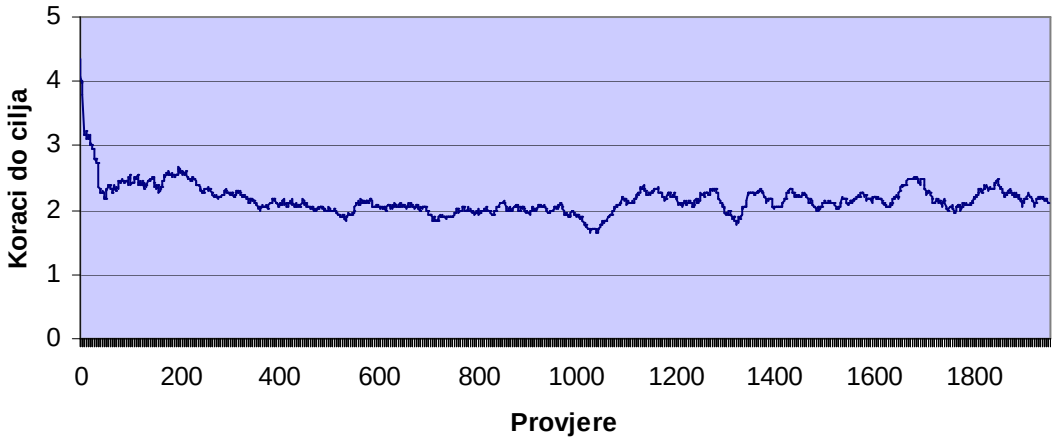


Slika 8.4 Rezultati za okolinu Maze1k

Svako mjerenje sastoji se od 2000 ispitnih provjera. Iako je njihov broj u usporedbi sa nekim mjerenjima postojećeg XCS sustava dosta manji, rezultati su zadovoljavajući.



Slika 8.5 Rezultati postojećeg sustava za okolinu Maze1k [7]

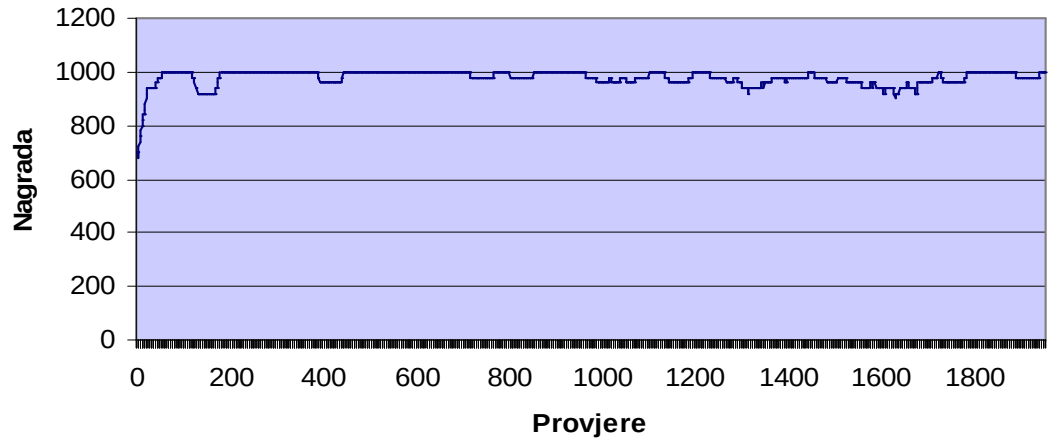


Slika 8.6 Rezultati koraka do cilja za okolinu Maze1e

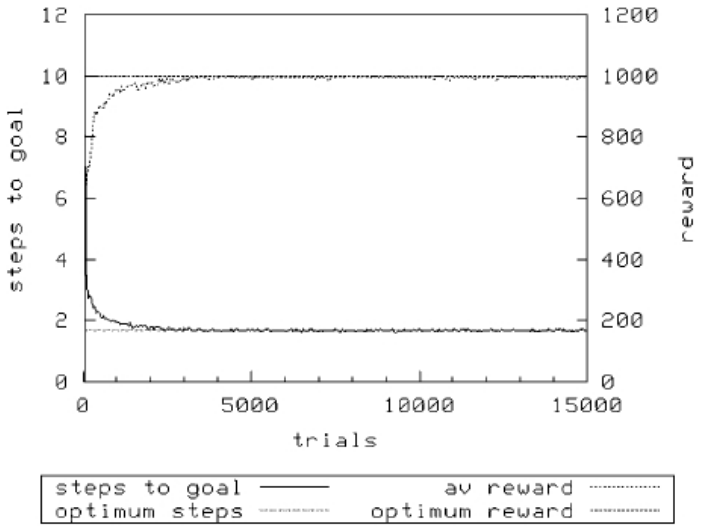
Broj koraka do cilja približava se optimalnim prosječnim vrijednostima koje su sljedeće za pojedine okoline [7]:

- *Maze1k* – 3.7
- *Maze1e* – 1.7
- *Maze1em* – 1.8

Optimalna prosječna vrijednost nagrade za svaku okolinu je 1000.

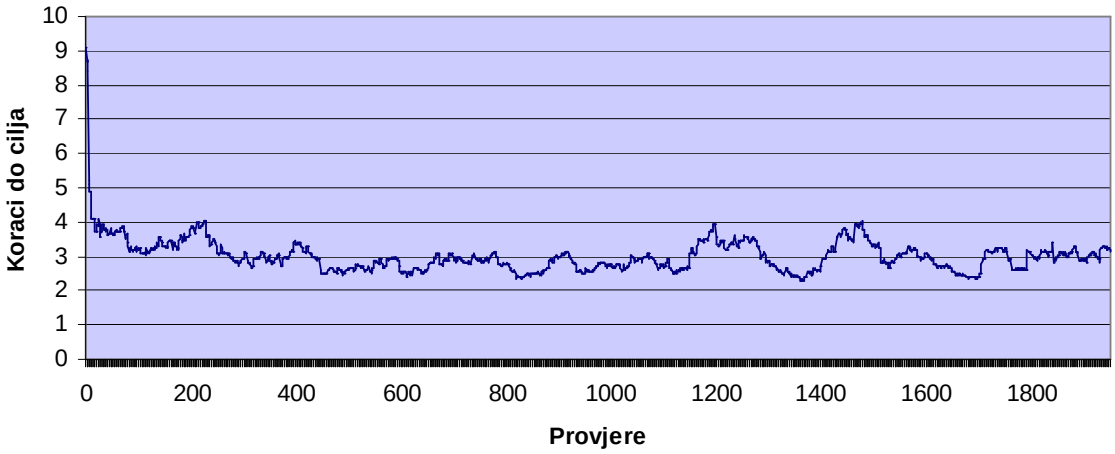


Slika 8.7 Rezultati dobivene nagrade za okolinu Maze1e

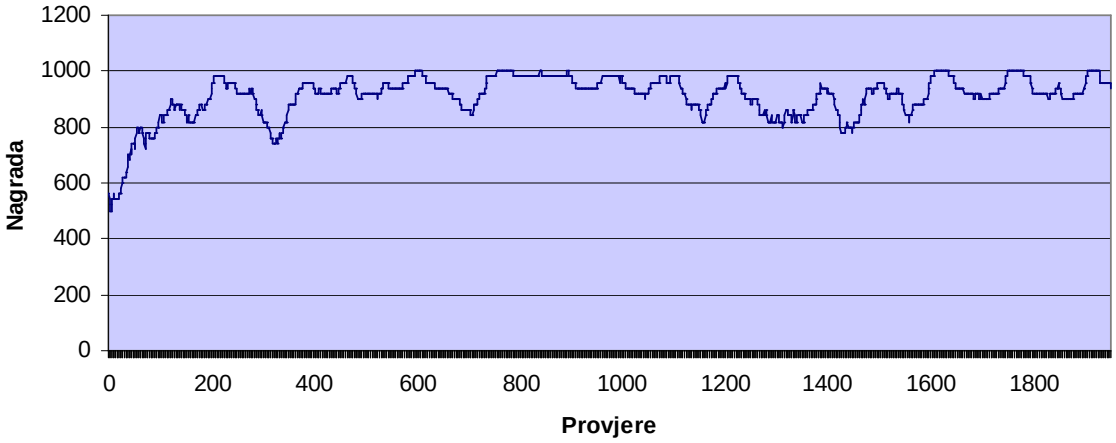


Slika 8.8 Rezultati postojećeg sustava za okolinu Maze1e [7]

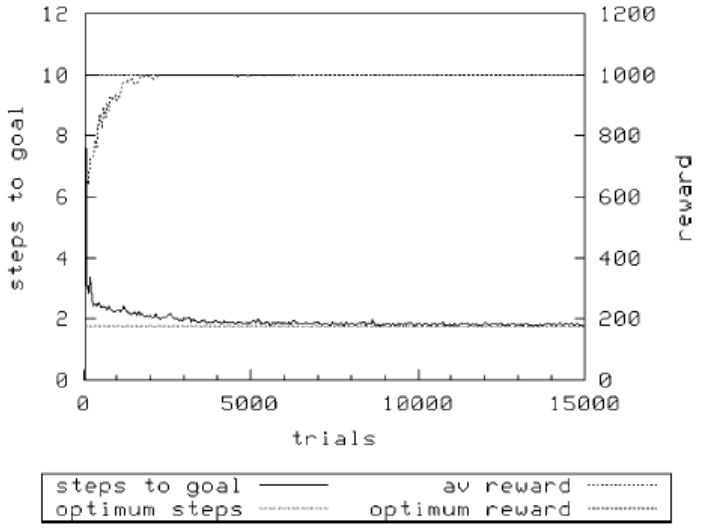
Kod okolina *Maze1e* i *Maze1em* vidljive su veće oscilacije u krivulji rezultata, što se može objasniti složenijim uvjetima za ostvarivanje cilja. Pri tome su oscilacije kod Maze1em okoline veći od onih kod Maze1e, zbog tri paralelna cilja koja je potrebno na određeni način zadovoljiti. U toj okolini animat lako može donijeti pogrešnu odluku što je vidljivo sa slika 8.9 i 8.10.



Slika 8.9 Rezultati koraka do cilja za okolinu Maze1em



Slika 8.10 Rezultati dobivene nagrade za okolinu Maze1em



Slika 8.11 Rezultati postojećeg sustava za okolinu Maze1em [7]

9. Zaključak

Klasifikatorski sustavi s mogućnošću učenja učinkovito sjedinjuju tehnike podržanog učenja i evolucijskog računanja pri rješavanju više vrsta problema. XCS sustav kao jedan od njihovih predstavnika uspješno savladava problem pretjerane generalizacije koja se javlja kod sustava baziranih na snazi, što je jedan od razloga da u zadnje vrijeme dobiva više pažnje. Također, jednostavan način komunikacije između okoline i sustava omogućava da se istom implementacijom sustava rješava više različitih problema uz odgovarajuće podešavanje parametara. XCS komponenta unutar ECF okruženja to omogućuje jednostavnim razvojem komponente okoline za svaki pojedini problem bez modifikacije sustava.

Primjena sustava na višekriterijski problem labirinta pokazuje zadovoljavajuće rezultate, gdje se performanse blizu optimalnih postižu dosta brzo. Slijedni višekriterijski problem se pokazao nešto lakšim od paralelnog što je u skladu sa rezultatima postojećeg XCS sustava. Uz simulacijsku okolinu lakše se prikazuje rad sustava, a način na koji je ostvareno povezivanje između dva različita jezika može se iskoristiti za razvoj drugih programa koji u nekom dijelu koriste ECF okruženje za rješavanje problema.

10. Literatura

[1] Sigaud O., Wilson S. W., *Learning Classifier Systems: A Survey*, Soft Computing - A Fusion of Foundations, Methodologies and Applications, svezak 11, broj 11, listopad 2007., str. 1065-1078

[2] Bull L., Applications of learning classifier systems: *Learning Classifier Systems: A Brief Introduction*, 1. izdanje, Berlin: Springer, svibanj 2004.

[3] Wilson S. W., *Classifier Systems and the Animat Problem*, Machine Learning, svezak 2, broj 3, studeni 1987., str. 199-228

[4] Bacardit J., Bernad`o-Mansilla E., Butz M. V., *Learning Classifier Systems: Looking Back and Glimpsing Ahead*, Lecture Notes In Artificial Intelligence, Berlin: Springer-Verlag, 2008.

[5] Butz M. V., *Rule-Based Evolutionary Online Learning Systems*, 1. izdanje, Berlin: Springer, listopad 2005.

[6] Butz M. V., Wilson S. W., Advances in Learning Classifier Systems : *An algorithmic Description of XCS*, Berlin: Springer, siječanj 2001.

[7] Studley M., Bull L., *Using the XCS Classifier System for Multi-objective Reinforcement Learning Problems*, Artificial Life, SAD: MIT Press, svezak 13, broj 1, 2007.

[8] Palade V., Howlett R. J., Jain L., *Knowledge-Based Intelligent Information and Engineering Systems*, 1. izdanje, Berlin: Springer, kolovoz 2003.

[9] M. Golub, *Genetski algoritam – Prvi dio*, rujan 2004.

[10] Sutton R. S., Barto A. G., *Reinforcement learning: an introduction*, SAD: MIT Press, ožujak 1998.

[11] Wilson S. W., *Classifier Fitness Based on Accuracy*, Evolutionary Computation, svezak 3, broj 2, USA: MIT Press, 1995. , str. 149-175

[12] Wilson S. W., *ZCS: A Zeroth Level Classifier System*, Evolutionary Computation, svezak 2, broj 1, USA: MIT Press, 1994., str. 1-18

[13] Bernad`o-Mansilla E., Llor`a X., Traus I., *Multiobjective Learning Classifier Systems: An Overview*, IlliGAL Report No. 2005020, lipanj 2005.

[14] Antony Robert: *EMuds: Adaptation in Text-Based Virtual Worlds*, doktorski rad, Institute of Informatics, University of Fribourg (Švicarska), veljača 2000.

[15] Urbanowicz R. J., Moore J. H.: *Learning Classifier Systems: A Complete Introduction, Review, and Roadmap*, Journal of Artificial Evolution and Applications, SAD: Hindawi Publishing Corp, 2009.

Dodatak A: Parametri XCS sustava

Oznaka	ECF parametar	Opis
N	maxPopSize	maksimalna veličina populacije
β	beta	stopa učenja (eng. *learning rate*)
α, ε_0, ν	alpha, eps0, accExp	parametri korišteni za izračunavanje dobrote
γ	gama	faktor odbitka (eng. *discount factor*)
θ_{GA}	thresholdGA	prag tolerancije za primjenu GA
χ	pCrossover	vjerojatnost križanja
μ	pMutation	vjerojatnost mutacije
θ_{del}	thresholdDel	prag tolerancije za izbacivanje iz populacije
θ_{sub}	thresholdSub	prag tolerancije za mehanizam obuhvaćanja
$p_{\#}$	pDontCare	vjerojatnost postavljanja slobodnog simbola kod cover mehanizma
$p_{explore}$	pExplore	vjerojatnost odabira slučajne akcije
θ_{mna}	mna	minimalan broj akcija u [A]
δ	delta	granica koja označava dio srednje dobrote populacije ispod koje se dobrota klasifikatora može uzeti u obzir pri izračunavanju glasa za izbacivanje

Sažetak

Klasifikatorski sustavi s mogućnošću učenja, vrsta su sustava temeljenih na pravilima koji sjedinjuju tehnike podržanog učenja i evolucijskog računanja. XCS sustav kao njihov najpoznatiji predstavnik, koristi tehniku Q-učenja i genetičkog algoritma za razvoj pravila koja uspješno generaliziraju prostor problema. Osim kod jednostavnih jednokriterijskih problema, XCS se uspješno primjenjuje kod slijednih i paralelnih višekriterijskih problema. Rad daje pregled LCS sustava i detaljniji opis XCS sustava primijenjenog na rješavanje višekriterijskog problema labirinta. Sustav je ostvaren u sklopu ECF okruženja uz dodatnu simulacijsku okolinu s grafičkim sučeljem.

Ključne riječi: klasifikatorski sustavi s mogućnošću učenja, klasifikatorski sustav XCS, podržano učenje, genetički algoritam, višekoračni problemi, višekriterijski problemi

Abstract

Learning classifier systems are rule-based systems that combine reinforcement learning methods and evolutionary computation techniques. XCS system as their best known representative uses Q-learning method and genetic algorithm for developing rules that appropriately generalize problem space. Apart from simple single-objective problems, XCS can be used for sequential and concurrent multi-objective problems. This paper provides insights in LCS systems and detail description of XCS system used for solving multi-objective maze problem. System is implemented as a par of ECF framework with additional graphical user interface.

Key words: learning classifier systems, XCS classifier system, reinforcement learning, genetic algorithm, multi-step problems, multi-objective problems