Generic model for experimenting and using a family of classifiers systems: description and basic applications.

Cédric Buche, Pierre de Loor

Generic Model for Experimenting and Using a Family of Classifiers Systems: Description and Basic Applications

Cédric Buche and Pierre De Loor

UEB / ENIB / LISyC
European center for virtual reality
Technopôle Brest-Iroise F-29280 Plouzané, France
{buche,deloor}@enib.fr

Abstract. Classifiers systems are tools adapted to learn interactions between autonomous agents and their environments. However, there are many kinds of classifiers systems which differ in subtle technical ways. This article presents a generic model (called GEMEAU) that is common to the major kinds of classifiers systems. GEMEAU was developed for different simple applications which are also described.

1 Introduction

Defining the behavior of autonomous artificial entities is faced with the problem of selecting a model able to account for the link between perceptions and actions in an efficient manner. There are a great number of proposed solutions to this issue. However, they require detailed descriptions which are difficult to achieve, either because they require a definition based on *a priori* rules and symbols [2,8], or because they are subject to configuration difficulties and behavioral indeterminism [1,7]. Another solution would be to define the entities' initial approximate behavior, which would then adapt according to its environment. This solution is implemented by classifiers systems. It uses a set of competing rules and incorporates learning processes by choosing and improving these rules. A great deal of literature exists on the subject [3,12,6,13,11]. A number of authors have put forward different variations of the approach, each offering different mechanisms adapted to specific problems. Our objective is to be able to test and advance these mechanisms without difficulty, consequently we are interested in designing and implementing a generic model.

This article is organized as following: first we present the general mechanisms of classifiers system. We then go on to present a generic model, called GEMEAU[1], which integrates these mechanisms, and with which we can easily test different versions. Next we explain how we applied this model to different types of applications: multiplexers and *woods* environments.

[1] **GEMEAU: GE**neric **M**odel for **E**xperimenting **A**nd **U**sing a family of classifiers systems.

L. Rutkowski et al. (Eds.): ICAISC 2010, Part I, LNAI 6113, pp. 299–306, 2010.

2 Classifiers Systems

2.1 Principles

A classifiers system is an adaptive system that learns to perform the best action given its input. The system manage a combination of "condition-action" rules called classifiers, pondered by quality parameters. The system perceives its environment (usually a vector of numerical values), deduces the applicable rules, carries out an action as a result of these rules. Technically, when a particular input occurs, classifiers whose conditions are satisfied by that input are selected and the system chooses one of the possible actions (from selected rules) according to the quality parameters. The system is able to receive a reward from the environment which it then uses to modify the rules or their quality parameters.

Through experimentation, classifiers system can therefore be used to learn the association between conditions and actions, thus maximizing credit intake. In order to avoid a combinatorial explosion of the quantity of rules, they are generalized; they apply to different perceptions of the environment. Mechanisms which allow the creation, enrichment (specialization/generalization), or destruction of these rules must therefore be used. Evolutionary algorithms are often used to do this, even though other heuristic approaches are available. The qualities of the rules are modified dynamically through reinforcement learning, and the rules themselves are modified by genetic algorithms.

2.2 Formalization

In this section we propose the incremental and generic formalization of classifiers systems, and gradually introduce learning mechanisms.

The global structure of a classifier system, is a 7-uplet
(`Ii`, `[P]`, `[M]`, `[A]`, `Matching`, `Selection`, `Io`) :

- `Ii` is the interface input due to which each `Perception` within the environment corresponds to a binary code.
- `[P]`, (*population*), is the set of the system's classifiers, coded by a succession of *n* bits[2]. The generalizing representations contain $\#$ symbols which correspond to an indeterminate value. A rule is a (C, A) pair with $C \cup A \in \{0, 1, \#\}^n$ where :
  - C : the condition for application of the rule.
  - A : the action(s) associated with the application of the rule.

  Let us take the example of a robot with four 'all-or-nothing' sensors and one action. The input interface converts the state of the sensors into a binary value and the output interface triggers the action depending on the action's bit value. Thus, a $\{011\#, 1\}$ rule means that the rule is applicable if the first sensor is inactive and the two following sensors active. The state of the fourth sensor has no influence, and applying the rule triggers the action.
- `[M]` $\subseteq$ `[P]` is the set of classifiers of which the condition element pairs with the perceived environmental information during a selection cycle. This is known as *Match-set*.

[2] Even if certain systems work with other alphabets [9,17,5].

- `[A]` $\subseteq$ `[M]` is the set of classifiers representing the selected action. This is known as *Action-set*.
- `Matching` is the mechanism which makes the transition from `[P]` to `[M]` possible. This is generally achieved using a matching rule between C and the information derived from `Ii`. This rule is able to interpret the generalization symbols that make up the classifier conditions.
- `Selection` is the mechanism which makes the transition from `[M]` to `[A]` possible. Depending on the details of the different versions of classifiers systems, it is able to determine the desired action.
- `Io` is the output interface through which the activated `Action` corresponds to a binary code.

Learning occurs due to an evaluation of the quality of the rules represented by one or a number of additional parameters. The definition of a classifier is thus extended to a $R = (C, A, f)$ triplet where f characterizes its quality. Learning developed by `Rewarding` the rules, by altering their quality using reinforcement learning algorithms and by `Generating` rules using evolutionary and heuristic covering algorithms. The dynamics of learning classifiers systems are therefore based on the following cycle: `Perception` / `Matching` / `Generation` *(covering)* / `Selection` / `Action` / `Reward` / `Generation` *(evolutionary algorithm)*.

`Selection` is guided by the quality of the rules, which are grouped together depending on their `[A]` element. Often, a 'wheel of fortune' mechanism[3] is applied, which means that each package has a probability proportional to its capacity to be selected. The credit assignment (`Reward`) mechanism distributes credit to the rules that have contributed to its acquisition. It increases the quality of the rules triggered prior to the acquisition of the credit and decreases that of the others. Its definition affects the relevant length of a series of actions: i.e. the number of rules in a sequence considered necessary in order to achieve a certain goal. The `generation` mechanism must both minimize the number of rules and conserve those which assist in achieving credit. A good rule is therefore a high-quality generalizing rule (relative to the others). The two generation (*rules discovery*) mechanisms used are *covering* (creation of rules when no classifiers match the perception of the environment) and *evolutionary algorithms*.

3 GEMEAU

Classical classifiers systems (ZCS, XCS, ACS, Hierachichal ...) [15,16,12,4] go some way to finding optimal solutions in Markovian or non-Markovian environments. Nevertheless, as Sanza notes [10], the improvements and supplementary systems are suitable only for specific cases and none of the models are able to supply an overall solution for all of the problems (XCS is only effective if the credits are discrete and of a fixed quantity; ACS is only useful if each action leads to a modification in the perception of the world ...).

[3] The wheel of fortune mechanism consists of picking elements randomly, so that their probability of being chosen is proportional to their selectivity.

There are, therefore, a great number of classifiers systems [14]. Developing and testing a variety of such systems take time and is not easy. Using the structure and dynamics analysis conducted previously we were able to come up with a generic background for a whole family of classifiers systems. Our architecture claims to be generic, in the sense that it can be used to implement ZCS and XCS systems (ZCS, XCS, ZCSM, XCSM).

3.1 Architecture

The architecture is displayed in Fig. 1 as a UML classes diagram. The system is called GEMEAU. It is based around two components: *interface with the environment* and *system*.

The interface with the environment determines the interactions between the system and environment, both of which are common to different classifiers systems. In our model, the different interfaces are implemented using three categories: *CS_II*, *CS_IO* and *CS_R* (respectively entry interface, output interface and credit). Communication between the interfaces and the environment takes place in the form of messages, enabling the classifiers system to have an implementation in parallel to the environment.

The System defines the elements and the mechanisms of our classifiers system in concrete terms. Let us consider the following elements:

- A classifier (*CS_Classifier*) is made up of several parts: condition (*CS_Condition*), action (*CS_Action*) and configuration (*CS_Parameter*);
- The sets `[P]`,`[M]`,`[A]` and $\texttt{[A]}_{-1}$ are lists of *CS_ClassifierList*-type classifiers.

We put forward the following mechanisms:

- The `Matching` mechanism, with which the classifiers that match the information coming from the environment can be extracted. It is included in the *CS_Classifier List* by the *match()* method;
- The `Generation` mechanism by *covering*, which creates rules according to the content of `[M]` after `Matching`. It is included in the *CS_ClassifierList* by the *cover()* method which can be configured (notably for the number of #);
- The general (*CS_System*) method represents the workings of a given cycle (*step()* method);
- The `Selection` mechanisms of the winning (*CS_SelectorAlgo*) actions (which must be able to differ according to the desired learning);
- The `Reward` mechanism, (*CS_AOCAlgo*), modifying the classifiers' configuration.
- The `Generation` genetic algoritm, (*CS_GeneticAlgo*), where different operators must be specified, i.e. crossing or mutation.

3.2 Use

GEMEAU can be specialized in order to obtain a ZCS (Fig. 1). Through inheritance, we can define:

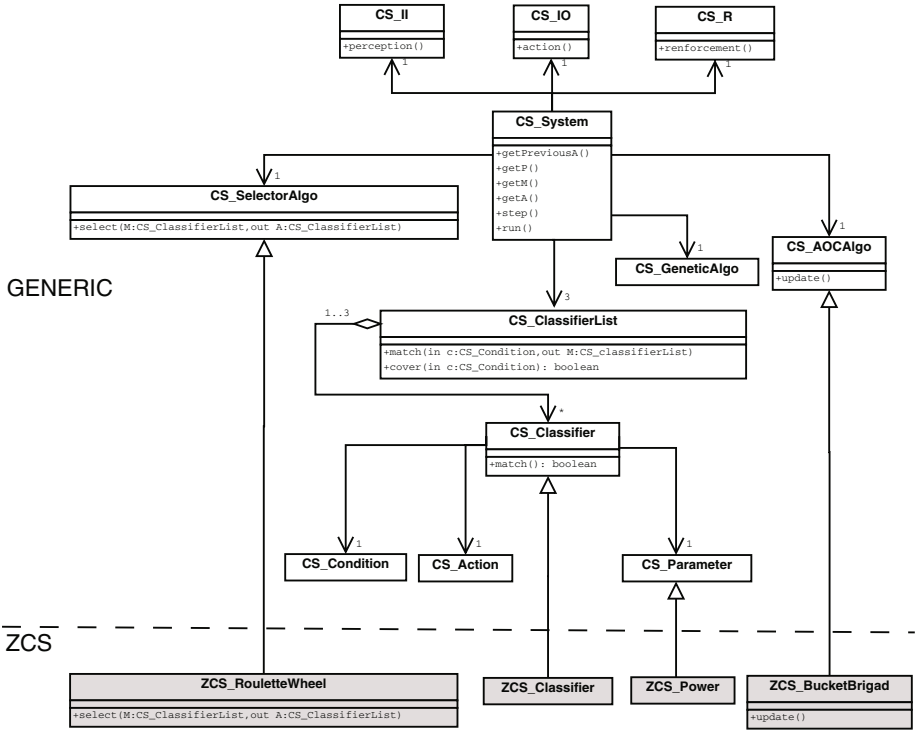

Fig. 1. UML diagram of GEMEAU augmented by a ZCS using inheritance

- The `Rules` (*ZCS_Classifier* derived from *CS_Classifier*) having a configuration force (*ZCS_Power* derived from *CS_Parameter*);
- The 'wheel of fortune' type `Selection` mechanism `ZCS_RouletteWheel` derived from *CS_SelectorAlgo*);
- The *Bucket Brigade*-type `Reward` mechanism (*ZCS_BucketBrigad* derived from *CS_AOCAlgo*);
- The `Generation` genetic algorithm (*ZCS_GeneticAlgo* derived from *ZCS_GeneticAlgo*) notably specifying that the selective value is the rule's strength.

The rest of the system uses the pre-existing default mechanisms such as *covering*. We implemented the XCS classifier system using the same techniques. In order to do so, we must overdefine *CS_Parameter*.

We can also easily add memory to ZCS in order to obtain a ZCSM [3]. In that case, we must increase the result of perception using an internal classifier system register which is modified by part of the last action to have been carried out. Using GEMEAU, the *step()* method can be redefined simply by inheriting from the *CS_System*. We implemented the XCSM classifier system using these same principles.

By using these specialization and extension mechanisms we were able to use our architecture to implement and test ZCS and XCS family classifiers systems (ZCS, ZCSM,

XCS, XCSM). Our perspectives are based on the implementation of supplementary traditional anticipatory or hierarchical systems (ACS: anticipation [12] and ALECSYS: hierarchy [4]). Implementing these systems could well be less simple that for the family of ZCS and XCS, and the architecture may need to be modified.

3.3 Validation

One simple and frequently used evaluation environment is a multiplexer. Let us consider a multiplexer with an input of 6 bits $a_0, a_1, d_0, d_1, d_2, d_3$ and an $output$ of one bit. a_0 and a_1 correspond to address bits. The multiplexer equation is $output = \overline{a_0}.\overline{a_1}.d_0 + \overline{a_0}.a_1.d_1 + a_0.\overline{a_1}.d_2 + a_0.a_1.d_3$. The output will be either the value of d_0, d_1, d_2 or d_3, depending on the address. The aim is to find this multiplexing function.

Using GEMEAU, we must simply determine the detectors and effectors that interface with the environment plus the reinforcement to be distributed, and then instantiate a *CS_System* (Fig. 2). The conditions and actions of the classifiers here correspond to the multiplexer's input and output respectively. The classifier system is rewarded when the rule selected corresponds to the multiplexing function.

```
/* Environnement */
  env = new EnvironmentMultiplexer(nbEntries,nbOut);
  detector = new CS_II_Boolean(env);
  effector = new CS_IO_Boolean(env);
  reinforcement = new CS_R(env);
/* System */
  system = new CS_System();
  system->setDetector(detector);
  system->setEffector(effector);
  system->setReinforcement(reinforcement);
  system->run();
```

Fig. 2. Use of GEMEAU for a multiplexer-type environment

Another advantageous evaluation environment is the *woods* environment. It corresponds to a graphical representation based on an infinitely repeating pattern. It represents a forest and the aim of a situated agent is to find food. Within this forest there are insurmountable obstacles (trees) and areas of unrestricted movement. The perception of the agent corresponds to a representation of the 8 squares surrounding the occupied position. The simplest of these environments is *woods1* (Fig. 3a). This is a deterministic Markovian environment[4] The *woods100* (Fig. 3b), however, is non-Markovian. Indeed the optimum displacement from square number 2 is to the right although for square number 5 it is to the left even though these two squares are perceived identically.

The system learns by alternating between an iteration in exploration mode (selecting the action using the 'wheel of fortune' mechanism) and an iteration in exploitation mode (choosing the best action). The curves only take into account the results in exploitation mode, dealing with the average of the last ten iterations.

GEMEAU can deal with this two classical examples, it converges for the multiplexer (Fig. 4a) and for *woods1* (Fig. 4b). For the multiplexer, we achieve a 100% success

[4] There are no perceptions values corresponding to different states of the agent.

Fig. 3. Markovian woods1 (a) and non-Markovian woods100 (b) environments

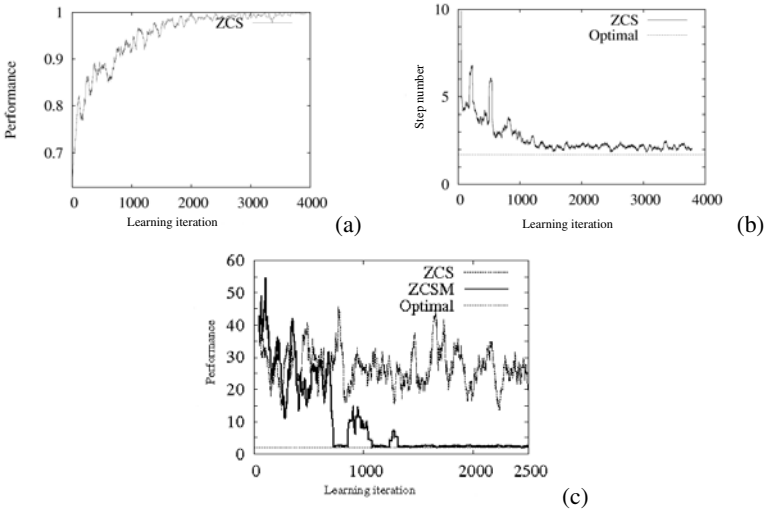

Fig. 4. ZCS multiplexer learning (a) and in woods1 (b). As of a sufficient number of iterations, our system conducts the multiplexing function and obtains the minimum number of movements in the case of woods1. ZCS and ZCSM learning in woods100 (c).

rate. For *woods1*, we achieve solutions similar to the minimum number of movements. The results are conclusive: in both cases we reported performances similar to those described in the results of [15]. Furthermore GEMEAU allows the rapid evaluation of derived classifier systems. We compared the ZCS and ZCSM results in the *woods100* non-Markovian environment (Fig. 4c). Our results rediscover the ZCS' difficulties obtaining optimal rules in non-Markovian environments. They also confirm that our architecture can be used to extend the capacities of ZCS to non-Markovian environments.

4 Conclusion

Having described existing classifiers systems, we illustrated a more general classifiers system which groups together the traditional systems. We put forward our model called GEMEAU enabling traditional systems and their variants to be both modelled and extended. This implementation is flexible enough to be used for a variety of problems as it proposes an interface between the environment and the classifier system (input/output/reinforcement). It has been used to test many types of classifiers systems and

different conceptual hypotheses quickly, as well as to obtain significant comparative results. Among other things, these tests showed us the interest of being able to access a library of classifiers systems with which we should be able to define a methodology for choosing learning algorithms based on certain stages of the tests.

References

1. Brooks, R.: Elephants don't play chess. Robotics and Autonomous Systems 6(1&2), 3–15 (1990)
2. Carver, M., Lesser, V.: The evolution of blackboard control architectures. Tech. Rep. UM-CS-1992-071, Department of Computer Science, University Massachusetts (1992)
3. Cliff, D., Ross, S.: Adding Temporary Memory to ZCS. Tech. Rep. CSRP347, School of Cognitive and Computing Sciences, University of Sussex (1995)
4. Dorigo, M.: Alecsys and the AutonoMouse: Learning to Control a Real Robot by Distributed Classifier Systems. Machine Learning 19, 209–240 (1995)
5. Heguy, O., Sanza, C., Berro, A., Duthen, Y.: GXCS: A generic classifier system and its application in a real time cooperative behavior simulations. In: International Symposium and School on Advanced Distribued System (2002)
6. Lanzi, P., Wilson, S.: Optimal classifier system performance in non-Markov environments. Tech. Rep. 99.36 (1999)
7. Maes, P.: The dynamics of action selection. In: Proceedings of the international Joint Conference on Artificial Intelligence (1989)
8. Mateas, M.: An Oz-centric review of interactive drama and believable agents. In: Veloso, M.M., Wooldridge, M.J. (eds.) Artificial Intelligence Today. LNCS (LNAI), vol. 1600, pp. 297–328. Springer, Heidelberg (1999)
9. Matteucci, M.: Fuzzy learning classifier system: Issues and architecture. Tech. Rep. 99.71 (1999)
10. Sanza, C., Heguy, O., Duthen, Y.: Evolution and cooperation of virtual entities with classifier systems. In: Eurographic Workshop on Computer Animation and Simulation (2001)
11. Sigaud, O., Wilson, W.: Learning classifier systems: A survey. Journal of Soft Computing 11(11), 1065–1078 (2007)
12. Stolzmann, W.: Anticipatory classifier systems. In: Third Annual Genetic Programming Conference, pp. 658–664. Morgan Kaufmann, San Francisco (1998)
13. Tomlinson, A., Bull, L.: A zeroth level corporate classifier system. In: Second International Workshop on Learning Classifier Systems. Springer, Heidelberg (1999)
14. Urbanowicz, R., Moore, J.: Learning classifier systems: A complete introduction, review, and roadmap. Journal of Artificial Evolution and Applications, 25 (2009)
15. Wilson, W.: ZCS: A zeroth level classifier system. Evolutionary Computation 2(1), 1–18 (1994)
16. Wilson, W.: Classifier Fitness Based on Accuracy. Evolutionary Computation 3(2), 149–175 (1995)
17. Wilson, W.: Get real! XCS with continuous-valued inputs. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 1999. LNCS (LNAI), vol. 1813, pp. 209–219. Springer, Heidelberg (2000)