Learning classifier systems: a survey

Olivier Sigaud, Stewart Wilson

Noname manuscript No.
(will be inserted by the editor)

Learning Classifier Systems: A Survey

Olivier Sigaud[1], **Stewart W. Wilson**[2]

[1] Université Pierre et Marie Curie - Paris 6
4 Place Jussieu
75252 Paris Cedex 05, France

[2] Prediction Dynamics
Concord, MA 01742, USA

Received: date / Revised version: date

Abstract Learning Classifier Systems (LCSs) are rule-based systems that automatically build their ruleset. At the origin of Holland's work, LCSs were seen as a model of the emergence of cognitive abilities thanks to adaptive mechanisms, particularly evolutionary processes. After a renewal of the field more focused on learning, LCSs are now considered as sequential decision problem-solving systems endowed with a generalization property. Indeed, from a Reinforcement Learning point of view, LCSs can be seen as learning systems building a compact representation of their problem thanks to generalization. More recently, LCSs have proved efficient at solving automatic classification tasks. The aim of the present contribution is to describe the state-of-the-art of LCSs, emphasizing recent developments, and focusing more on the sequential decision domain than on automatic classification.

Key words Learning Classifier Systems, Reinforcement Learning, Generalization

1 Introduction

All Learning Classifier Systems (LCSs) [1] have in common that they are rule-based systems able to automatically build the ruleset they manipulate. Invented in 1975 by John Holland (Holland, 1975), these systems are paradoxically less famous than Genetic Algorithms (GAs) though GAs were originally a sub-part of LCSs. However, during the past several years LCS research has gained more visibility, giving

Correspondence to: Olivier.Sigaud@lip6.fr,wilson@prediction-dynamics.com

[1] The term *Learning* was added around 2000 so as to clearly distinguish this research line from other, unrelated, systems that classify.

rise to the opportunity of publishing a general presentation for a wide scientific audience. The goal of this paper is to offer an overview of the fundamental aspects of LCSs and of the recent developments they are giving rise to.

In order to reach that goal, we first present the two mechanisms on which they rely, namely GAs and Reinforcement Learning (RL). Then we provide a brief history of LCS research intended to highlight the emergence of three families of systems: strength-based LCSs, accuracy-based LCSs, and anticipatory LCSs (ALCSs). Afterward, in section 4, we present everything that is common to all systems of these three families, from their representation formalism to their fundamental mechanisms. The next three sections are dedicated to the particular aspects of each family, focusing particularly on the most recent theoretical and applied extensions. We devote particular effort to the accuracy-based family whose main member, XCS, is the most studied LCS at this time. Finally, we try to highlight what seem to be the most promising lines of research given the current state of the art, and we conclude with the available resources that can be consulted in order to get a more detailed knowledge of these systems.

2 Background

2.1 Genetic Algorithms

First, we briefly present GAs (Holland, 1975; Booker et al., 1989; Goldberg, 1989), which are freely inspired from the neo-darwinist theory of natural selection. These algorithms manipulate a population of individuals representing possible solutions to a given problem. GAs rely on four analogies with their biological counterpart: they use a code, the *genotype* or *genome*, simple transformations operating on that code, the *genetic operators*, the expression of a solution from the code, the *genotype-to-phenotype mapping*, and a solution selection process, the *survival of the fittest*. The genetic operators are used to introduce some variations in the genotypes. There are two classes of operators: crossover operators, which create new genotypes by recombining sub-parts of the genotypes of two or more individuals, and mutation operators, which randomly modify the genotype of an individual. The selection process extracts the genotypes that deserve to be reproduced, upon which genetic operators will be applied.

A GA manipulates a set of arbitrarily initialized genotypes which are selected and modified generation after generation. Those which are not selected are eliminated. A utility function, or fitness function, evaluates the interest of a phenotype with regard to a given problem. The survival of the corresponding solution or its number of offspring in the next generation depends on this evaluation.

The offspring of an individual are built from copies of its genotype to which genetic operators are applied. As a result, the overall process consists in the iteration of the following loop:

1. select n_e genotypes according to the fitness of corresponding phenotypes,
2. apply genetic operators to these genotypes to generate offspring,
3. build phenotypes from these new genotypes and evaluate them,

4. go to 1.

If some empirical conditions that we will not detail here are fulfilled, such a process gives rise to an improvement of the fitnesses of the individuals over the generations.

Since research on GAs is now a field in itself, we will not survey it in this paper. Though GAs are at their root, LCSs have made limited use of the important extensions of this field. As a consequence, in order to introduce the GAs used in LCSs, it is only necessary to describe the following aspects:

- One must classically distinguish between the *one-point crossover* operator, which cuts two genotypes into two parts at a randomly selected place and builds a new genotype by inverting the sub-parts from distinct parents, and the *multi-point crossover* operator, which does the same after cutting the parent genotypes into several pieces. Historically, most early LCSs were using the one-point crossover operator. Recently, a surge of interest on the discovery of complex 'building blocks' in the structure of input data led to a more frequent use of multi-point crossover.
- One must also distinguish between *generational* GAs, where all or an important part of the population is renewed from one generation to the next, and *steady state* GAs, where individuals are changed in the population one by one without notion of generation. Most LCSs use a steady-state GA, since this less disruptive mechanism results in a better interplay between the evolutionary process and the learning process, as explained below.

2.2 Markov Decision Processes and Reinforcement Learning

The second fundamental mechanism in LCSs is RL. In order to describe this mechanism, it is necessary to briefly present the Markov Decision Process (MDP) framework and the Q-LEARNING algorithm, which is now the learning algorithm most used in LCSs. This presentation is as succinct as possible; the reader who wants to get a deeper view is referred to Sutton and Barto (1998).

2.2.1 Markov Decision Processes A MDP is defined as the collection of the following elements:

- a finite set S of discrete states s of an agent ;
- a finite set A of discrete actions a ;
- a transition function $P : S \times A \rightarrow \Pi(S)$ where $\Pi(S)$ is the set of probability distributions over S. A particular probability distribution $Pr(s_{t+1}|s_t, a_t)$ indicates the probabilities that the agent reaches the different s_{t+1} possible states when he performs action a_t in state s_t ;
- a reward function $R : S \times A \rightarrow \mathbb{R}$ which gives for each (s_t, a_t) pair the scalar reward signal that the agent receives when he performs action a_t in state s_t .

The MDP formalism describes the stochastic structure of a problem faced by an agent, and does not tell anything about the behavior of this agent in its environment. It only tells what, depending on its current state and action, will be its future situation and reward.

The above definition of the transition function implies a specific assumption about the nature of the state of the agent. This assumption, known as the *Markov property*, stipulates that the probability distribution specifying the s_{t+1} state only depends on s_t and a_t, but not on the past of the agent. Thus $P(s_{t+1}|s_t, a_t) = P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0)$. This means that, when the Markov property holds, a knowledge of the past of the agent does not bring any further information on its next state.

The behavior of the agent is described by a policy π giving for each state the probability distribution of the choice of all possible actions.

When the transition and reward functions are known in advance, *Dynamic Programming* (DP) methods such as *policy iteration* (Bellman, 1961; Puterman and Shin, 1978) and *value iteration* (Bellman, 1957) efficiently find a policy maximizing the accumulated reward that the agent can get out of its behavior.

In order to define the accumulated reward, we introduce the *discount factor* $\gamma \in [0, 1]$. This factor defines how much the future rewards are taken into account in the computation of the accumulated reward at time t as follows:

$$Rc_\pi(t) = \sum_{k=t}^{T_{max}} \gamma^{(k-t)} r_\pi(k)$$

where T_{max} can be finite or infinite and $r_\pi(k)$ represents the immediate reward received at time k if the agent follows policy π.

DP methods introduce a *value function* V^π where $V^\pi(s)$ represents for each state s the accumulated reward that the agent can expect if it follows policy π from state s. If the Markov property holds, V^π is solution of the Bellman equation (Bertsekas, 1995):

$$\forall s \in S, V^\pi(s) = \sum_a \pi(s_t, a_t)[R(s_t, a_t) + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t)V^\pi(s_{t+1})] \quad (1)$$

Rather than the value function V^π, it is often useful to introduce an action-value function Q^π where $Q^\pi(s, a)$ represents the accumulated reward that the agent can expect if it follows policy π after having done action a in state s. Everything that was said of V^π directly applies to Q^π, given that $V^\pi(s) = \max_a Q^\pi(s, a)$. The corresponding optimal functions are independent of the policy of the agent; they are denoted V^* and Q^*.

2.2.2 Reinforcement Learning Learning becomes necessary when the transition and reward functions are not known in advance. In such a case, the agent must explore the outcome of each action in each situation, looking for the (s_t, a_t) pairs that bring it a high reward.

The main RL methods consist in trying to estimate V^* or Q^* iteratively from the trials of the agent in its environment. All these methods rely on a general approximation technique in order to estimate the average of a stochastic signal received at each time step without storing any information from the past of the

agent. Let us consider the case of the average immediate reward. Its exact value after k iterations is

$$E_k(s) = (r_1 + r_2 + \cdots + r_k)/k$$

Furthermore,

$$E_{k+1}(s) = (r_1 + r_2 + \cdots + r_k + r_{k+1})/(k+1)$$

thus

$$E_{k+1}(s) = k/(k+1)E_k(s) + r_{k+1}/(k+1)$$

which can be rewritten:

$$E_{k+1}(s) = (k+1)/(k+1)E_k(s) - E_k(s)/(k+1) + r_{k+1}/(k+1)$$

or

$$E_{k+1}(s) = E_k(s) + 1/(k+1)[r_{k+1} - E_k(s)]$$

.

Formulated that way, we can compute the exact average by merely storing k. If we do not want to store even k, we can approximate $1/(k+1)$ with α, which results in equation (2) whose general form is found everywhere in RL:

$$E_{k+1}(s) = E_k(s) + \alpha[r_{k+1} - E_k(s)] \tag{2}$$

The parameter α, called *learning rate*, must be tuned adequately because it influences the speed of convergence towards the exact average.

We do not detail all of the RL method relying on this estimation principle. We only give the update equation of the Q-LEARNING algorithm, which is the following:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{3}$$

3 Brief History of LCS

3.1 Pittsburgh versus Michigan

LCSs were invented by Holland (Holland, 1975) in order to model the emergence of cognition based on adaptive mechanisms. They consist of a set of rules called *classifiers* combined with adaptive mechanisms in charge of evolving the population of rules. The initial goal was to solve problems of interaction with an environment such as the one presented in figure 1, as was described by Wilson as the "Animat problem" (Wilson, 1985).

In the context of the initial research on LCSs, the emphasis was put on parallelism in the architecture and evolutionary processes that let it adapt at any time to the variations of the environment (Golberg and Holland, 1988). This approach was seen as a way of "escaping brittleness" (Holland, 1986) in reference to the lack of robustness of traditional artificial intelligence systems faced with problems more complex than toy or closed-world problems.

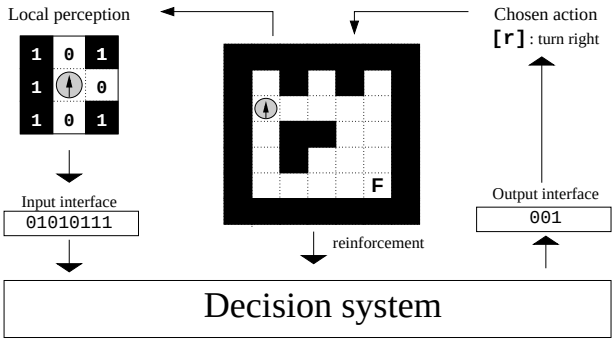

Fig. 1 Representation of an interaction problem. The agent senses a situation as a set of attributes. In this example, it is situated in a maze and senses either the presence (symbol 1) or the absence (symbol 0) of walls in the eight surrounding cells, considered clockwise starting from the north. Thus, in the above example it senses `[01010111]`. This information is sent to its input interface. At each time step, the agent must choose between going forward `[f]`, turning right `[r]` or left `[l]`. The chosen action is sent through the output interface.

This period of research on LCSs was structured by the controversy between the so-called "Pittsburgh" and "Michigan" approaches. In Smith's approach (Smith, 1980), from the University of Pittsburgh, the only adaptive process was a GA applied to a population of LCSs in order to choose from among this population the fittest LCS for a given problem.

By contrast, in the systems from Holland and his PhD students, at the University of Michigan, the GA was combined since the very beginning with an RL mechanism and was applied more subtly within a single LCS, the population being represented by the set of classifiers in this system.

Though the Pittsburgh approach is becoming more popular again currently, (Llorà and Garrell, 2002; Bacardit and Garrell, 2003; Landau et al., 2005), the Michigan approach quickly became the standard LCS framework, the Pittsburgh approach becoming absorbed into the wider evolutionary computation research domain.

The first concrete implementation of Michigan style LCS, called "CS1", was published by Holland and Reitman (Holland and Reitman, 1978). This first, rather complex model is described in figure 2. The GA creates new classifiers from the existing ones. The fitness of a classifier is measured by its capacity to propose an efficient action in adequate situations. This efficacy itself is evaluated by a RL algorithm called BUCKET BRIGADE (Holland et al., 1986).

The presence of an internal message list can emulate memory mechanisms when a message is kept on the list over several time steps. But some of the mechanisms have proved difficult to design, among them the internal message suppression process and the interactions between classifier evaluation and the evolution of the population (Wilson and Goldberg, 1989).

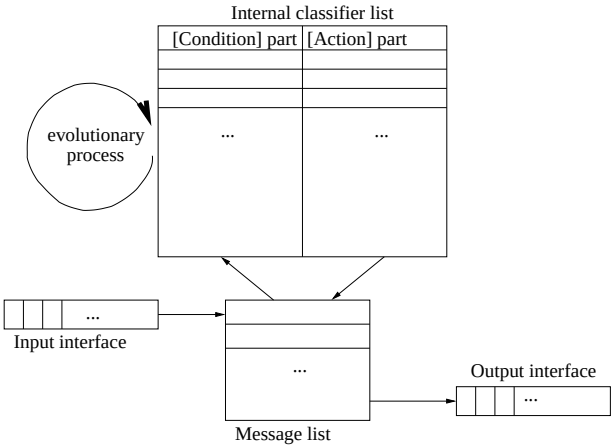

Fig. 2 Architecture of Holland's CS1 system, after Booker et al. (1989). The input interface produces messages which are added to the internal message list. A set of production rules called "classifiers" is applied to these messages to produce new messages by firing rules whose [Condition] part matches with the original messages. Some messages are sent to the output interface, giving rise to actions.

Given the difficulty of obtaining reliably convincing performance with diverse versions of these systems, LCS research became less active in the 1980s. An historical overview of the extension of these systems was published in Wilson and Goldberg (1989).

3.2 The renewal

The first important evolution in the history of LCS research is correlated to the parallel progress in RL research, particularly with the publication of the Q-LEARNING algorithm (Watkins, 1989).

Classical RL algorithms such as Q-LEARNING rely on an explicit enumeration of all the states of the system. But, since they represent the state as a collection of a set of sensations called "*attributes*", LCSs do not need this explicit enumeration thanks to a *generalization* property that we will describe later on. This *generalization* property has been recognized as the distinguishing feature of LCSs with respect to the classical RL framework. Indeed, it led Lanzi to define LCSs as RL systems endowed with a generalization capability (Lanzi, 2002).

An important step in this change of perspective was the analysis by Dorigo and Bersini of the similarity between the BUCKET BRIGADE algorithm (Holland, 1986) used so far in LCSs and the Q-LEARNING algorithm (Dorigo and Bersini, 1994). At the same time, Wilson published a radically simplified version of the initial LCS architecture, called ZCS [2] (Wilson, 1994), in which the list of internal messages was removed.

[2] *Zeroth-level Classifier System.*

ZCS defines the fitness or *strength* of a classifier as the accumulated reward that the agent can get from firing the classifier, giving rise to the "strength-based" family of LCSs. As a result, the GA eliminates classifiers providing less reward than others from the population.

After ZCS, Wilson invented a more subtle system called XCS (Wilson, 1995), in which the fitness is bound to the capacity of the classifier to accurately predict the reward received when firing it, while action selection still relies on the expected reward itself. XCS appeared very efficient and is the starting point of a new family of "accuracy-based" LCSs.

Finally, two years later, Stolzmann proposed an anticipatory LCS called ACS (Stolzmann, 1998; Butz et al., 2000) giving rise to the "anticipation-based" LCS family. As we will show, this third family is quite distinct from the other two. Its scientific roots come from research in experimental psychology about latent learning (Tolman, 1932; Seward, 1949). More precisely, Stolzmann was a student of Hoffmann (Hoffmann, 1993) who built a psychological theory of learning called "Anticipatory Behavioral Control" inspired from Herbart's work (Herbart, 1825).

The extension of these three families is at the heart of modern LCS research. Before closing this historical overview, we must add that, after a second survey of the field (Lanzi and Riolo, 2000), a further important evolution is taking place. Even if the initial impulse in modern LCS research was based on the solution of sequential decision problems, the excellent results of XCS on *data mining* problems (Bernadó et al., 2001) have given rise to an important extension of researches towards automatic classification problems, as exemplified by Booker (2000) or Holmes (2002).

4 Fundamental aspects

In order to present model-free RL methods in the context of MDPs, let us adopt the viewpoint of Lanzi (Lanzi, 2002), who proposes a nice framework that defines LCSs as RL systems with generalization properties.

A simple implementation of the Q-LEARNING algorithm consists in building a table called *Q-table* which contains $< s, a, p >$ triples, where s and a represent the state and action of the agent, and p the current estimate of the long term reward that the agent can expect from this (s, a) pair. The process consists in adding an entry to the table each time the agent carries out a new state-action pair, and updating the value of p corresponding to the action performed in the current state according to equation (3).

The problem with this tabular representation is that the table can grow to a very large size. As Lanzi (2002) clearly explains, in order to avoid this problem, one must use a different representation endowed with a generalization capability. In practice, this can be done by using $< c, a, p >$ triples where the c elements do not stand for one single state anymore, but rather for a condition which will cluster together all states for which this condition holds.

LCSs are characterized by their specific way of representing the conditions c and by the processes they call upon for finding conditions giving rise to both

compact and efficient representations. We will detail these features in the next section.

4.1 Classifier Formalism

An LCS is composed of a population of classifiers. Each classifier is a triple $< c, a, p >$ containing a [Condition] part, an [Action] part, and an estimation of the expected accumulated reward that the agent can get if it fires this classifier.

Formally, the [Condition] part of classifiers is a list of tests. There are as many tests as attributes in the problem description, each test being applied to a specific attribute. In the most common case where the test specifies a value that an attribute must take for the [Condition] to match, the test is represented just by this value. There exists a particular test, denoted "#" and called "don't care", which means that the [Condition] of the classifier will match whatever the value of the corresponding attribute. At a more global level, the [Condition] part of a classifier matches if all its tests hold in the current situation. In such a case, the classifier can be fired.

The first LCSs only used Boolean tests. In that case, each test takes its value from $\{0, 1, \#\}$. More recently, several systems have been extended to deal with nominal values [3] or continuous values [4]. In the continuous case, a test generally specifies an interval in which the value of the attribute must lie so that the [Condition] part matches. There are also LCSs in which the tests are expressed as a symbolic condition called "S-expression" (Ahluwalia and Bull, 1999; Lanzi and Perrucci, 1999). In the boolean and nominal case, the generalization property of LCSs relies on the # symbol. Thanks to # symbols in the [Condition] part of classifiers, two input situations are considered equivalent with respect to a given classifier if the specified (non-#) values in the condition match the corresponding attributes of the two situations. In the continuous case, a test is more general than another if the specified interval of the first contains the specified interval of the second.

After describing the representation manipulated by LCSs, we must present their mechanisms. The general goal is to design an RL system; thus there will be at its heart an *action selection* mechanism relying on the value of all actions in different situations. Furthermore, these systems are endowed with a generalization capability which relies on *classifier population evolution* mechanisms in order to reach a satisfactory level of generality. We present both categories of mechanisms in the next sections and we will show afterward that families of systems can be distinguished by the way they deal with interactions between these mechanisms.

4.2 Action Selection

The set of classifiers whose [Condition] part matches the current situation is called the "match-set" and denoted [M]. Furthermore, we denote by [A]—the "action-

[3] For instance, MACS (Gérard et al., 2005).

[4] For instance, XCSF (Wilson, 2001).

set"—the set of classifiers in [M] which advocate the action a that is actually chosen. Given the generalization property of classifiers, the [Condition] part of several classifiers can match at the same time, while they do not necessarily specify the same action. Thus, LCSs must contain an action selection mechanism which chooses the action executed given the list of classifiers in [M]. In order to benefit from RL properties, this mechanism must use the expected accumulated reward of each classifier, but it must also include some trade-off between exploration and exploitation. When presenting the different LCSs, we will describe the different action selection and classifier evaluation mechanisms.

4.3 Classifier population evolution

Ensuring that each classifier reaches the ideal generalization level is a crucial concern in LCSs. The system must find a population which covers the state space as compactly as possible, without being detrimental to the optimality of behavior. The mechanisms responsible for this property differ from one system to the other, but they all rely on adding and deleting classifiers.

In strength-based and accuracy-based systems, generalization and specialization of classifiers are caused by a GA which evolves the classifiers' [Condition] parts. Compactness emerges from the competition between classifiers and the fact that the population is limited. In the case of anticipation-based systems, more deterministic generalization and specialization heuristics are used, even if a GA is at work in the generalization process in ACS2. We will describe the various classifier creation and deletion mechanisms once we have presented the three main families of modern LCSs.

5 Strength-based systems: ZCS

5.1 Presentation

Strength-based LCSs are the simplest ones. Each classifier contains only one evaluation variable which is both its estimation of the accumulated reward brought by its firing and its fitness for the population evolution process. The most typical strength-based LCS is ZCS. It works with a classifier population of fixed size P. A general view of the interaction between an agent controlled by ZCS and its environment is shown in figure 3.

5.2 Action selection

In ZCS, once [M] is determined, the selection of the fired action is based on a roulette wheel (Goldberg, 1989) mechanism which induces some exploration. Indeed, instead of always firing the action specified by the strongest classifier in [M], each classifier has a probability to be fired depending on its relative strength with respect to the strength of all other classifiers in [M], and one classifier is chosen randomly based on these probabilities.

Fig. 3 In a strength-based LCS, classifiers are composed of a [Condition] part, an [Action] part, and a value that is both the classifier fitness and its estimation of expected reward. Among the classifiers, the system selects those whose [Condition] part matches the current situation and chooses one classifier in that set with a roulette wheel mechanism. The corresponding action is executed (in the example, the agent executes $[001]$.

5.3 Classifier evaluation

The reward propagation mechanism in ZCS is close to the original BUCKET BRIGADE algorithm (Holland, 1986), but it differs by the fact that there is no bidding process. More precisely, there is a three-step process. First, all classifiers which advocated the action a_{t-1} chosen at the previous time step share equally a fraction $\gamma.\alpha$ of the sum of the values of classifiers in [A]. Second, all classifiers in [A] share equally a fraction α of the reward r_t received for executing a_t. Last, the value of all classifiers in [M] which do not belong to [A] is reduced with a tax τ. This mechanism is closer to SARSA (Sutton, 1996) than to Q-LEARNING (Watkins, 1989).

5.4 Classifier Creation and Deletion

In ZCS, the evolution of the classifier population is driven by a GA and a covering operator.

- At each time step, there is a probability p of running the GA. If it is run, the GA uses a roulette wheel mechanism based on fitness to choose two classifiers from the global population. Two offspring of these classifiers are generated thanks to a one-point crossover operator and a mutation operator. The initial fitness of the offspring is the average of the fitness of their parents. They replace in the population two classifiers chosen by a roulette wheel mechanism based on the inverse of the fitness.
- The covering operator is called each time [M] is empty or only contains classifiers whose fitness is Φ times weaker than the average fitness of the global population. The operator adds to the population a new classifier matching the current situation, whose action is chosen randomly and whose initial fitness is equal to the average fitness of the population. Each test in the [Condition] part can take the # value with a 33% probability.

5.5 Parameters

For ZCS parameters, Bull and Hurst (2002) gives the following default values: population size $P = 400$, initial fitness $S_0 = 20.0$, learning rate $\alpha = 0.2$, discount factor $\gamma = 0.71$, tax $\tau = 0.1$, GA firing rate $p = 0.25$, crossover rate $P_c = 0.5$, mutation rate $P_m = 0.002$, covering operator firing rate $\Phi = 0.5$.

5.6 Recent evolution

At the time of writing this survey, the last publication about ZCS internal processes dates back to 2002 (Bull and Hurst, 2002), which can be interpreted as a decline of research on strength-based LCSs in favor of accuracy-based LCSs. However, more general works on the strength-based approach are still being published (Bull, 2004a, 2005).

Before this drop of interest, ZCS was extended in two directions in order to solve problems where the Markov property does not hold: the addition of internal registers (Cliff and Ross, 1994) and addition of a classifier-chaining mechanism (Tomlinson and Bull, 1998). We will describe these extensions more accurately in the context of XCS, to which they have also been applied.

Moreover, ZCS has been used in several applications. In particular, Bull (Bull, 1998, 1999) used it to represent trading agents in a simplified market, Cao (Cao et al., 1999, 2001) used it to control traffic junction simulations and Miramontes Hercog (Miramontes Hercog and Fogarty, 2002) used it to solve classical collective behavior benchmarks.

The main drawback of ZCS is that overly general classifiers can be sustained in the population and can result in the system taking suboptimal actions. Consider a classifier that can fire (matches) both for an optimal action close to some reward and a suboptimal action far from a reward. Its strength will be the average between a high expected value and a low expected value. Unfortunately, this average can be higher than the expected reward of a more specialized classifier specifying the optimal action far from the reward. As a result, the GA will only keep the first classifier while the second would be more useful for efficient behavior far from the goal.

Bull and Hurst (2002) shows on a toy problem that, given well-chosen parameters this problem can be overcome, but this paper did not completely rescue ZCS, because XCS proposes a more elegant solution to the same problem. This fact can be seen as the reason for the preference for accuracy-based over strength-based LCSs.

6 Accuracy-based systems: XCS

6.1 Presentation

XCS, invented by Wilson (Wilson, 1995) one year after ZCS, has been the most studied and applied LCS for several years (Kovacs, 2002). The important research

efforts concerned with this system have resulted in much improvement in both its performance and in understanding the reasons for this performance. The reader can find in Butz and Wilson (2002) a detailed algorithmic description of XCS and in Butz et al. (2004) a synthesis of its underlying mechanisms and their interactions.

The central idea in XCS consists in decoupling the RL process and the population evolution process by introducing a fitness function that is not proportional to the expected reward, but to the accuracy of the prediction of this reward. The classifiers that survive in the population are no longer necessarily those predicting a large reward, but those accurately predicting the reward they receive, be it large or small. As a consequence, an important difference between ZCS and XCS is that the second keeps in its population classifiers firing far from a source of positive reward, thus predicting a small reward, given that they do so accurately. As a result, XCS covers the state space more efficiently than ZCS.

Moreover, the generalization process in XCS groups together in the same [Condition] part situations for which the expected reward is similar. Otherwise, the prediction could not be accurate. To learn more about the comparison between both families, we refer to Kovacs (2004).

Beyond this first difference, the processes in XCS differ significantly from those of ZCS. We will describe them more precisely in the following sections.

6.2 Action selection

Studies of XCS describe different ways of dealing with the compromise between exploration and exploitation. In fact, XCS (as well as, in principle, ZCS) is neutral with respect to action selection, and any of the methods seen in RL apply. Among these, we can mention ϵ-$greedy$, alternation of pure exploration trials and pure exploitation trials, or the roulette wheel process also used in ZCS. The choice of the executed action can be done by treating each classifier separately, or by grouping together all classifiers advocating for the same action in *prediction arrays*.

6.3 Classifier evaluation

The classifier evaluation algorithm is closer in spirit to Q-LEARNING than to BUCKET BRIGADE. It relies on the estimation update process described in equation (2). The local RL process, applied only to the classifiers of [A], is the following:

- the expected reward p is updated given the immediate reward received r in the following way: $p \leftarrow p + \beta(r - p)$,
- then the prediction error corresponding to this expectancy is updated: $\epsilon \leftarrow \epsilon + \beta(|r - p| - \epsilon)$
- the raw prediction accuracy is derived: $k = \begin{cases} 1 & \text{if } \epsilon < \epsilon_0 \\ \alpha(\frac{\epsilon}{\epsilon_0})^{-\nu} & \text{otherwise} \end{cases}$ where $\nu > 0$.
- the *relative* prediction accuracy is computed with respect to its value for other classifiers of [A]: $k' = \frac{k}{\sum_{x \in A} k_x}$

– finally, the fitness f of classifiers is derived: $f \leftarrow f + \beta(k' - f)$

In sequential decision problems, Butz obtained a significant performance improvement by replacing the algorithm above by a gradient descent technique (Butz et al., 2003a).

6.4 Classifier Creation and Deletion

As in ZCS, the creation of classifiers relies both on a GA and a covering operator. But, in contrast with ZCS, in XCS the GA is applied in [A] rather than in the global population. This induces a competition between classifiers matching the same situations rather than a global competition. The GA is run each θ_{GA} time steps. The parents are chosen according to a roulette wheel process with a probability proportional to their fitness. Recently, Butz (Butz et al., 2003b) obtained a significant performance improvement using tournament selection instead of roulette-wheel, in both single-step classification tasks and in multi-step, sequential decision tasks.

Two offspring of the chosen parents are inserted in the global population, after the application of a mutation operator that can be either non-directional or guided by the current input. Mutation is said to be "guided" when one test cannot mutate toward a value different from the corresponding value in the current input.

The covering operator adds classifiers when some actions are absent from [M]. These classifiers specify the missing actions, and their [Condition] part matches the current input, after generalization by adding some # tests with a probability $P_{\#}$ for each test.

In XCS, the global population size is bounded. When new classifiers are added, if the size limit is reached, a corresponding number of classifiers must be deleted. The deletion process is based on an estimation of the average size of the action sets in which each classifier is involved. A classifier is selected by a roulette wheel process based on this estimation and deleted. In order to estimate the average size of the action sets in which a classifier is involved, the estimation process updates an estimator a_s according to the classical technique described by equation (2) each time the classifier is involved in an action set: $a_s = a_s + \beta(size - a_s)$ where $size$ is the size of the current action set.

Two additional processes control the population size in XCS.

– Given that the creation of classifiers can generate classifiers identical to already existing ones, then rather than keeping several identical classifiers in the population, which would not be efficient from a memory and computation time viewpoint, XCS uses a notion of *macro-classifier* which associates to each classifier a *numerosity* N corresponding to the number of exemplars in the population. The population size bound is reached if the sum of numerosities is higher than the bound.
– There is an optional *subsumption* process which favors generalization of classifiers within the population. Each time a new classifier is created, this process checks whether the population contains a sufficiently accurate classifier which subsumes (is logically more general than) the new classifier. If such a classifier exists, instead of adding the new one to the population, the numerosity of

the more general classifier is augmented. Though it favors generalization, this additional process is computationally costly. As a result, it is not always used in typical experiments with XCS.

6.5 Parameters

The standard parameter values in XCS are the following: population size $P = 800$ or $P = 2000$ depending on the problem, learning rate and estimation rate $\alpha = 0.1$ and $\beta = 0.2$, GA firing interval $\theta_{GA} = 25$, crossover rate $\xi = 0.8$, mutation rate $\mu = 0.04$, probability of # in covering $P_{\#} = 0.6$. See Butz and Wilson (2002) for a more detailed parameter description.

6.6 Recent evolution

Some recent improvements in XCS have already been described in the previous sections, namely the incorporation of tournament selection in the GA (Butz et al., 2003b) and the gradient descent approach to classifier evaluation (Butz et al., 2003a). We must add that, very recently, Butz and collaborators (Butz et al., 2006) have shown that importing recent techniques from the evolutionary computation literature to efficiently detect *building blocks* (Goldberg, 1989) in the [Condition] part of classifiers can improve the performance of XCS when such building blocks are present in the structure of the problems.

Further, other extensions result in an increased application domain for XCS. The oldest of these extensions are devoted to problems where the Markov property does not hold. In particular, the *perceptual aliasing* problem, where two or more different underlying states of the environment have the same appearance to the system, is the most studied.

The two main ways of dealing with this problem are *classifier chaining* and *internal register management*. In the first case, implemented in CXCS (Tomlinson and Bull, 2000) after ZCCS (Tomlinson and Bull, 1998) which relied on ZCS, the system chains classifiers leading up to an ambiguous situation so as to avoid deciding in the ambiguous context.

In the second case, implemented in XCSM and XCSMH (Lanzi, 1998) after ZCSM (Cliff and Ross, 1994) which relied on ZCS, classifiers incorporate in the [Condition] part some additional tests on the value of an internal register, and this value is modified by additional information in the [Action] part. The adaptive mechanisms must make sure that the value of the internal register reliably discriminates between different aliased situations. For a more detailed description of all mechanisms dedicated to the solution of the perceptual aliasing problem in LCSs, see Landau and Sigaud (2006).

Other extensions of XCS deal with continuous state and reward spaces. With XCSR (Wilson, 2000), Wilson proposed to represent continuous state spaces with tests checking if the real number corresponding to the value of an attribute is included in an interval in $\mathbb{R}$. In XCSR, the intervals are coded with two real numbers representing the center of the interval and the spread around this center. Later,

Stone and Bull (2003) showed that this representation was inducing a bias detrimental to the generality of the system and proposed another representation where the real numbers specify the bounds of the interval in any order. In parallel, Wilson proposed in (Wilson, 2004) an extension of XCS to the management of rewards that are a continuous function of a continuous state.

In line with these works about continuous representations, one of the most active research trends in the domain of XCS extension results from Wilson's theoretical account of XCS as a generic function approximator. The resulting system, XCSF (Wilson, 2001), is at the heart of several studies and extensions, one of the most recent being Lanzi et al. (2006).

7 Anticipation-based systems: ALCSs

7.1 Presentation

Although they share a number of common characteristics with standard LCSs, ALCSs deviate from the classical framework on one fundamental point. Instead of [Condition] → [Action] classifiers, they manipulate [Condition] [Action] → [Effect] classifiers. The [Effect] part represents the expected effect (next state) of the [Action] part in all situations that match the [Condition] part of the classifier. Such a set of classifiers constitutes what is called in the RL literature a *model of transitions*. Since they learn a model of transitions, ALCSs are an instance of model-based RL [5] architecture, a category of systems whose prototype is the DYNA architecture (Sutton, 1990). As a result, ALCSs can be seen as combining two crucial properties of RL systems. Like DYNA architectures, they learn a model of transitions, which endows them with anticipation and planning capabilities and speeds up the learning process. Like classical LCSs, they are endowed with a generalization property, which lets them build much more compact models than tabular DYNA architectures (Gérard and Sigaud, 2003).

Historically, it seems that Riolo (Riolo, 1991) was the first to publish an LCS endowed with an explicit anticipation capability. His system, CFSC2, was directly inspired by the original LCS architecture of Holland (Holland and Reitman, 1978) with internal messages.

The first ALCS designed after Wilson's simplifications of the original LCS architectures (see section 3) was ACS (Stolzmann, 1998; Butz et al., 2000). Central to ACS, the ALP (*Anticipatory Learning Process*) algorithm is the formal counterpart of Hoffmann's psychological theory of *Anticipatory Behavioral Control* (Hoffmann, 1993). ACS was later extended by Butz to become ACS2 (Butz, 2002). In parallel, Gérard proposed YACS (Gérard et al., 2002) and MACS (Gérard et al., 2005).

In ACS, ACS2 and YACS, the [Effect] part of each classifier tells which attributes do change and which do not. To represent that, the [Effect] parts can contain a "=" symbol, which means that the corresponding attribute does not change. For instance, applied to the situation `[1031]`, the classifier `[#0#1] [0] [=10=]`

[5] Or indirect RL

predicts that the situation resulting from the application of action [0] will be [1101]. Applied to [2011], it predicts [2101]. This formalism is able to represent regularities such as *"when the agent perceives a wall to the north, whatever it perceives in any other direction, going north does not produce any sensory change"*, which is represented by the following classifier: [1#######] [North] [========].

By contrast, MACS can represent regularities between different attributes with a classifier such as [#1#######] [East] [1????????], where the "?" symbol means that the classifier cannot predict the value of the considered attribute. The addition of this new symbol results in the capacity to predict separately the value of different attributes at the next time step. In the case of MACS, the authors chose to predict the value of one attribute only in each [Effect] part.

Experimental results on model compactness and convergence speed of MACS have shown that it builds a slightly more compact model than YACS, which builds models four times more compact than ACS (Gérard et al., 2005). Furthermore, MACS builds this model three times faster than YACS, and nine times faster than ACS in number of time steps. Thanks to these improvements, MACS can deal with much more complex problems than the other ALCSs. Offsetting this is that the algorithmic description of MACS is more complex. Furthermore, some heuristics in MACS are designed for deterministic problems rather than stochastic ones. As a result, the application domain of MACS is more restricted.

In the following sections, we focus on the processes of MACS, showing in what respect they differ from those of the other systems.

7.2 Action selection

ACS, ACS2 et YACS use classical solutions to deal with the exploration versus exploitation trade-off. By contrast, in order to efficiently explore large size problems, MACS combines three criteria:

- the agent first chooses actions bringing more information about the transitions that have not been tried enough;
- then, if the best actions are equivalent with respect to the first criterion, it chooses actions bringing more external reward, as any RL system does;
- finally, if the best actions are equivalent with respect to the first and second criteria, it chooses actions that have not been tried for the longest time, so as to handle non-stationary environments as efficiently as possible.

7.3 Classifier evaluation

In MACS, an immediate reward function and an iterative propagation process are associated with all criteria defined above. The propagation process converges to the expected reward only if the model of transitions is accurate enough. This is why MACS favors exploration first. The architecture of MACS clearly distinguishes two aspects of classifier evaluation:

- first, the values corresponding to the three previous criteria are associated with each (situation, action) pair;
- second, classifiers themselves incorporate some accuracy indicators driving the population evolution process as described hereafter.

7.4 Classifier Creation and Deletion

In order to obtain a model of transitions as general, accurate and compact as possible, ALCSs generally rely on the combination of two heuristics:

- a specialization heuristic is applied to inaccurate classifiers;
- a generalization heuristic is applied to overspecialized classifiers.

When appropriate, the combination of both heuristics results in the convergence of the population to a maximally general and accurate set of classifiers.

For the specialization process, all ALCSs rely on the same idea: when a general classifier oscillates between correct and incorrect predictions, it is too general and must be specialized. Its [Condition] part must be modified so as to match only in situations where its prediction is correct. ACS, ACS2 and YACS randomly choose a # test and change it into a specialized test, whereas MACS uses a further heuristic to efficiently choose the most appropriate # test to be specialized.

The generalization process is more complex. In ACS and ACS2, a GA is used to replace specific classifiers with more general ones. In YACS and MACS, a more complex algorithm relies on the estimated accuracy of classifiers to determine rationally if generalization will result in an improvement or not. We refer to the papers on each of these systems for a more detailed description of these processes and the corresponding parameters.

7.5 Recent evolution

In YACS and MACS, the systems were looking for the optimal generalization rate only in the model of transitions, without generalizing the reward model or the value function model. The last two models are represented by a table giving a value for each encountered state, which is detrimental to the compactness argument in favor of these systems. Recently, Butz (Butz and Goldberg, 2003) proposed XACS which also generalizes the value function in ACS2, by using XCS.

The most active research trend in this domain consists in trying to extend XCS in order to endow it with explicit anticipatory capabilities, but there is no convincing result so far. In addition, the natural continuation of ALCS research relies in the Factored MDP domain, as we will describe in the next section.

8 Trends and future directions

The modern LCS research community is still small, but rapidly expanding. This expansion should result in a better exploitation of the suitability of LCS for application in complex real-world problems, due to their high power of expression

combined with a very readable formalism for the human expert. Indeed, quite paradoxically, there have been few publications about industrial applications of LCSs. Note that Bull (2004b) is dedicated to this question, however. Moreover in France, several specific LCS architectures have been used to solve complex sequential decision problems in video games (Sanza, 2001; Sanchez, 2004; Robert, 2005). Furthermore, as we mentioned in the introduction of this paper, outstanding results of XCS on automatic classification problems (Bernadó et al., 2001) have induced an important applied research effort in the data mining domain (Holmes, 2002).

In more theoretical directions, two important trends can be distinguished. First, the numerous research works dedicated to the improvement of XCS are combined with theoretical investigations which improve the understanding of the efficiency of its underlying mechanisms. Such investigations benefit from parallel progress in the theoretical analysis of evolutionary approaches (Poli and Langdon, 1998), but this research trend is far from exhausted (Drugowitsch and Barry, 2006).

Second, on the RL side, the recent discovery of the Factored MDP framework (Boutilier et al., 1995) raises a serious hope of convergence between the LCS community and the much wider RL community. Indeed, in this theoretical framework, the state is represented as a collection of random variables (Boutilier et al., 2000), which exactly corresponds to the LCS representation formalism, each test of the [Condition] part corresponding to one random variable (Sigaud et al., 2004). The generalization property of LCSs is equivalent to the factorization property of the new theoretical framework.

But, whereas research on Factored MDPs is restricted to the resolution of planning problems, with the model of transitions being given in advance, LCSs and particularly ALCSs do learn the model of transitions simultaneously with determining an optimal policy. Thus they solve true RL problems. Directly inspired by previous work in ALCSs, Degris et al. have recently published some work about learning the model of transitions in the standard Factored MDP framework (Degris et al., 2006a,b), which may constitute a step towards the convergence of both research trends and should give rise to further research efforts in the future.

9 Conclusion

In this paper, we have presented Learning Classifier Systems, which add to the classical Reinforcement Learning framework the possibility of representing the state as a vector of attributes and finding a compact expression of the representation so induced. Their formalism conveys a nice interaction between learning and evolution, which makes them a class of particularly rich systems, at the intersection of several research domains. As a result, they profit from the accumulated extensions of these domains.

We hope that this survey has given to the interested reader an appropriate starting point to investigate the different streams of research that underlie the rapid evolution of LCS. In order to further study the different topics treated here, new resources have become available since Kovacs (2002), which surveyed the field in 2001. In particular, a key starting point is the website dedicated to the LCS community, which can be found at the following URL: *http://lcsweb.cs.bath.ac.uk/*.

A more detailed view of recent extensions can be found in the proceedings of the IWLCS workshops (Lanzi et al., 2000, 2001, 2002a,b; Stolzmann et al., 2002) and GECCO conferences (Banzhaf et al., 1999; Whitley et al., 2000; Spector et al., 2001; Langdon et al., 2002; Cantu-Paz et al., 2003; Deb et al., 2004; Beyer et al., 2005), which bring together each year most of the work dedicated to this research domain [6].

References

Ahluwalia, M. and Bull, L. (1999). A Genetic Programming-based Classifier System. In Banzhaf et al. (1999), pages 11–18.

Bacardit, J. and Garrell, J. M. (2003). Evolving multiple discretizations with adaptive intervals for a Pittsburgh rule-based learning classifier system. In Cantú-Paz, E., Foster, J. A., Deb, K., Davis, D., Roy, R., O'Reilly, U.-M., Beyer, H.-G., Standish, R., Kendall, G., Wilson, S., Harman, M., Wegener, J., Dasgupta, D., Potter, M. A., Schultz, A. C., Dowsland, K., Jonoska, N., and Miller, J., (Eds.), *Genetic and Evolutionary Computation – GECCO-2003*, pages 1818–1831, Berlin. Springer-Verlag.

Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., and E., S. R., (Eds.) (1999). *Proceedings of the 1999 Genetic and Evolutionary Computation Conference Workshop Program*. Morgan Kaufmann.

Bellman, R. E. (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ.

Bellman, R. E. (1961). *Adaptive Control Processes: A Guided Tour*. Princeton University Press.

Bernadó, E., Llorà, X., and Garrel, J. M. (2001). XCS and GALE : a comparative study of two Learning Classifer Systems with six other learning algorithms on classification tasks. In Lanzi, P.-L., Stolzmann, W., and Wilson, S. W., (Eds.), *Proceedings of the fourth international workshop on Learning Classifer Systems*.

Bertsekas, D. P. (1995). *Dynamic Programming and Optimal Control*. Athena Scientific, Belmont, MA.

Beyer, H.-G., O'Reilly, U.-M., Arnold, D., Banzhaf, W., Blum, C., Bonabeau, E., Cant Paz, E., Dasgupta, D., Deb, K., Foste r, J., de Jong, E., Lipson, H., Llora, X., Mancoridis, S., Pelikan, M., Raidl, G., Soule, T., Tyrrell, A., Watson, J.-P., and Zitzler, E., (Eds.) (2005). *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2005*. ACM Press, Washington DC.

Booker, L., Goldberg, D. E., and Holland, J. H. (1989). Classifier Systems and Genetic Algorithms. *Artificial Intelligence*, 40(1-3):235–282.

Booker, L. B. (2000). Do we really need to estimate rule utilities in classifier systems? In Lanzi, P.-L., Stolzmann, W., and Wilson, S. W., (Eds.), *Learning Classifier Systems. From Foundations to Applications*, volume 1813 of *Lecture Notes in Artificial Intelligence*, pages 125–142, Berlin. Springer-Verlag.

[6] The GECCO 2006 proceedings and the IWLCS 2006 and combined 2003-2005 proceedings were not yet published at the time we finished this survey.

Boutilier, C., Dearden, R., and Goldszmidt, M. (1995). Exploiting Structure in Policy Construction. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1104–1111, Montreal.

Boutilier, C., Dearden, R., and Goldszmidt, M. (2000). Stochastic Dynamic Programming with Factored Representations. *Artificial Intelligence*, 121(1):49–107.

Bull, L. (1998). On ZCS in Multi-agent Environments. *Lecture Notes in Computer Science*, 1498:471–479.

Bull, L. (1999). On using ZCS in a Simulated Continuous Double-Auction Market. In Banzhaf et al. (1999), pages 83–90.

Bull, L. (2004a). A Simple Payoff-based Learning Classifier System. In Yao, X., Burke, E., Lozano, J. A., Smith, J., Merelo-Guervs, J. J., Bullinaria, J. A., Rowe, J., Tino, P.and Kabn, A., and Schwefel, H.-P., (Eds.), *Proceedings of Parallel Problem Solving from Nature (PPSN VIII)*. Springer-Verlag.

Bull, L., (Ed.) (2004b). *Applications of Learning Classifier Systems*. Springer.

Bull, L. (2005). Two simple learning classifier systems. In Bull, L. and Kovacs, T., (Eds.), *Foundations of Learning Classifier Systems*. Springer-Verlag.

Bull, L. and Hurst, J. (2002). ZCS redux. *Evolutionary Computation*, 10(2):185–205.

Butz, M., Kovacs, T., Lanzi, P. L., and Wilson, S. W. (2004). Toward a theory of generalization and learning in xcs. *IEEE Transactions on Evolutionary Computation*, 8(1):28–46.

Butz, M., Pelikan, M., Llorà, X., and Goldberg, D. E. (to appear, 2006). Automated Global Structure Extraction for Eeffective Local Building Block Processing in XCS. *Journal of Evolutionary Computation*.

Butz, M. V. (2002). An Algorithmic Description of ACS2. In Lanzi et al. (2002a), pages 211–229.

Butz, M. V. and Goldberg, D. E. (2003). Generalized state values in an anticipatory Learning Classifier System. In Butz, M. V., Sigaud, O., and Gérard, P., (Eds.), *LNAI 2684: Anticipatory Behavior in Adaptive Learning Systems*, pages 281–301. Springer-Verlag.

Butz, M. V., Goldberg, D. E., and Lanzi, P. L. (2003a). Gradient descent methods in learning classifier systems : Improving XCS performance in multistep problem. Technical Report 2003028, IlliGAL.

Butz, M. V., Goldberg, D. E., and Stolzmann, W. (2000). Introducing a genetic generalization pressure to the Anticipatory Classifier Systems part I: Theoretical approach. In Whitley et al. (2000), pages 34–41. Also Technical Report 2000005 of the Illinois Genetic Algorithms Laboratory.

Butz, M. V., Sastry, K., and Goldberg, D. E. (2003b). Tournament selection: Stable fitness pressure in XCS. In Cantú-Paz, E., Foster, J. A., Deb, K., Davis, D., Roy, R., O'Reilly, U.-M., Beyer, H.-G., Standish, R., Kendall, G., Wilson, S., Harman, M., Wegener, J., Dasgupta, D., Potter, M. A., Schultz, A. C., Dowsland, K., Jonoska, N., and Miller, J., (Eds.), *Genetic and Evolutionary Computation – GECCO-2003*, volume 2724 of *LNCS*, pages 1857–1869. Springer-Verlag.

Butz, M. V. and Wilson, S. W. (2002). An algorithmic description of xcs. *Journal of Soft Computing*, 6(3–4):144–153.

Cantu-Paz, E., Foster, J. A., Deb, K., O'Reilly, U.-M., Beyer, H.-G., Standish, R., Kendall, G., Wilson, S., Harman, M., Weneger, J., Dasgupta, D., Potter, M. A., Schultz, A. C., Dowsland, K. A., Jonoska, N., and Miller, J., (Eds.) (2003). *Proceedings of the 2003 Genetic and Evolutionary Computation Conference Workshop Program*, Chicago, IL. Springer Verlag.

Cao, Y. J., Ireson, N., Bull, L., and Miles, R. (1999). Design of a Traffic Junction Controller using a Classifier System and Fuzzy Logic. In Reusch, B., (Ed.), *Proceedings of the Sixth International Conference on Computational Intelligence, Theory and Applications (6th Fuzzy Days)*, volume 1625 of *LNCS*, pages 342–353. Springer-Verlag.

Cao, Y. J., Ireson, N., Bull, L., and Miles, R. (2001). An evolutionary intelligent agents approach to traffic signal control. *International Journal of Knowledge-based Intelligent Engineering Systems*, 5(4):279–289.

Cliff, D. and Ross, S. (1994). Adding memory to ZCS. *Adaptive Behavior*, 3(2):101–150.

Deb, K., Poli, R., Banzhaf, W., Beyer, H.-G., Burke, E., Darwen, P., Dasgupta, D., Floreano, D., Foster, J. A., Harman, M., Holland, O., Lanzi, P.-L., Spector, L., Tettamanzi, A., Thierens, D., and Tyrrell, A., (Eds.) (2004). *Proceedings of the 2004 Genetic and Evolutionary Computation Conference Workshop Program*, Seattle, WA. Springer Verlag.

Degris, T., Sigaud, O., and Wuillemin, P.-H. (to appear, 2006a). Chi-square tests driven method for learning the structure of factored mdps. In *Proceedings of the 22rd Uncertainty in Artificial Intelligence Conference (UAI'2006)*, MIT, Massachussetts.

Degris, T., Sigaud, O., and Wuillemin, P.-H. (to appear, 2006b). Learning the structure of factored markov decision processes in reinforcement learning problems. In *Proceedings of the 23rd International Conference on Machine Learning (ICML'2006)*, CMU, Pensylvania.

Dorigo, M. and Bersini, H. (1994). A comparison of Q-Learning and Classifier Systems. In Cliff, D., Husbands, P., Meyer, J.-A., and Wilson, S. W., (Eds.), *From Animals to Animats 3*, pages 248–255, Cambridge, MA. MIT Press.

Drugowitsch, J. and Barry, A. (2006). Towards Convergence of Learning Classifier Systems Value Iteration. Technical Report CSBU-2006-03,, Department of Computer Science, University of Bath.

Gérard, P., Meyer, J.-A., and Sigaud, O. (2005). Combining latent learning with dynamic programming in MACS. *European Journal of Operational Research*, 160:614–637.

Gérard, P. and Sigaud, O. (2003). Designing efficient exploration with MACS: Modules and function approximation. In *Proceedings of the Genetic and Evolutionary Computation Conference 2003 (GECCO'03)*, pages 1882–1893, Chicago, IL. Springer-Verlag.

Gérard, P., Stolzmann, W., and Sigaud, O. (2002). YACS: a new Learning Classifier System with Anticipation. *Journal of Soft Computing : Special Issue on Learning Classifier Systems*, 6(3-4):216–228.

Golberg, D. E. and Holland, J. H. (1988). Guest Editorial: Genetic Algorithms and Machine Learning. *Machine Learning*, 3:95–99.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley, Reading, MA.

Herbart, J. F. (1825). *Psychologie als Wissenschaft neu gegründet auf Erfahrung, Metaphysik und Mathematik. Zweiter, analytischer Teil*. August Wilhem Unzer, Koenigsberg, Germany.

Hoffmann, J. (1993). *Vorhersage und Erkenntnis [Anticipation and Cognition]*. Hogrefe, Göttingen.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. University of Michigan Press, Ann Arbor, MI.

Holland, J. H. (1986). Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In *Machine Learning, An Artificial Intelligence Approach (volume II)*. Morgan Kaufmann.

Holland, J. H., Holyoak, K. J., Nisbett, R. E., and Thagard, P. R. (1986). *Induction*. MIT Press.

Holland, J. H. and Reitman, J. S. (1978). Cognitive Systems based on Adaptive Algorithms. *Pattern Directed Inference Systems*, 7(2):125–149.

Holmes, J. H. (2002). A new representation for assessing classifier performance in mining large databases. In Stolzmann, W., Lanzi, P.-L., and Wilson, S. W., (Eds.), *IWLCS-02. Proceedings of the International Workshop on Learning Classifier Systems*, LNAI, Granada. Springer-Verlag.

Kovacs, T. (2002). Learning classifier systems resources. *Journal of Soft Computing*, 6(3-4):240–243.

Kovacs, T. (2004). *Strength or Accuracy: Credit Assignment in Learning Classifier Systems*. Springer.

Landau, S. and Sigaud, O. (to appear, 2006). A Comparison between ATNoSFERES and LCSs on non-Markov problems. *Information Sciences*, *accepted for publication*.

Landau, S., Sigaud, O., and Schoenauer, M. (2005). ATNoSFERES revisited. In Beyer, H.-G., O'Reilly, U.-M., Arnold, D., Banzhaf, W., Blum, C., Bonabeau, E., Cant Paz, E., Dasgupta, D., Deb, K., Foste r, J., de Jong, E., Lipson, H., Llora, X., Mancoridis, S., Pelikan, M., Raidl, G., Soule, T., Tyrrell, A., Watson, J.-P., and Zitzler, E., (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2005*, pages 1867–1874, Washington DC. ACM Press.

Langdon, W. B., Cantu-Paz, E., Mathias, K., Roy, R., Davis, D., Poli, R., Balakrishnan, K., Honavar, V., Rudolph, G., Wegener, J., Bull, L., Potter, M. A., Schultz, A. C., Miller, J. F., Burke, E., and Jonoska, N., (Eds.) (2002). *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2002*, New York, NY. Morgan Kaufmann.

Lanzi, P.-L. (1998). Adding memory to XCS. In *Proceedings of the IEEE Conference on Evolutionary Computation (ICEC98)*. IEEE Press.

Lanzi, P.-L. (2002). Learning Classifier Systems from a Reinforcement Learning Perspective. *Journal of Soft Computing*, 6(3-4):162–170.

Lanzi, P.-L., Loiacono, D., Wilson, S., and Goldberg, D. E. (to appear, 2006). Classifier Prediction based on Tile Coding. In *Proceedings of the 2006 Genetic*

and Evolutionary Computation Conference Workshop Program.

Lanzi, P.-L. and Perrucci, A. (1999). Extending the representation of classifier conditions part ii: From messy coding to s-expressions. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., and E., S. R., (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 99)*, pages 345–352, Orlando, FL. Morgan Kaufmann.

Lanzi, P.-L. and Riolo, R. L. (2000). A roadmap to the last decade of Learning Classifier Systems research (from 1989 to 1999). In Lanzi, P.-L., Stolzmann, W., and Wilson, S. W., (Eds.), *Learning Classifier Systems: from Foundations to Applications*, pages 33–62. Springer-Verlag, Heidelberg.

Lanzi, P.-L., Stolzmann, W., and Wilson, S. W., (Eds.) (2000). *Learning Classifier Systems. From Foundations to Applications*, volume 1813 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin.

Lanzi, P.-L., Stolzmann, W., and Wilson, S. W., (Eds.) (2001). *Advances in Learning Classifier Systems*, volume 1996 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin.

Lanzi, P.-L., Stolzmann, W., and Wilson, S. W., (Eds.) (2002a). *Advances in Learning Classifier Systems*, volume 2321 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin.

Lanzi, P.-L., Stolzmann, W., and Wilson, S. W., (Eds.) (2002b). *Proceedings of the International Workshop on Learning Classifier Systems (IWLCS2002)*, Granada.

Llorà, X. and Garrell, J. M. (2002). Co-evolving different knowledge representations with fine-grained parallel learning classifier systems. In Langdon, W. B., Cantu-Paz, E., Mathias, K., Roy, R., Davis, D., Poli, R., Balakrishnan, K., Honavar, V., Rudolph, G., Wegener, J., Bull, L., Potter, M. A., Schultz, A. C., Miller, J. F., Burke, E., and Jonoska, N., (Eds.), *Proceeding of the Genetic and Evolutionary Computation Conference (GECCO2002)*. Morgan Kaufmann.

Miramontes Hercog, L. and Fogarty, T. C. (2002). Co-evolutionary classifier systems for multi-agent simulation. In Fogel, D. B., El-Sharkawi, M. A., Yao, X., Greenwood, G., Iba, H., Marrow, P., and Shackleton, M., (Eds.), *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, pages 1798–1803. IEEE Press.

Poli, R. and Langdon, W. B. (1998). Schema Theory for Genetic Programming with One-point Crossover and Point Mutation. *Evolutionary Computation Journal*, 6(3):231–252.

Puterman, M. L. and Shin, M. C. (1978). Modified Policy Iteration Algorithms for Discounted Markov Decision Problems. *Management Science*, 24:1127–1137.

Riolo, R. L. (1991). Lookahead planning and latent learning in a Classifier System. In Meyer, J.-A. and Wilson, S. W., (Eds.), *From animals to animats: Proceedings of the First International Conference on Simulation of Adaptative Behavior*, pages 316–326. MIT Press.

Robert, G. (2005). *MHiCS, une architecture de Sélection de l'Action Motivationnelle et Hiérarchique Systèmes de Classeurs pour Personnages Non Joueurs Adaptatifs*. PhD thesis, Laboratoire d'Informatique de Paris 6.

Sanchez, S. (2004). *Mécanismes évolutionnistes pour la simulation comportementale d'acteurs virtuels*. PhD thesis, Université de Siences Sociales Toulouse 1,

Toulouse.

Sanza, C. (2001). *Evolution d'entités vituelles coop'eratives par systèmes de classifieurs*. PhD thesis, Université Paul Sabatier, Toulouse.

Seward, J. P. (1949). An Experimental Analysis of Latent Learning. *Journal of Experimental Psychology*, 39:177–186.

Sigaud, O., Gourdin, T., and Wuillemin, P.-H. (2004). Improving MACS thanks to a comparison with 2TBNs. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO'04*, pages 810–823. Springer Verlag.

Smith, S. F. (1980). *A Learning System Based on Genetic Algorithms*. PhD thesis, Department of Computer Science, University of Pittsburg, Pittsburg, MA.

Spector, L., D., G. E., Wu, A., Langdon, W. B., Voigt, H. M., and Gen, M., (Eds.) (2001). *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO01)*. Morgan Kaufmann.

Stolzmann, W. (1998). Anticipatory Classifier Systems. In Koza, J., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M. H., Goldberg, D. E., Iba, H., and Riolo, R., (Eds.), *Genetic Programming*, pages 658–664. Morgan Kaufmann Publishers, Inc., San Francisco, CA.

Stolzmann, W., Lanzi, P.-L., and Wilson, S. W., (Eds.) (2002). *Proceedings of the International Workshop on Learning Classifier Systems (IWLCS2003)*, Chicago, IL.

Stone, C. and Bull, L. (2003). Towards learning classifier systems for continuous-valued online environments. In Cantú-Paz, E., Foster, J. A., Deb, K., Davis, D., Roy, R., O'Reilly, U.-M., Beyer, H.-G., Standish, R., Kendall, G., Wilson, S., Harman, M., Wegener, J., Dasgupta, D., Potter, M. A., Schultz, A. C., Dowsland, K., Jonoska, N., and Miller, J., (Eds.), *Genetic and Evolutionary Computation – GECCO-2003*, pages 1924–1925, Berlin. Springer-Verlag.

Sutton, R. S. (1990). Planning by incremental dynamic programming. In *Proceedings of the Eighth International Conference on Machine Learning*, pages 353–357, San Mateo, CA. Morgan Kaufmann.

Sutton, R. S. (1996). Generalization in reinforcement learning: Successul examples using sparse coarse coding. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., (Eds.), *Advances in Neural Information Processing Systems: Proceedings of the 1995 Conference*, pages 1038–1044, Cambridge, MA. MIT Press.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Tolman, E. C. (1932). *Purposive behavior in animals and men*. Appletown, New York.

Tomlinson, A. and Bull, L. (1998). A Corporate Classifier System. In Eiben, A. E., Bäck, T., Shoenauer, M., and Schwefel, H.-P., (Eds.), *Proceedings of the Fifth International Conference on Parallel Problem Solving From Nature – PPSN V*, number 1498 in LNCS, pages 550–559. Springer Verlag.

Tomlinson, A. and Bull, L. (2000). CXCS. In Lanzi, P.-L., Stolzmann, W., and Wilson, S. W., (Eds.), *Learning Classifier Systems: From Foundations to Applications*, pages 194–208. Springer-Verlag, Heidelberg.

Watkins, C. J. C. H. (1989). *Learning with Delayed Rewards*. PhD thesis, Psychology Department, University of Cambridge, England.

Whitley, L. D., Goldberg, D. E., Cantú-Paz, E., Spector, L., Parmee, I. C., and Beyer, H.-G., (Eds.) (2000). *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO00)*. Morgan Kaufmann.

Wilson, S. W. (1985). Knowledge Growth in an Artificial Animat. In Grefenstette, J. J., (Ed.), *Proceedings of the 1st international Conference on Genetic Algorithms and their applications (ICGA85)*, pages 16–23. L. E. Associates.

Wilson, S. W. (1994). ZCS, a Zeroth level Classifier System. *Evolutionary Computation*, 2(1):1–18.

Wilson, S. W. (1995). Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175.

Wilson, S. W. (2000). Get Real! XCS with Continuous-valued Inputs. In Lanzi, P.-L., Stolzmann, W., and Wilson, S. W., (Eds.), *LNAI 1813 : From Foundations to Applications*, pages 209–220. Springer Verlag.

Wilson, S. W. (2001). Function Approximation with a Classifier System. In Spector, L., D., G. E., Wu, A., Langdon, W. B., Voigt, H. M., and Gen, M., (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO01)*, pages 974–981. Morgan Kaufmann.

Wilson, S. W. (2004). Classifier systems for continuous payoff environments.

Wilson, S. W. and Goldberg, D. E. (1989). A critical review of Classifier Systems. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 244–255, Los Altos, California. Morgan Kaufmann.