Learning Classifier Systems: the representation point of view

Arnaud Quirin, Jerzy Korczak

***Abstract*— The paper overviews the methods and theoretical contributions of Learning Classifier Systems (LCS). The fundamental principles regarding the rule representation are described and the common representation techniques are detailed (binary, real, fuzzy and functional). A taxonomy of recent LCS is proposed using the rule representation concept as the classification criterion. The rule representations can be compared using pre-defined criteria such as accuracy, representation of the knowledge and comprehensibility of the classifiers. Three systems corresponding to the current trends are detailed and analyzed, namely XCS, LFCS and XCSL.**

***Index Terms*— Classifier systems, reinforcement learning algorithms, rule representation, data mining.**

I. Introduction

Knowledge discovery in large databases traditionally uses data mining techniques that, through an iterative process of extraction, are able to generalize data and create knowledge bases. A variety of approaches and algorithms have been applied to discover rules and create knowledge bases. More recently, beside of inductive systems [50], [49], [20], bio-inspired algorithms have been developed, notably based on neural networks [26], [5], fuzzy systems [78], [45], [44], [16], [52] and evolutionary algorithms [22], [40], [88], [35], [54], [18], [57], [48], [38], [58].

The goal of the paper is to make a synthesis of knowledge representation methods in Learning Classifiers Systems (LCS) and point out the main streams of research in this domain. The problem of designing a good representation is not trivial. Several orthogonal criteria have to be considered, manly comprehensibility and quality of the rules. For instance, a simple rule is easy to understand for a human expert, and easy to use in a new context, but the risk is that it may have a poor quality. LCS is a particular class of systems which uses *reinforcement* in conjunction with a genetic algorithm (GA) to evolve a set of rules called *a classifier set*. When expertise is not easily available and the models are not well-known, reinforcement learning is an interesting approach. In the LCS a reinforcement mechanism is based on supervised learning that interacts with an external environment [68], [7]. The system communicates with the environment by exchanging *messages* and improves itself by perceiving penalties or rewards resulted from its behavior. The messages allow the system to *interpret* the environment through its *detectors* or *sensors* (with input messages) and to *interact* with this environment using its *effectors* (with output messages). The processing of LCS is performed by *classifiers* that are rules used to model the environment by predicting the output messages from the input messages.

LSIIT/CNRS Laboratory, Boulevard Sébastien Brant, 67400 Illkirch, France. {quirin,jjk}@lsiit.u-strasbg.fr

To evaluate a given rule representation a certain number of criteria can be proposed that in general refer either to extracted classifiers or to the learning algorithm. Most of them relate to the following features:

- The *generalization accuracy* is the accuracy of an algorithm for the classification of non-learned data. This can be computed as a ratio between the number of correct classified samples and the total number of samples.
- The *comprehensibility* is the fact that a classifier is understandable by a human expert. This can be approximated by looking at the syntactic complexity of the classifier, the size of the rules, the number of attributes or relationships in the rules, ...
- The *robustness in regards to the data* is the ability to deal with noisy, incomplete, false or missing data. This can be evaluated by scoring the algorithm on real-world problems.
- The *algorithmic independence* is the ability for an algorithm to produce a rule base that can be optimized by another algorithm.
- The *representation of the knowledge*. Depending on the representation of its internal knowledge, an algorithm can reduce the data complexity.
- The *time of the learning*.
- The *stability of learned knowledge through parametrization*.
- The *determinism of the algorithms*. A deterministic algorithm learns the same rule set at each run.
- The *scalability with huge data*. This can be computed using the complexity of the algorithm in terms of several parameters, specific to the algorithm or to the data (e.g., the number of learning samples).
- The *quantity of external expertise needed*. Interactive systems need more external expertise than batch systems. The more an algorithm needs external expertise, the more it will be difficult to understand.

To introduce the LCS, a comprehensive taxonomy of the LCS is proposed (Fig. 1). The taxonomy is built using two properties: the strategy of classifier usage and the representation type. The strategy of classifier usage is illustrated from top to bottom. *Single-step* and *multi-step* problems are well known problems in the field of data mining. From the left to the right, the classifier representations used in LCS is detailed from the earliest solutions(known as *binary*, e.g. CS-1 in 1978)

to the most modern ones (known as *functional*, e.g. XCSL in 1999). It can be shown that a functional classifier is more comprehensive than a binary one.

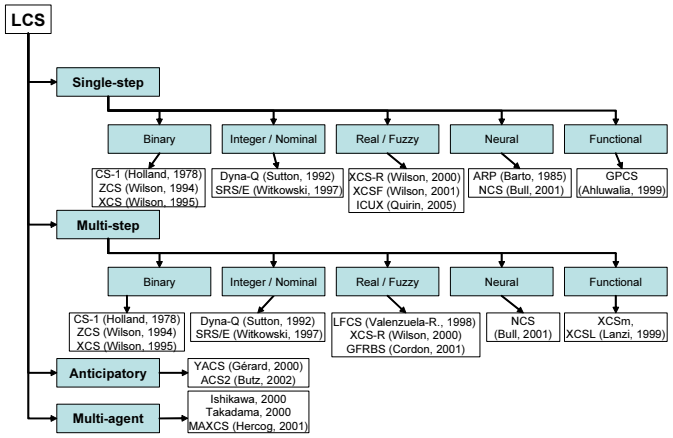

Fig. 1. A taxonomy of LCS with some examples.

The paper is structured as follows. Section II introduces the key concepts for understanding the fundamental principles of classifier systems. Section III and section IV details recent architectures. In section V some examples of LCS are presented, in particular the XCS system, fuzzy learning systems and S-classifier systems are described. Finally, in the last section, the current improvements and comments on future research in the domain are discussed.

II. Classifier Systems : fundamentals

A. Learning classifier systems

LCS are founded on *symbolic learning*. A classifier, in a similar manner to other reinforcement learning techniques, has to associate each situation with the best possible action. Each situation (m,a), where m is an environmental message (or *stimulus*) and a the corresponding action (its behavior), can be modeled as a *state*. The number of states for a complex problem can be relatively large: if each state is described by n discrete attributes, there are 2^n possible states in the case of binary attributes. With a traditional representation such as decision rules, solving a problem with a large number of states requires quite a large number of rules, each one connecting a state with a given action. The algorithmic complexity of such a situation could quickly explode. The concern of CS is that they can directly associate the action with the value of the attributes rather than the state which the system is in.

The difference between LCS and traditional learning techniques, such as inductive systems or neural networks, lies in their collaborative learning ability. A solution of a problem is represented by a sequence of related classifiers. The information is transmitted from one classifier to another as messages to interpret. According to Holland [31], the characteristics which distinguish the LCS from the CS concern the classifier activation and the knowledge base. The activation of each classifier relies on parameters that are modified in time, so as to mimick the experience acquisition. This experience is brought about by the environment: the classifiers obtain a *feedback* (or *reward*), i.e. a penalisation or a gratification according to their behavior from the environment. Furthermore, the classifiers are stored in a knowledge base which is modified during the steps of the algorithm: new classifiers are generated or suppressed, and the <*condition*> and the <*action*> parts (see the next section) of existing classifiers are recombinated. This generation of new classifiers is performed by a genetic algorithm (GA).

Let us describe the strategy of classifier usage. According to the number of runs k, the learning can be defined as *single-step* ($k = 1$) or *multi-step* ($k > 1$).

In a *single-step* learning run, the goal is to discover a function in a classification or a regression problem. The problem is solved in only one step by applying of a classifier which returns the classification decision or the function value. During the learning, the classifier activation is independent of the system's previous states and the classifier's reward is perceived until all the samples have been passed into the system. Problems of classifications are typically *single-step* if these samples are independent.

In a *multi-step* learning run, a message generated by the environment is passed from one classifier to another by activating the most suitable classifier each time. This forms an *activation chain* that is dynamic in complex environment and follows the evolution of the environment in time. In learning, the reward can be perceived at any step and the content of the activation messages depend on the previous steps.

Some data mining problems (such as classification) can better be considered as *single-step* learning, although it may be possible to view their environment as *multi-step*. In this case, the objective is to discover a sequence of relevant classifiers corresponding to the various data filters. Unfortunately, such systems are currently not solvable algorithmically.

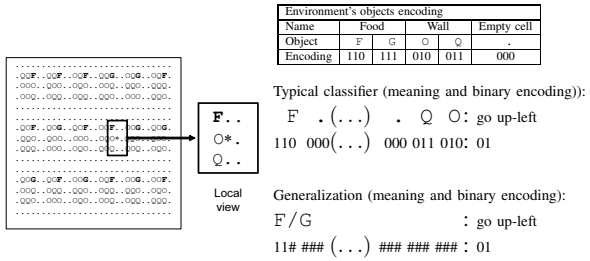

Fig. 2. Labyrinth problem.

For instance, the environment presented in Fig. 2 illustrates an one-sight robot ("*" symbol) immersed in a labyrinth. A view of a labyrinth is shown in the left part, the encoding table of the symbols is shown in the upper right part, a typical classifier is shown in the middle right part and a learned generic classifier used to solve the labyrinth is shown in the lower right part. The goal is to find food ("F" or "G" symbols) by avoiding the walls ("O" or "Q" symbols). Symbols "." are the empty cells. The robot owns a classifier base which indicates the direction of the next move according to the contents of its local environment. Classifiers are contained in a classifier base. They are encoded by considering the cells in a clockwise manner, starting from the top-left cell, using the table on the right hand side of figure 2. A typical classifier is

shown under the table. An additional symbol is introduced (#), coding for a « 0 » or a « 1 ». The "#" symbol is a *wildcard* character used to activate any symbols of $\mathcal{A} = \{0, 1\}$. It is also called the « *don't care symbol* » or the « *joker* » [79]. The goal of the system is to propose more generic classifiers that it can find, for instance by encoding the food by « 11# » rather than by « 110 » or « 111 ».

The very simple process of an environment which produces a message activating the conditions of a given rule, then the emission of the corresponding action in this same environment strongly evokes the general principle of an inference system [37]. Post has shown that such systems (called *Post Production System*) are computable in a polynomial time [56]. These rules are in the form *If ... then ... else* in high level programming languages and are used by knowledge-based systems [47]. They benefit from the fact that they are a formal model of a universal program. In fact these high level programming rules can have a high representation ability.

The knowledge base, learned and managed by the system as a *classifier population* is not a simple list of classifiers. In the Holland definition, a classifier is a rule encoding a low level knowledge of the system. It includes one or more specific conditions (the <*condition*> part) and one action (the <*action*> part). A classifier, activated by the input message, is determined by the evaluation (or *matching*) of the <*condition*> part, and the <*action*> part contains the message transmitted by the system when the <*condition*> part is satisfied. In general, two approaches have been identified:

- The Pittsburgh approach [19] in which the population is a set of rules (*rule-sets*). Thus, each individual is a distinct production system and at each generation, the best rule-set is selected and evolved.
- The Michigan approach [31] is the traditional approach, in which each individual is a distinct rule. Rules are evaluated by the GA and the populations are evolved by crossing the rules between them. This approach is not only the most traditional, and most documented in the literature, but is also that which gives the best results in general [31], [7].

There is also a third approach, less current, in which the rules are learned in an iterative way [76]. They are encoded into the chromosomes and, for each cycle of the GA, a new rule is adapted and then added to the rule base. In design, the adopted approach must be carefully selected. In both basic approaches, difficulties may appear [13]: in Michigan-like systems, the equilibrium between cooperation and individual competition is delicate, but the solution representation using independent individuals is effective for many problems. In Pittsburgh-like systems, the individuals are more complex entities and are more difficult to evolve by the genetic operators.

Formally, a classifier has the following form:

$$<\textit{condition}> : <\textit{action}> \equiv c_1, c_2, \ldots, c_i, \ldots, c_n : a \quad (1)$$

where c_i ($i \leq n$ and $n \geq 1$) is one of the binary values of the classifier <*condition*> part and a is the <*action*> part. Each of the members c_i of the <*condition*> part is a string with a fixed size k defined in a given alphabet $\mathcal{A}$. Historically, Holland used $\mathcal{A} = \{0, 1, \#\}$ [29]. As explained before, the "#" symbol is a *joker* character used to activate any symbols of $\{0, 1\}$. When a binary message is matched to the <*condition*> part of a classifier, each bit is compared one by one. If all the bits are identical, or if the message differs where the "#" symbols appear in the <*condition*> part, the <*action*> part of the classifier is used to select the next action to emit in the environment. This binary representation is quite simple and also useful for the LCS theory, notably for convergence proofs and analysis of population development such as the schema theorem [22].

The goal of a classifier system is to entirely model the function (or *mapping*) X x A, where X is the input message space and A the corresponding actions which maximize the received reward obtained from the environment.

Let us now introduce a few terms from the classifier theory. The *specificity* of a classifier is the number of instantiated values (i.e. different from the "#" symbol) in a <*condition*> part. No *joker* implies a maximal *specificity* and only *jokers* means that the *specificity* is null. When one classifier has a lower *specificity* than the other, and the instantiated values are both identical, the first classifier *subsumes* the second one. In the following example, the rule base BR1 is equivalent to both of the rule bases BR2 and BR3, but not to BR4. In this example, one can say that the classifier in BR4 *subsumes* the one in BR2.

BR1	001010 : 11100 101010 : 11100 001110 : 11100
BR2	#01010, 001110 : 11100
BR3	001#10, 101010 : 11100
BR4	#01#10 : 11100

It is important that the *specificity* is correctly managed by the genetic operators, to control the generalization level of the classifiers. This can be done, for instance, by selecting the appropriate production probabilities for each symbol in $\mathcal{A}$.

In binary representations, the "#" symbol corresponds to a "1" or to a "0". In real representations, the "#" symbol corresponds to the widest interval possible. Notice that using the intervals directly is more suitable and easier for the mutation and crossover operators. In fuzzy representations, the "#" symbol is replaced by a function equal to *true* on the whole definition domain. According to Schaeffer [66], the "#" symbol is capable of improving the performance of the rules. It can reduce the number of classifiers in the system and increase the number of solutions covered by the classifiers. So, *specificity* is related to the *comprehensibility* criterion: if a classifier contains more joker symbols, then this classifier is easier to simplify, to interpret and to understand. Moreover, the learning performance of the system is improved because the *joker* allows the concepts of *general principles* and *exceptions* [23]. Using jokers, one can also introduce the concept of hierarchization of classifiers. In [65] Riolo has shown the influence of hierarchization during the learning through the concept of *niches*. A *niche* is a set of environment states, activated by the same classifier set, sharing all the rewards.

If the solutions space is not known, the initial rules cannot be introduced by the programmer or by an expert in the field. Therefore, during the evolution, the rules selection operators (roulette wheel or tournament) bring a capital contribution. During the learning, the *hypothesis* competitiveness introduced by the <*action*> part comes directly from their past accuracy and their *specificity* [64].

In addition to the binary representation, those that are most used now are continuous representations, functional expressions (written in LISP, [42]), register conditions and auto-reinjected messages in the system[1]. In continuous representations, each condition is a sequence of real values or real intervals. In register conditions, the classifier <*condition*><*register*> : <*action*><*register*> is activated when the internal registers match <*register*>.

An interesting model has been proposed by Wilson [83]. In his model, the input message is a vector of real values and the alphabet $\mathcal{A}$ used in the <*condition*> part is replaced by intervals of real values. The classifier is then activated when all the real values of the message correspond to each real interval of the <*condition*> part. As any information can be encoded by a real value, this representation is much more powerful than the binary one and can be used in almost all applications [74], [60]. But this representation suffers from one drawback: the too-strict matching method of Wilson can prematurely eject some almost-good classifiers. For instance, the message "0.15" activates the classifier "$[0.15; 0.19]$", but not the message "0.14" in a way in which the system cannot improve the classifier (for instance, by enlarging the interval). In this case, the fuzzy representation can be applied to solve this major drawback [43], and has only recently been explored. For instance in [58], the result of the matching between a classifier and a message (actually *true* or *false*) is replaced by a score computed using a mathematical distance between the message and the intervals of the <*condition*> part. The message is viewed as a point in an N-dimensional space (N being the size of the message), the intervals are viewed as hyper-rectangles and the score corresponds to euclidian distance. This representation can solve the drawback exposed before but is computationally more complex to evolve.

The <*condition*> part allows the encoding of any elementary information, such as a color, a form, a spectrum or any other constant. This part is used during the activation of the system by the input messages. As previously, the classifier is *activated* when their conditions are satisfied. The c_i values are considered as conjunctions of the <*condition*> part. There are no disjunctions in the traditional classifier model.

The <*action*> part is the output message coding an action to be carried out when the classifier is activated. Generally, this action switches the system to the next state, for instance, the modification of the robot direction or the food ingestion. In text mining, it might be the attribution of a semantic class to a sentence. In image mining, it might be the attribution of a class to a pixel.

[1]These systems look like the initial propositions of Holland and his initial definition, but were given up because of their complexity

III. Architecture of a classifier system

An architecture of a classifier system describes its structure and the relationships between its components. The main components of LCS are the following:

- *The environment*. This component describes the problem to solve, i.e. an area in the search space in which the classifiers are evolved. It evolves and changes in an autonomous way but interacts with the LCS using a message system.
- *The detectors (or an input interface)*. These translate the current state and the current events from the environment into input messages.
- *The effectors (or an output interface)*. These allows the LCS to interact with the environment by producing some actions, delivered by the output messages from the system.
- *Several message lists* of which the interest and use are related to the characteristics of each LCS. Usually, the presence of the three following architectures is invariant:
  - A classifier base $[P]_t$ (or *Population Set*). This is the classifier set, randomly initialized before the first generation, and represents the system knowledge at the instant t. More formally, in the case of a Markovian environment, the transition from $[P]_t$ to $[P]_{t+1}$ is homogeneous[2],
  - A message list $[M]_t$ (or *Match Set*). This contains all the active messages, i.e. all the messages which match the current input of the system,
  - An action list $[A]_t$ (or *Action Set*). This contains the <*action*> parts (see the section **??**) of the active messages which will be used by the LCS to determine the message to send from the effectors.

The system uses the evolutionary paradigm to create individuals and to adapt their behavior to the environment, which generally changes itself during the learning. A GA is used in the case of a binary representation and an evolutionary algorithm (EA) in the case of a real or a fuzzy representation. The initialization, the mutation and the crossover operators are adapted for each case. Generally, new parameters are added to constrain the search space for each operator. For instance, to create new classifiers with the initialization operator in the binary case, the user needs to set two parameters: the apparition probability of a "0", compared to a "1"; and the apparition probability of a "#", compared to another symbol.

Figure 3 shows an illustration of the various components of the classifier system proposed by Wilson [79].

The following parts compose the system:

- an environment which represents the problem to solve,
- the knowledge base (Population set),
- the activated classifier list (Match set),
- the selected classifier list (Action set).

The selection of classifiers and manipulation in the various lists is performed by two algorithms:

[2]The pool at the instant $t + 1$ is entirely given in a probabilistic way by the pool at the instant t.

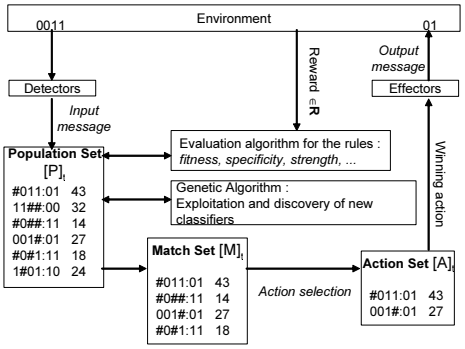

Fig. 3. Simplified model of a classifier system. Source: Wilson [79].

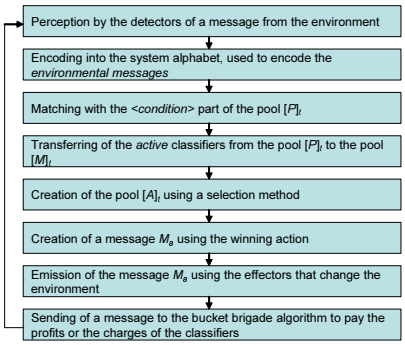

Fig. 4. *Application Mode* : interactions with the environment.

- an algorithm for the credit distribution giving good rewards at effective classifiers and penalisation at the others (*the credit repartition mechanism*),
- a GA used to evolve the classifier base (*the classifier discovery mechanism*).

One of the commonly used algorithms in the credit repartition mechanism is the *Bucket Brigade Algorithm* described by Holland in [30]. The principle of the algorithm is the following: the effective classifiers are rewarded while the others are eliminated.

IV. Interactions between components

The classifiers compete to control the system and to optimize the reaction of the environment. This reaction is given as a reward or a penalisation based on the performance or the failure of the current action with respect to the environment. From the point of view of the system, this reward cumulates with the credit (*strength*) of the classifiers having generated the action and determines, during the learning, the strength of each classifier in the competition. In fact, individuals are created during the algorithm initialization and only the *strength* evolution of each one helps to differentiate themselves. In this way, the learning will never be efficient until an evolutionary process is used to renew the population. This is carried out by a genetic algorithm, acting on the rule base [P], allowing to create new classifiers by crossover or mutation.

To evaluate the assimilated knowledge by the system, the LCS uses an evaluation function specific to the problem to solve. Usually this function uses the environment answer in a straightforward way. The classifiers accuracy is computed according to the degree of its contribution to the resolution of the problem.

R. Richards [64] details two modes in which a LCS can operate at the instant t: the *Learning Mode* and the *Application Mode*. When the system is in its learning mode, detectors perceive information from the environment, determine the relevant action then carry out actions in the environment. This process is called the *Application Mode*. The process is shown in figure 4.

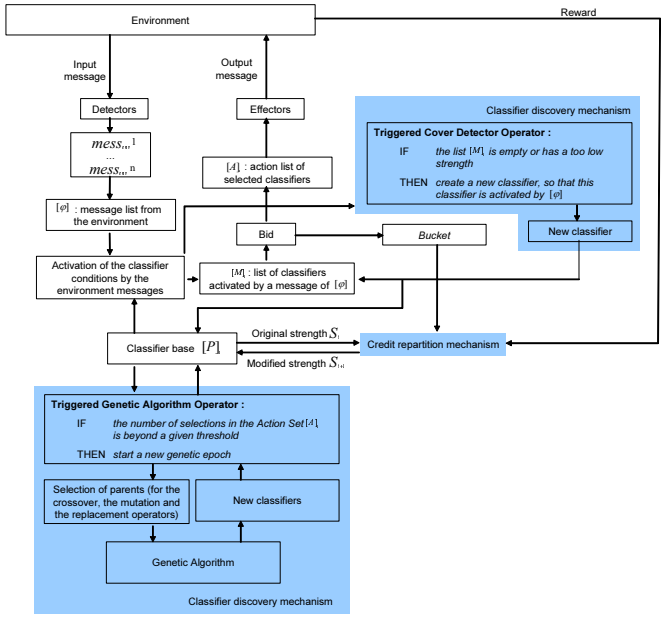

Fig. 5. Detailed messages interactions in a LCS. The mechanisms dedicated to the classifiers discovery are shown in black boxes. Source: Richards [64].

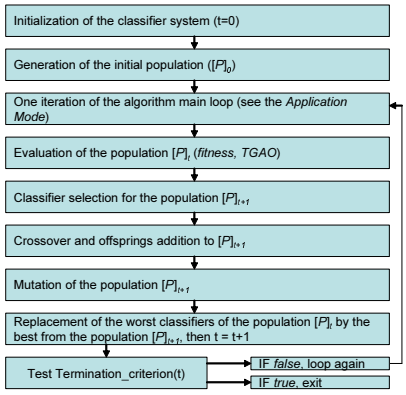

Fig. 6. *Learning Mode* : main algorithm.

The base of classifiers can be viewed as a space to test hypothesis. Here, a hypothesis is a classifier that is selected when appropriate for the current state of the system. This relevance, known as the *classifier competitiveness*, is determined by the value of the classifier *strength* during previous generations as

well as its *specificity*. The genetic algorithm has to exchange the information acquired by the classifiers. They are selected using an evaluation function and a special operator, known as a *Triggered Genetic Algorithm Operator* (TGAO). TGAO is used to control the frequency of the crossover and the mutation operators.

Figure 5 shows the main interactions between components and the message flow in the system. The messages are emitted by the environment (in the top of the figure) towards the system, which stores them in a classifier base ($[P]_t$). Then, the TGAO operator is applied to the base and new classifiers are created using the genetic algorithm during the Learning Mode (Fig. 6). Now the system has to choose an action to emit to the environment. To do this, the Bucket Brigade Algorithm elects the winning action, which pays a bid to non-activated classifiers, as described in the previous section.

The Covering Operator

The *Triggered Cover Detector Operator* (TCDO) defines when a new classifier needs to be generated. This could arise if no classifier is activated by the environmental stimulus or when the Population Set does not pass some criteria (for instance, an average strength threshold). For instance, if the list [M] is empty, i.e. when the message from the detectors $m = \{mess_{env}\ 0, \ldots, mess_{env}\ n\}$ activates no classifier in [P], the algorithm is blocked. In such a case, the population cannot be improved: the LCS might be immobilized in a local minimum and, at best, the final solution might not be relevant. Wilson [79] quotes several detection strategies for such situations and proposes methods for retrieving a complete list [M]. The corresponding operators are known as *Covering Operators*. These operators are triggered when the list is empty or when the total strength $S_{[M]_t}$ (sum of all the classifier strengths S in M at the instant t) is below the limit $\phi S_{[P]_t^\mu}$ where ϕ is a parameter and $S_{[P]_t^\mu}$ is the average strength of the population [P]. In such a situation, the *Covering Operator* has to generate a new classifier C. This is performed, in the case of a binary representation, in the following way:

- the <*condition*> part is built using exactly the message m and by adding a fixed rate $\theta_\#$ of *joker* symbols (#),
- the <*action*> part is randomly chosen,
- the strength of C is equal to the average strength of the pool [P].

In a real representation, the <*condition*> is built using random intervals. Small and large intervals correspond respectively to instanced values and *joker* symbols in the binary case. In more elaborated representations, such as the fuzzy or the functional, the <*condition*> part is always built following these three requirements:

- The message m has to activate the classifier.
- The expression is coherent in terms of the target representation.
- The generated expressions need to be diversified, so they are generated as randomly as possible. Additional *jokers* are used to fulfil the first and the last requirements.

Then, C is introduced into [P] by replacing an existing classifier, chosen using a eugenic selection operator defined by the genetic algorithm. A new pool $[M]_{t+1}$ is created and the algorithm continues in the usual way. For example, according to Wilson, $\phi = 0.5$ and $\theta_\# = 1/3$ seem to be good parameters to solve the labyrinth problem. In fact, this operator is a slightly too strong, since it directs the LCS towards a learning-by-heart and does not exploit the already learned knowledge, as does the genetic algorithm.

Nevertheless, messages inducing empty pools are considered as unavoidable for the majority of the problems, and the *Covering Operator* can be viewed as the last chance. Moreover, it allows the testing of new hypotheses, based on original messages, rather than to act in a completely random way.

V. Research in the LCS field

In this section, a few representative classifiers of the current trends will be detailed (see the taxonomy, Fig. 1), notably the XCS classifier (binary), the LFCS classifiers (fuzzy) and the S-classifiers (functional).

A. The Wilson classifier (XCS)

Usually, the *strength* parameter is used both as a prediction of the future perceived reward and as the main parameter of the evaluation function. But as Wilson [80] has noticed, in some cases, this prediction value is not correct when used in the evaluation function. To solve this problem, the XCS algorithm [80] extends the classifier system algorithms by adding several parameters into each rule, notably the reward prediction, the error of prediction, and the measure of the precision of the prediction used in the genetic algorithm. Another extension of Wilson relates to the macro-classifiers. But this concept does not refer to a different classifier representation; it is an algorithmic technique to reduce the complexity of the computing of the classifiers activated by the input messages. Table I shows the essential parameters for XCS and their recommended values, according to Wilson.

In his next paper [81], Wilson introduces two new improvements, appreciably increasing the generalization capability and the accuracy of the system:

1) the set on which the genetic operators act,
2) and the subsumption replacement.

Summing up, the XCS rules have good generalisation and robustness abilities, but they are not very comprehensive. In general, the research reports show good results with XCS compared to other systems [2], [46], [84]. In spite of the fact that XCS was first designed for multi-step problems, we have also obtained good results on single-step problems [61]. It should be noted that this model has been used in many projects (classification, object recognition, robot environment, ...), and is a base of many theoretical and optimization studies [81], [11], [12].

However, the ternary representation of XCS ($\{0, 1\#\}$) is not convenient when dealing with real-world problem. Even if this representation is still in use, new representations are now deeply explored. For instance, investigations have been done using Centre-Spread Representations (CSR) [82], Lower-Upper bound Representations (LUR) [86] and Unordered

Meaning	Ideal value
Number of classifiers in the population	50
Learning rate for the actualization of the *fitness* the error, the prediction and the size estimation of the *Action Set*	0.2
A classifier can be suppressed below this percentage of the average fitness	0.1
Triggered Genetic Algorithm Operator (TGAO) : Average number of classifiers selection in the *Action Set* before triggering the GA	5
Covering Detector Operator (CDO) : Percentage of the average strength in the *Match Set* to trigger the *Covering Operator*	0.1
The classifier precision is fixed to 1 if the error is beyond this threshold	10
Minimal lifetime of a classifier	20
Crossover probability	0.8
Mutation probability	0.15
Error reduction when a new classifier is generated by the GA	0.25
Fitness reduction when a new classifier is generated by the GA	0.1
Initial prediction of a classifier	10.0
Initial error of a classifier	0.0
Initial fitness of a classifier	0.01

TABLE I
THE MAIN PARAMETERS USED IN A XCS-LIKE CLASSIFIER. SOURCE: WILSON [79].

Bound Representations (UBR) [67], where half-open intervals $[p_i; q_i)$ encode a constraint corresponding to the i-th input variable. In the case of centre-spread, the intervals are expressed by a center and a radius; in the case of lower-upper bound, the limits are directly encoded but $p_i < q_i$ for all i and in the case of unordered bound, no restriction is made on the order of the limits. To map a genotype to a phenotype, CSR needs a truncation that is undesirable in some cases [67]. UBR addresses this issue and has shown good performance on very specific problems (for instance, the checkerboard problem), but the alternance of genes that represent *min* and *max* values decreases the efficiency of the crossover operator. In the Min Percentage Representation (MPR), the intervals are described by a minimum bound and a value corresponding to the proportion of the quantity $q_i - p_i$ compared to the maximal interval range [17]. However, even if this representation does not suffer from the truncation bias or the semantic alternance of genes, theses systems are restricted to very specific problems like real multiplexer or checkerboard problems because of their limited phenotypic expression (hyper-rectangles in the space of data)[67].

B. Classifiers and fuzzy logic

To overcome this issue, more elaborate phenotypic expressions for the classifiers have been explored, and among them, the fuzzy logic. These systems, called *Learning Fuzzy Classifier System* (LFCS) [62], are able to deal with classifiers encoding continuous variables. A linguistic interpretation of these classifiers can also be proposed. This paradigm is known in the literature under the generic name of *Genetic Fuzzy System* (GFS), where the most sophisticated systems are probably the *Genetic Fuzzy Rule-Based Systems* (GFRBS) [15], [72], [55]. The integration of fuzzy logic allows the use of certainty factors in *natural language* expressions such as heat, cold, reliable, indisputable, not very sure, etc.

Concerning the representation of knowledge expressed by the classifiers, this can be divided into rule base and database representations. The *rule base* contains all the fuzzy rules. Using genetic algorithms to learn new rules requires the integration of fuzzy membership functions. The literature has considered all three possible approaches (see the section II-A): the Michigan-like approach [34], the Pittsburgh-like approach [28] and the iterative approach [24]. In the Pittsburgh approach, the representation most used for the classifier populations are the relational matrices and the decision tables. In the case of the Michigan approach, the classifiers are encoded in simple rule lists. To encode an individual rule, bit strings with fixed length are commonly employed [24], or in a better way using *messy-coding* (see the next section). In the case of the iterative approach, the best rule is selected each time. This rule is incorporated into the current set of rules. To obtain new and different rules, the selected rule is penalized by eliminating the samples covered by this rule and the process is repeated [24]. The *database* representation contains the definitions of the scale factors and the membership functions for the associated fuzzy set, with the linguistic terms used. Contrary to the rule base, the learning of a database is more complex [75]. Indeed, the components representation of a database and consequently the search space are heterogeneous. In this case, it might be more suitable to use optimization techniques, for instance *tuning*, to optimize an existing database instead of discovering a new one, i.e. to find the optimal parameters for the membership functions or the scale factors.

The algorithm shown in figure 7 describes the general process of a LFCS [73].

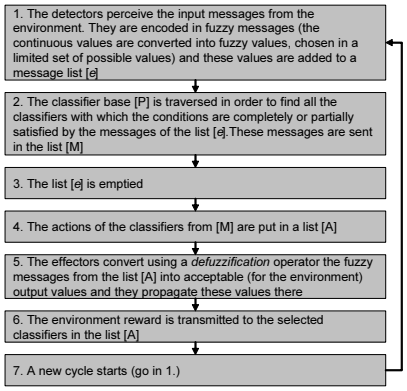

Fig. 7. The process of a LFCS algorithm.

LFCS seem to produce comprehensive classifiers, and the representation of the knowledge seems to be higher, compared to others LCS. However, it appears that the stability of the algorithms through the parametrization is not so good. For instance, Casillas [14] describes a theoretical study (i.e. without real case studies) about using Mamdani-type fuzzy classifiers instead of the traditional binary or real representation. In

general, fuzzy-based systems use cooperative inference to generate the fuzzy output: all the rules that are fired are used under an implication, an aggregation and a defuzzification operator [89], while classifier systems have a competitive behavior. In his proposal, Casillas studies several implication operators showing the effect on the results. In spite of the simple fuzzy representation used (compared to those used in fuzzy-based systems), specific cares about the parametrization are still needed.

Nevertheless, LFCS systems enable the use of fuzzy membership functions and allow fuzzy approximation of the input and, more important, ouput variables rather than using an exact activation function such as the LCS. Now, in spite of the apparent power of these classifiers, it is not obvious that the LFCS-like systems are able to build effective rule chains of a given size, for instance to solve *multi-step* problems. It seems, according to Furuhashi [21], that when new classifiers are sought by the system, the inaccuracy transferred from a classifier to another by the fuzzy variables explodes and that nothing can be deduced any longer by the system.

C. The S-classifiers

The binary representations of the classifiers have two main drawbacks: firstly, sometimes the binary encoding of the detectors is not well suited when the environment structure is continuous. Secondly, there is a fixed mapping between the bit positions in a classifier <*condition*> part and the bit positions of the detectors, that prohibits variable size messages.

Up until now, the applications solved by classifier systems have been sufficiently satisfied with a binary representation, because the learning capability of the system was regarded as more important than its generalization capability. Lanzi realized that the representation problem becomes a question of increasing interest. In his two papers ([41] and [42]), he proposed two new approaches for the classifiers formalism, by introducing the *S-classifiers*.

The first, called *messy-coding*, is exempted from the fixed mapping between the bit positions in the <*condition*> part and the bit positions of the detectors. This technique allows the conditions to have a variable size: the genes are divided into several independent blocks. To align each classifier block with each detector, each block is mapped to a *tag* which itself refers to a detector. The model allows a classifier with two genes having the same *tag* (over-specified classifier) or missing genes (under-specified classifier). The generalization property using *joker* symbols is now naturally expressed by under-specified classifiers. In the *messy-coding*, the representation of a classifier is a variable-sized list of <variable, value> pairs, in which the order does not have any importance. All this work has resulted to the implementation of XCSm [41], a suitable version of XCS for the *messy-coding*, in which the *covering* operator, the *activation* operator and the genetic operators have been rewritten.

The second approach, suggested by Lanzi, goes in the same direction of classifier representation improvement. The system *eXtended Classifier System in LISP* (XCSL) [42] is an extension based on the S-expressions of the LISP language. Although these expressions are written in a linear way, the genetic operators manipulate them as trees to reduce the evaluation time. But in his paper, Lanzi only considered simple problems for his XCSL system: the solutions have to be expressed by boolean functions. The <*condition*> parts of the classifiers are compositions of the logical operators *AND*, *OR* and *NOT*.

```
<cond> :=   "(" NOT <cond> ")"
          | "(" AND <cond> <cond> ")"
          | "(" OR  <cond> <cond> ")"
          | <var>
<var> := "X0" | "X1" | "X2"
```

Fig. 8. The BNF grammar of the XCSL classifiers.

All the classifier conditions of XCSL are generated by the BNF grammar [53] presented in figure 8. These conditions fit in a boolean functions subset of three input variables. In the figure, the terminal symbols are represented between quotation marks and the functional symbols between brackets. This grammar was used for the boolean function problem[3]. For instance, « (OR S1 S2) » is the expression of a complete condition, where S1 and S2 are environment-dependent terminals (a boolean variable in the multiplexer problem, a predicate for the orientation sensors in the labyrinth problem, ...).

There exists the same differences compared with XCS, as in the first approach. Some operators are modified in a suitable way:

- The *activation* operator does not present a particular difficulty, and the expression is just evaluated observing the traditional LISP syntax,
- The *covering* operator, used to create a new classifier with random condition when no classifier is activated by the input message, requires some attention. Lanzi realized that to avoid an over-fitting problem, induced by an under-specialization of the classifier genes, it is better to build, for the condition, a conjunction of three expressions, each one being activated by the input message. Each one of these sub-expressions is a disjunction of an arbitrary number of bits from the input message, except for the first sub-expression which stores the exact message. This representation guarantees a specific classifier for the input message, while the two last sub-expressions introduce a generalization capability. Unfortunately, the author does not explain why the choice of *three* sub-expressions gives optimal results in his experiments.
- *The genetic operators*: In XCSL, the GA is applied to the *Action Set* [81] (see the section V-A). Two classifiers are selected with probabilities proportional to their *fitness*, copied, combined with the probability χ, mutated with the probability μ and reinserted in their *Action Set*. Genetic programming principles [39] are used for the crossover and the mutation operators: the crossover exchanges two sub-trees in the condition part, and the mutation replaces a sub-tree by a randomly-created one.

[3]Function f of n variables $(x_0, \ldots, x_n)$ defined by $f : \{0,1\}^n \rightarrow \{0,1\}$.

Discussing the classifier representation, Lanzi points out a subtle problem occurring with the OR operator just-defined, which neither occurs with the other boolean operators, nor even in the typical XCS binary classifiers. He shows that a higher representation level introduces a bias which lets the genetic algorithm create corrupted classifiers and leads to an instability of the system (in fact, the total accuracy measurement will oscillate[4]). A suggested solution may be in a random reordering of the evaluations of the conditions containing the OR operator, during a specific step of the genetic generations.

This system was also tested in the *multi-step* environments such as labyrinths, in which the OR-function bias proved to be more restrained. This may be explained by the fact that in a *single-step* problem the over-general conditions are more frequent because the search space is more deeply explored. In a labyrinth, a condition is never active over a long time period because the input message changes frequently, as the agent moves within its environment.

The XCS and the XCSL systems were tested in quasi-similar environments, both for *single-step* and for *multi-step* problems, allowing the experimental comparison of their performances. Optimal performances were obtained in both cases on all kind of problems with a restricted number of learning steps. The classifiers of XCSL seem to be comprehensive and the representation of the knowledge allows to export them for human experts, or to re-learn them in another algorithm. However, many problems still need to be solved, with the emergent apparition of the LISP formalism. Currently, this formalism is undoubtedly too little explored in the domain of classifier systems. Among these problems, we can quote the computation of the generalization degree (or, on the contrary, the *specificity*) of a classifier. With binary representation, this is a trivial problem: only the *joker* symbols need to be counted. With the S-expressions, this problem is much more complex (NP-complete if we need to compare all the nodes of each tree). This strongly restricts the use of some improvements suggested by Wilson [81], in particular the subsumers removal, unless in the presence of some particulary efficient heuristics. Another problem is the increase in complexity of the classifier conditions, which require more processor time to be computed. Again, heuristics for simplification and/or expressions evaluation might be developed.

[4]To illustrate, let (a) and (b) be two classifiers defined as:
(a) **if** C1 **then** action A
(b) **if** C2 **then** action A
where C1 and C2 are conditions derived from the `<cond>` symbol of the BNF grammar presented in figure 8. In fact, these classifiers are similar in any point to the following classifier (c), because this classifier applies in the same situations as the classifiers (a) and (b) and returns the same action A:
(c) **if** C1 **or** C2 **then** action A
If C2 is slightly more a specific condition than C1, we can imagine that, for a relatively long time period, the input message activates the C1 condition more frequently than C2. If the system uses the classifiers (a) and (b), the accuracy of classifier (b) will decrease and it will be finally dropped. On the other hand, if the system uses the classifier (c), the C2 condition will not influence the growth of the classifier accuracy, even if the C2 condition is completely irrelevant, or false. In this case, the algorithm can create ineffective classifiers, without be informed about that. Such a condition is called by Lanzi a *hidden condition* by the OR operator.
From this, he concluded that some condition parts of a classifier might be altered without negatively influencing its accuracy (and consequently without these classifiers or conditions being able to be eliminated) as long as all the parts of the conjunction have not been checked.

VI. Conclusions and future directions

In this paper, a panorama of classifier systems has been overviewed and analyzed, from the representation point of view. The criteria, such as the *comprehensibility*, the *algorithmic independence* or the *scalability with huge data* have been used to compare the extracted classifiers or the learning algorithm itself. A taxonomy of classifier systems has been then proposed, in which the representation criterion is detailed. Some examples of recent LCS are given, using representations known as binary, integer, nominal, real, fuzzy, neural and functional. To introduce the discussion about the recent LCS, a simplified model of a LCS has been presented, and the communication of classifiers and messages from one component to another was also detailed. We have seen that in LCS, two main operators are used to evolve the classifiers: the Cover Operator (TCDO) for the initialisation, and the Genetic Operator (TGAO) for the discovery of new classifiers and the exploitation of existing classifiers.

Three classifier systems have been detailed: the Wilson classifier XCS, LFCS, and XCSL. New directions are now explored, considering multi-agent problems [27], [33], [36], [70] and anticipatory learning [10], [25]. These systems are able to learn *latently* (i.e. learning without getting a reward) a complete mapping of the environment. The representation is then extended in consequence (for instance, the *expectation part* in anticipatory learning, a population of classifiers for MAXCS, ...). However, these systems have not yet been tested on real problems.

Presently, new models are being developed in this direction. The new models are based on the concept of *niching*, such as the *Implicit Niching* model or the COGIN method. *Niching* promotes the co-evolution of the individuals. Distributed models promoting the independent evolution of individuals also appears, such as the Island Model Gentic Algorithm model. The main interest of this model is that it can be very easily paralleled: at the end of n iterations, $x\%$ of the population is sent to the closest processor, which replaces the same part of its population. With one island for each processor, the whole unit can deal with the complete population in a simultaneous way.

Modern applications now show a growing interest in the LCS field. The extraction of knowledge in a form of comprehensive and readable rules is used in many real-world applications, including, for instance, classification ([4], [74], [77]), object recognition ([60], [59]), robot problems ([79], [51], [6]), steel production [8] and even poker [63].

Acknowledgements

The authors would like to thank Paul Montgomery, from the Louis Pasteur University, Strasbourg, France, for dedicating his time to improve the first version of this paper.

REFERENCES

[1] M. Ahluwalia and L. Bull. A genetic programming classifier system. In *Proceedings of the Genetic and Evolutionary Computation Conference – GECCO-99*, pages 11–18, Morgan Kaufmann, 1999.

[2] A. J. Bagnall and G. C. Cawley. Learning classifier systems for data mining: A comparison of XCS with other classifiers for the forest cover dataset. In *Proceedings of the IEEE/INNS International Joint Conference on Artificial Neural Networks (IJCNN-2003)*, Portland, 2003.

[3] A. G. Barto and P. Anandan. Pattern-recognizing stochastic learning automata. *IEEE Transactions on Systems, Man and Cybernetics*, 15:360–375, 1985.

[4] E. Bernadó-Mansilla and J. M. Garrell-Guiu. Accuracy-based learning classifier systems: Models, analysis and applications to classification tasks. *Evolutionary Computation*, 11(3):209–238, 2003.

[5] C. Bishop. *Neural Networks for Pattern Recognition*. Oxford: University Press, 1995.

[6] A. Bonarini. Fuzzy and crisp representations of real-valued input for learning classifier systems. In *Proceedings of Genetic and Evolutionary Computation Conference 99 (GECCO 99)*, pages 52–59, Morgan Kaufmann, San Francisco, 1999.

[7] L. B. Booker, D. E. Goldberg, and J. H. Holland. Classifier systems and genetic algorithms. *Artificial Intelligence*, 40:235–282, 1989.

[8] W. Browne, K. Holford, C. Moore, and J. Bullock. An industrial learning classifier system: The importance of pre-processing real data and choice of alphabet. *Engineering Applications of Artificial Intelligence*, 13(1):25–36, 2000.

[9] L. Bull and T. O'Hara. An accuracy-based neural classifier system. Technical report, UWE Learning Classifier Systems Group Technical Report - UWELCSG01-008, 2001.

[10] M. V. Butz. *Anticipatory Learning Classifier Systems*. Volume 4 of the Series Genetic Algorithms and Evolutionary Computation GENA, Kluwer Academic Publishers, 2002.

[11] M. V. Butz and D. E. Goldberg. Bounding the population size in XCS to ensure reproductive opportunities. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2003)*, 2003.

[12] M. V. Butz, D. E. Goldberg, P. L. Lanzi, and K. Sastry. Bounding the population size to ensure niche support in XCS. Technical Report 2004033, Department of General Engineering The University of Illinois, Urbana-Champaign, 2004.

[13] B. Carse and A. G. Pipe. X-FCS: A fuzzy classifier system using accuracy based fitness - first results. In *Proceedings of the International Conference on Fuzzy Logic and Technology (EUSFLAT)*, pages 195–198, 2001.

[14] J. Casillas, B. Carse, and L. Bull. Fuzzy-xcs: an accuracy-based fuzzy classifier system. In *Proceedings of the XII Congreso Español sobre Tecnología y Lógica Fuzzy (ESTYLF 2004)*, pages 369–376, 2004.

[15] O. Cordon, F. Herrera, F. Gomide, F. Hoffmann, and L. Magdalena. Ten years of genetic fuzzy systems: Current framework and new trends. In *IFSA/NAFIPS*, Vancouver, 2001.

[16] O. Cordón and F. Herrera. *Genetic Algorithms in Engineering and Computer Science*, chapter A general study on genetic fuzzy systems, pages 33–57. John Wiley and Sons, 1995.

[17] H. H. Dam, H. A. Abbass, and C. Lokan. Be real! xcs with continuous-valued inputs. In *Proceedings of the 2005 Genetic And Evolutionary Computation Conference*, pages 85–87, 2005.

[18] I. De Falco, A. Della Cioppa, and E. Tarantino. Discovering interesting classification rules with genetic programming. *Applied Soft Computing*, 1(4):257–269, 2001.

[19] K. A. DeJong. Using genetic algorithms to learn task programs: the Pitt Approach. *Machine Learning*, 2(2-3), 1988.

[20] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. From data mining to knowledge discovery: An overview. In *Advances in Knowledge Discovery and Data Mining*, pages 1–34, Menlo Park, 1996.

[21] T. Furuhashi, K. Kakaoka, K. Morikawa, and Y. Uchikawa. Controlling excessive fuzziness in a fuzzy classifier system. In *Proceeedings of the Fourth International Conference on Genetic Algorithms*, 1993.

[22] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Mass, 1989.

[23] D. E. Goldberg. *Algorithmes génétiques, exploration, optimisation et apprentissage automatique*. Addison-Wesley, Paris, 1994.

[24] A. Gonzalez and R. Perez. Slave: a genetic learning system based on an iterative approach. *IEEE Transactions on Fuzzy Systems*, 7(2):176–191, 1999.

[25] P. Gérard, W. Stolzmann, and O. Sigaud. Yacs : a new learning classifier system using anticipation. *Journal of Soft Computing*, 6(3-4):216–228, 2002.

[26] S. Haykin. *Neural Networks*. Prentice Hall (seconde édition), 1999.

[27] L. M. Hercog and T. C. Fogarty. Analysis of inductive intelligence in XCS-based multi-agent system (MAXCS). In *In J.Periaux, P. Joly, and E. Onate, editors, Innovative Tools for Scientific Computation in Aeronautical Engineering*, pages 351–366, CIMNE, Barcelona, 2001.

[28] F. Hoffmann and G. Pfister. Evolutionary design of a fuzzy knowledge base for a mobile robot. *International Journal of Approximate Reasoning*, 17(4):447–69, 1997.

[29] J. H. Holland. *Adaptation in natural and artificial systems*. Cambridge: MIT Press, 1975.

[30] J. H. Holland. Properties of the bucket brigade. In *Proceedings of the International Conference on Genetic Algorithms*, Hillsdale, 1985.

[31] J. H. Holland. *Machine learning: An artificial intelligence approach, vol. 2*, chapter Escaping brittleness: The possibilities of general purpose learning algorithms applied to parallel rule-based systems. Morgan Kaufmann, San Mateo, 1986.

[32] J. H. Holland and J. S. Reitman. *Pattern-directed inference systems*, chapter Cognitive systems based on adaptive algorithms. New York: Academic Press, 1978.

[33] N. Ireson, Y. J. Cao, L. Bull, and R. Miles. A communication architecture for multi-agent learning systems. In *Proceedings of the EvoNet Workshops - EvoTel 2000*, pages 255–266, 2000.

[34] H. Ishibuchi, T. Nakashima, and T. Murata. Performance evaluation of fuzzy classifier systems formultidimensional pattern classification problems. *IEEE Transactions on Systems, Man and Cybernetics - Part B: Cybernetics*, 29:601–618, 1999.

[35] H. Ishibuchi, K. Nozaki, and N. Yamamoto. Selecting fuzzy rules by genetic algorithm for classification problems. In *Second IEEE International Conference on Fuzzy Systems*, pages 1119–1124, 1993.

[36] Y. Ishikawa and T. Terano. Co-evolution of multiagents via organizational-learning classifier system and its application to marketing simulation. In *Proceedings of the 4th Pacific-Asia Conf. on Information Systems (PACIS-2000)*, pages 1114–1127, 2000.

[37] L. Jouffe. Actor-critic learning based on fuzzy inference system. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pages 339–344, Beijing, 1996.

[38] E. Kentala, J. Laurikkala, I. Pyykko, and M. Juhola. Discovering diagnostic rules from a neurotologic database with genetic algorithms. *Annals of otology, rhinology, and laryngology*, 108(10):948–954, 1999.

[39] J. R. Koza. *Genetic Programming - On the programming of computers by means of natural selection*. Cambridge: The MIT Press/Bradford Books, 1992.

[40] J. R. Koza, F. H. Bennett, D. Andre, and M. A. Keane. *Genetic Programming III: Darwinian Invention and Problem Solving*. Morgan Kaufmann, 1999.

[41] P. L. Lanzi. Extending the representation of classifier conditions. Part I: From binary to messy coding. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, pages 337–344, 1999.

[42] P. L. Lanzi. Extending the representation of classifier conditions. Part II: From messy coding to s-expressions. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, pages pages 345–352, 1999.

[43] M. Laviolette and J. W. Seaman. The efficacy of fuzzy representations of uncertainty. *IEEE Transactions Fuzzy Systems*, 2(1):4–15, 1994.

[44] M. H. Lim, S. Rahardja, and B. H. Gwee. A ga paradigm for learning fuzzy rules. *Fuzzy Sets and Systems*, 82(2):177–186, 1996.

[45] Y. Lin and G. A. Cunningham III. A new approach to fuzzy-neural system modeling. *IEEE Transaction on Fuzzy Systems*, 3(2):190–198, 1995.

[46] E. B. Mansilla and T. K. Ho. Domain of competence of xcs classifier system in complexity measurement space. *IEEE Transaction on Evolutionary Computation*, 9(1):82–104, 2005.

[47] Y. Marchand and R. I. Damper. A multi-strategy approach to improving pronunciation by analogy. *Computational Linguistics*, 26(2):195–219, 2000.

[48] A. R. McCallum and K. A. Spackman. Using genetic algorithms to learn disjunctive rules from examples. In *Proceedings of the seventh international conference on Machine Learning*, pages 149–152, 1990.

[49] R. S. Michalski. A theory and methodology of inductive learning. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 83–134. Springer, Berlin, Heidelberg, 1984.

[50] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[51] D. E. Moriarty and R. Miikkulainen. Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, 22:11–33, 1996.

[52] D. Nauck and R. Kruse. A neuro-fuzzy method to learn fuzzy classification rules from data. *Fuzzy Sets and Systems*, 89(3):277–288, 1997.

[53] P. Naur. Revised report on the algorithmic language algol 60. In *Communications of the ACM, Vol. 3, No.5*, pages 299–314, 1960.

[54] E. Noda, A. A. Freitas, and H. S. Lopes. Discovering interesting prediction rules with a genetic algorithm. In *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 1322–1329, 1999.

[55] A. Parodi and P. Bonelli. A new approach to fuzzy classifier systems. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 223–230, San Mateo: Morgan Kaufmann, 1993.

[56] E. L. Post. Formal reductions of the general combinatorial decision problem. *American Journal of Math*, 52:264–268, 1943.

[57] M. A. Potter, K. A. De Jong, and J. J. Grefenstette. A coevolutionary approach to learning sequential decision rules. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 366–372, 1995.

[58] A. Quirin and J. J. Korczak. Representation of genetic individuals for unmixing multispectral data. In *2005 IEEE Congress on Evolutionary Computation (CEC 2005)*, Edinburg, 2005.

[59] A. Quirin and J. J. Korczak. *Genetic and Evolutionary Computation in Image Processing and Computer Vision*, chapter Discovering of Classification Rules from Hyperspectral Images. Springer-Verlag, LNCS Series, vol. 2936, 2006.

[60] A. Quirin, J. J. Korczak, M. V. Butz, and D. E. Goldberg. Learning classifier systems for hyperspectral images processing. Technical Report ULP-LSIIT-RR-2004-01, Laboratoire des Sciences de l'Image, de l'Informatique et de la Télédétection, CNRS, Université Louis Pasteur, Illkirch, 2004.

[61] A. Quirin, J. J. Korczak, M. V. Butz, and D. E. Goldberg. Analysis and evaluation of learning classifier systems applied to hyperspectral image classification. In *5th International Conference on Intelligent Systems Design and Applications (ISDA 2005)*, pages 280–285, Wroclaw, 2005.

[62] M. V. Rendon. Reinforcement learning in the fuzzy classifier system. Technical report, Institut Technologique des Etudes Supérieures, Monterrey, 1997.

[63] C. Reveley. A learning classifier system adapted for hold'em poker. Master's thesis, Birkbeck College, University of London, 2002.

[64] R. A. Richards. *Classifier Systems & Genetic Algorithms*, chapter Zeroth-Order Shape Optimization utilizing a Learning Classifier System, document available at http://www.stanford.edu/~buc/SPHINcsX/book.html. 2001.

[65] R. L. Riolo. *Empirical Studies of Default Hierarchies and Sequences of Rules in Learning Classifier Systems*. PhD thesis, Computer Science and Engineering Department, University of Michigan, 1988.

[66] J. Schaeffer and D. Schuurmans. Representational difficulties with classifier systems. In *International Conference of Genetic Algorithms*, pages 328–333, 1989.

[67] C. Stone and L. Bull. For real! xcs with continuous-valued inputs. *Evolutionary Computation*, 11(3):299–336, 2003.

[68] R. Sutton and A. Barto. *Reinforcement learning: An introduction*. The MIT Press, 1998.

[69] R. S. Sutton. Reinforcement learning architectures. In *Proceedings ISKIT'92 International Symposium on Neural Information Processing*, Fukuoka, 1992.

[70] K. Takadama, H. Inoue, M. Okada, K. Shimohara, and O. Katai. Agent architecture based on interactive self-reflection classifier system. *International Journal of Artificial Life and Robotics (AROB)*, 2001.

[71] K. Takadama, T. Terano, and K. Shimohara. Learning classifier systems meet multiagent environments. In *Proceedings of the International Workshop on Learning Classifier Systems (IWLCS-2000), in the Joint Workshops of SAB 2000 and PPSN 2000*, 2000. Extended abstract.

[72] M. Valenzuela-Rendón. The fuzzy classifier system: a classifier system for continuously varying variables. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 346–353, San Mateo: Morgan Kaufmann, 1991.

[73] M. Valenzuela-Rendón. Reinforcement learning in the fuzzy classifier system. *Expert Systems with Applications*, 14(1-2):237–247, 1998.

[74] G. Valigiani, C. Fonlupt, and P. Collet. Analysis of GP improvement techniques over the real-world inverse problem of ocean colour. In *EUROGP'04*, Coimbra, 2004.

[75] J. R. Velasco. Genetic-based on-line learning for fuzzy process control. *International Journal of Intelligent Systems*, 13(10-11):891–903, 1998.

[76] G. Venturini. SIA: a supervised inductive algorithm with genetic search for learning attribute based concepts. In *Proceedings of the European Conference on Machine Learning*, pages 280–296, Viena, 1993.

[77] D. Walter and C. K. Mohan. ClaDia: A fuzzy classifier system for disease diagnosis. In *Proceedings of the 2000 Congress on Evolutionary Computation (CEC00)*, pages 1429–1435, 2000.

[78] L.-X. Wang and J. M. Mendel. Generating fuzzy rules by learning from examples. *IEEE Transactions on Systems, Man and Cybernetics*, 22(6):1414–1427, 1992.

[79] S. W. Wilson. ZCS: A zeroth level classifier system. *Evolutionary Computation*, 2(1):1–18, 1994.

[80] S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.

[81] S. W. Wilson. Generalization in the XCS classifier system. In *Proceedings of the Third Annual Genetic Programming Conference*, pages 665–674, Madison (WI), Morgan Kaufmann, San Francisco, 1998.

[82] S. W. Wilson. Get real! xcs with continuous-valued inputs. In *Learning Classifier Systems. From Foundations to Applications, Lecture Notes in Artificial Intelligence (LNAI-1813)*, pages 209–219, 2000.

[83] S. W. Wilson. *Learning Classifier Systems: From Foundations to Applications, LNAI 1813*, chapter Get real! XCS with continuous-valued inputs, pages 209–220. 2000.

[84] S. W. Wilson. Mining oblique data with XCS. *Lecture Notes in Computer Science*, 1996:158–176, 2000.

[85] S. W. Wilson. Function approximation with a classifier system. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 974–981, San Francisco, 2001.

[86] S. W. Wilson. Mining oblique data with xcs. In *Advances in Learning Classifier Systems. Proceedings of the Third International Workshop (IWLCS-2000), Lecture Notes in Artificial Intelligence (LNAI-1996)*, pages 158–174, 2001.

[87] C. M. Witkowski. *Schemes for Learning and Behaviour: A New Expectancy Model*. PhD thesis, University of London, 1997.

[88] Y. Yuan and H. Zhuang. A genetic algorithm for generating fuzzy classification rules. *Fuzzy Sets and Systems*, 84(1):1–19, 1996.

[89] . A. Zadeh. *Fuzzy Sets, Fuzzy Logic, and Fuzzy Systems: Selected Papers*. World Scientific Publishing, 1996.