Hindawi Publishing Corporation
Journal of Artificial Evolution and Applications
Volume 2009, Article ID 736398, 25 pages
doi:10.1155/2009/736398

Review Article

Learning Classifier Systems: A Complete Introduction, Review, and Roadmap

Ryan J. Urbanowicz and Jason H. Moore

Department of Genetics, Dartmouth College, Hanover, NH 03755, USA

Correspondence should be addressed to Jason H. Moore, jason.h.moore@dartmouth.edu

Received 24 November 2008; Accepted 23 June 2009

Recommended by Marylyn Ritchie

If complexity is your problem, learning classifier systems (LCSs) may offer a solution. These rule-based, multifaceted, machine learning algorithms originated and have evolved in the cradle of evolutionary biology and artificial intelligence. The LCS concept has inspired a multitude of implementations adapted to manage the different problem domains to which it has been applied (e.g., autonomous robotics, classification, knowledge discovery, and modeling). One field that is taking increasing notice of LCS is epidemiology, where there is a growing demand for powerful tools to facilitate etiological discovery. Unfortunately, implementation optimization is nontrivial, and a cohesive encapsulation of implementation alternatives seems to be lacking. This paper aims to provide an accessible foundation for researchers of different backgrounds interested in selecting or developing their own LCS. Included is a simple yet thorough introduction, a historical review, and a roadmap of algorithmic components, emphasizing differences in alternative LCS implementations.

1. Introduction

As our understanding of the world advances, the paradigm of a universe reigned by linear models, and simple "cause and effect" etiologies becomes staggeringly insufficient. Our world and the innumerable systems that it encompasses are each composed of interconnected parts that as a whole exhibit one or more properties not obvious from the properties of the individual parts. These "complex systems" feature a large number of interacting components, whose collective activity is nonlinear. Complex systems become "adaptive" when they possess the capacity to change and learn from experience. Immune systems, central nervous systems, stock markets, ecosystems, weather, and traffic are all examples of complex adaptive systems (CASs). In the book "Hidden Order," John Holland specifically gives the example of New York City, as a system that exists in a steady state of operation, made up of "*buyers, sellers, administrations, streets, bridges, and buildings* [that] *are always changing. Like the standing wave in front of a rock in a fast-moving stream, a city is a pattern in time.*" Holland conceptually outlines the generalized problem domain of a CAS and characterizes how this type of system might be represented by rule-based "agents" [1]. The term "agent" is used to generally refer to a single component of a given system. Examples might include antibodies in an immune system, or water molecules in a weather system. Overall, CASs may be viewed as a group of interacting agents, where each agent's behavior can be represented by a collection of simple rules. Rules are typically represented in the form of "IF *condition* THEN *action*". In the immune system, antibody "agents" possess hyper-variable regions in their protein structure, which allows them to bind to specific targets known as antigens. In this way the immune system has a way to identify and neutralize foreign objects such as bacteria and viruses. Using this same example, the behavior of a specific antibody might be represented by rules such as "IF the antigen-binding site fits the antigen THEN bind to the antigen", or "IF the antigen-binding site does not fit the antigen THEN do not bind to the antigen". Rules such as these use information from the system's environment to make decisions. Knowing the problem domain and having a basic framework for representing that domain, we can begin to describe the LCS algorithm. At the heart of this algorithm is the idea that, when dealing

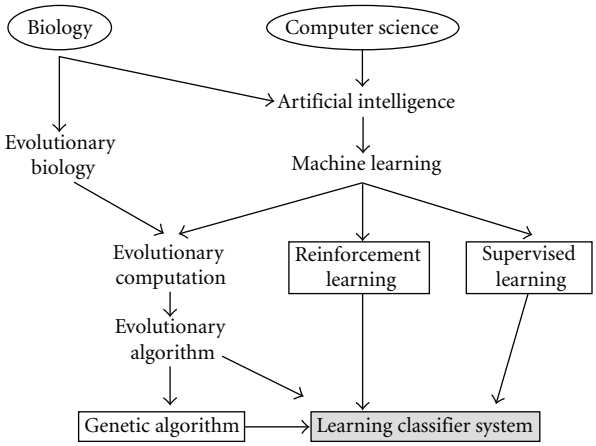

Figure 1: Field tree—foundations of the LCS community.

with complex systems, seeking a single best-fit model is less desirable than evolving a population of rules which collectively model that system. LCSs represent the merger of different fields of research encapsulated within a single algorithm. Figure 1 illustrates the field hierarchy that founds the LCS algorithmic concept. Now that the basic LCS concept and its origin have been introduced, the remaining sections are organized as follows: Section 2 summarizes the founding components of the algorithm, Section 3 discusses the major mechanisms, Section 4 provides an algorithmic walk through of a very simple LCS, Section 5 provides a historical review, Section 6 discusses general problem domains to which LCS has been applied, Section 7 identifies biological applications of the LCS algorithm, Section 8 briefly introduces some general optimization theory, Section 9 outlines a roadmap of algorithmic components, Section 10 gives some overall perspective on future directions for the field, and Section 11 identifies some helpful resources.

2. A General LCS

Let us begin with a conceptual tour of LCS anatomy. As previously mentioned, the core of an LCS is a set of rules (called the population of classifiers). The desired outcome of running the LCS algorithm is for those classifiers to collectively model an intelligent decision maker. To obtain that end, "*LCSs employ two biological metaphors; evolution and learning...* [where] *learning guides the evolutionary component to move toward a better set of rules.*" [2] These concepts are respectively embodied by two mechanisms: the genetic algorithm, and a learning mechanism appropriate for the given problem (see Sections 3.1 and 3.2 resp.). Both mechanisms rely on what is referred to as the "environment" of the system. Within the context of LCS literature, the environment is simply the source of input data for the LCS algorithm. The information being passed from the environment is limited only by the scope of the problem being examined. Consider the scenario of a robot being asked to navigate a maze environment. Here, the input data may be in the form of sensory information roughly describing the robot's physical environment [3]. Alternatively, for a classification problem such as medical diagnosis, the environment is a training set of preclassified subjects (i.e., cases and controls) described by multiple attributes (e.g., genetic polymorphisms). By interacting with the environment, LCSs receive feedback in the form of numerical reward which drives the learning process. While many different implementations of LCS algorithms exist, Holmes et al. [4] outline four practically universal components: (1) a finite population of classifiers that represents the current knowledge of the system, (2) a performance component, which regulates interaction between the environment and classifier population, (3) a reinforcement component (also called credit assignment component [5]), which distributes the reward received from the environment to the classifiers, and (4) a discovery component which uses different operators to discover better rules and improve existing ones. Together, these components represent a basic framework upon which a number of novel alterations to the LCS algorithm have been built. Figure 2 illustrates how specific mechanisms of LCS (detailed in Section 9) interact in the context of these major components.

3. The Driving Mechanisms

While the above four components represent an algorithmic framework, two primary mechanisms are responsible for driving the system. These include discovery, generally by way of the genetic algorithm, and learning. Both mechanisms have generated respective fields of study, but it is in the context of LCS that we wish to understand their function and purpose.

3.1. Discovery—The Genetic Algorithm. Discovery refers to "rule discovery" or the introduction of rules that do not currently exist in the population. Ideally, new rules will be better at getting payoff (i.e., making good decisions) than existing ones. From the start, this task has almost always been achieved through the use of a genetic algorithm (GA). The GA is a computational search technique which manipulates (evolves) a population of individuals (rules) each representing a potential solution (or piece of a solution) to a given problem. The GA, as a major component of the first conceptualized LCS [6], has largely surpassed LCS in terms of celebrity and common usage. GAs [7, 8] are founded on ideas borrowed from nature. Inspired from the neo-Darwinist theory of natural selection, the evolution of rules is modeled after the evolution of organisms using four biological analogies: (1) a code is used to represent the genotype/genome (condition), (2) a solution (or phenotype) representation is associated with that genome (action), (3) a phenotype selection process (survival of the fittest), where the fittest organism (rule) has a greater chance of reproducing and passing its "genetic" information on to the next generation, and (4) genetic operators are utilized to allow simple transformations of the genome in search of fitter organisms (rules) [9, 10]. Variation in a genome (rule) is typically generated by two genetic operators: mutation and crossover (recombination). Crossover operators create new

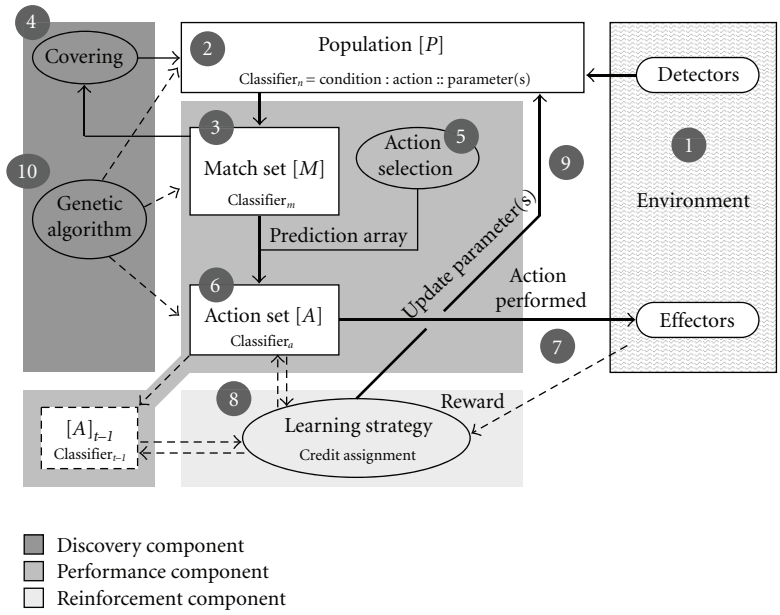

Figure 2: A Generic LCS—the values 1–10 indicate the typical steps included in a single learning iteration of the system. Thick lines indicate the flow of information, thin lines indicate a mechanism being activated, and dashed lines indicate either steps that do not occur every iteration, or mechanisms that might occur at different locals.

genotypes by recombining subparts of the genotypes of two or more individuals (rules). Mutation operators randomly modify an element in the genotype of an individual (rule). The selection pressure which drives "better" organisms (rules) to reproduce more often is dependent on the fitness function. The fitness function quantifies the optimality of a given rule, allowing that rule to be ranked against all other rules in the population. In a simple classification problem, one might use classification accuracy as a metric of fitness. Running a genetic algorithm requires looping through a series of steps for some number of iterations (generations). Initially, the user must predefine a number of parameters such as the population size (N) and the number of generations, based on the user's needs. Additionally the GA needs to be initialized with a population of rules which can be generated randomly to broadly cover the range of possible solutions (the search space). The following steps will guide the reader through a single iteration of a simple genetic algorithm.

(1) Evaluate the fitness of all rules in the current population.

(2) Select "parent" rules from the population (with probability proportional to fitness).

(3) Crossover and/or mutate "parent" rules to form "offspring" rules.

(4) Add "offspring" rules to the next generation.

(5) Remove enough rules from the next generation (with probability of being removed inversely proportional to fitness) to restore the number of rules to N.

As with LCSs, there are a variety of GA implementations which may vary the details underlying the steps described above (see Section 9.5). GA research constitutes its own field which goes beyond the scope of this paper. For a more detailed introduction to GAs we refer readers to Goldberg [8, 11].

3.2. Learning. In the context of artificial intelligence, learning can be defined as, "*the improvement of performance in some environment through the acquisition of knowledge resulting from experience in that environment*" [12]. This notion of learning via reinforcement (also referred to as *credit assignment* [3]) is an essential mechanism of the LCS architecture. Often the terms learning, reinforcement, and credit assignment are used interchangeably within the literature. In addition to a condition and action, each classifier in the LCS population has one or more parameter values associated with it (e.g., fitness). The iterative update of these parameter values drives the process of LCS reinforcement. More generally speaking, the update of parameters distributes any incoming reward (and/or punishment) to the classifiers that are accountable for it. This mechanism serves two purposes: (1) to identify classifiers that are useful in obtaining future rewards and (2) to encourage the discovery of better rules. Many of the existing LCS implementations utilize different learning strategies. One of the main reasons for this is that different problem domains demand different styles of learning. For example, learning can be categorized based on the manner in which information is received from this environment. *Offline* or "*batch*" learning implies that all training instances are presented simultaneously to the learner. The end result is a single rule set embodying a solution that does not change with respect to time. This type of learning is often characteristic of data mining problems. Alternatively, *online*

or "*incremental*" learning implies that training instances are presented to the learner one at a time, the end result of which is a rule set which changes continuously with the addition of each additional observation [12–14]. This type of learning may have no prespecified endpoint, as the system solution may continually modify itself with respect to a continuous stream of input. Consider, for example, a robot which receives a continuous stream of data about the environment it is attempting to navigate. Over time it may need to adapt its movements to maneuver around obstacles it has not yet faced. Learning can also be distinguished by the type of feedback that is made available to the learner. In this context, two learning styles have been employed by LCSs; *supervised* learning and *reinforcement* learning, of which the latter is often considered to be synonymous with LCS. Supervised learning implies that for each training instance, the learner is supplied not only with the condition information, but also with the "correct" action. The goal here is to infer a solution that generalizes to unseen instances based on training examples that possess correct input/output pairs. Reinforcement learning (RL), on the other hand, is closer to *unsupervised* learning, in that the "correct" action of a training instance is not known. However, RL problems do provide feedback, indicating the "goodness" of an action decision with respect to some goal. In this way, learning is achieved through trial-and-error interactions with the environment where occasional immediate reward is used to generate a policy that maximizes long term reward (delayed reward). The term 'policy' is used to describe a state-action map which models the agent-environment interactions. For a detailed introduction to RL we refer readers to Sutton and Barto (1998) [15], Harmon (1996) [16], and Wyatt (2005) [17]. Specific LCS learning schemes will be discussed further in Section 9.4.

4. A Minimal Classifier System

The working LCS algorithm is a relatively complex assembly of interacting mechanisms operating in an iterative fashion. We complete our functional introduction to LCSs with an algorithmic walk through. For simplicity, we will explore what might be considered one of the most basic LCS implementations, a minimal classifier system (MCS) [18]. This system is heavily influenced by modern LCS architecture. For an earlier perspective on simple LCS architecture see Goldberg's SCS [8]. MCS was developed by Larry Bull as a platform for advancing LCS theory. While it was not designed for real-world applications, MCS offers a convenient archetype upon which to better understand more complex implementations. Figure 3 outlines a single iteration of MCS. In this example the input data takes the form of a four-digit binary number, representing discrete observations detected from an instance in the environment. MCS learns iteratively, sampling one data instance at a time, learning from it, and then moving to the next. As usual, a population of classifiers represents the evolving solution to our given problem. Each of the (N) classifiers in the population are made up of a condition, an action, and an associated fitness parameter $\{F\}$. The condition is represented by a string of characters from the ternary alphabet 0, 1, # where # acts as a wildcard such that the rule condition 00#1 matches both the input 0011 and the input 0001. The action is represented by a binary string where in this case only two actions are possible (0 or 1). The fitness parameter gives an indication of how good a given classifier is, which is important not only for action selection, but for application of the GA to evolve better and better classifiers. Before the algorithm is run, the population of classifiers is randomly initialized, and the fitness parameters are each set to some initial value f_0. Figure 3 depicts the MCS after having already been run for a number of iterations made evident by the diversity of fitness scores. With the receipt of input data, the population is scanned and any rule whose condition matches the input string at each position becomes a member of the current "match set" $[M]$. If none of the rules in the population match the input, a covering operator generates a rule with a matching condition and a random action [19]. The number of wildcards incorporated into the new rule condition is dependent on the rate $(p_\#)$ set by the user. With the addition of a new rule, an existing rule must be removed from the population to keep (N) constant. This is done using roulette wheel selection where the probability of a rule being selected for replacement is inversely proportional to its fitness, that is, $1/(F_j+1)$ [20, 21]. Once the match set is established, an action is selected using a simple explore/exploit scheme [22]. This scheme alternates between randomly selecting an action found within $[M]$ one round (explore), and selecting deterministically with a prediction array the next (exploit). The prediction array is a list of prediction values calculated for each action found in $[M]$. In MCS, the prediction value is the sum of fitness values found in the subset of $[M]$ advocating the same action. The subset with the highest prediction value becomes the action set $[A]$, and the corresponding action is performed in the environment. Learning begins with receipt of an immediate reward (payoff = P) from the environment in response to the performed action. MCS uses a simple form of RL that uses the Widrow-Hoff procedure (see Section 9.4.2) with a user defined learning rate of β. The following equation updates the fitness of each rule in the current $[A]$:

$$F_j \longleftarrow F_j + \left(\left(\frac{P}{|[A]|}\right) - F_j\right). \tag{1}$$

The final step in MCS is the activation of a GA that operates within the entire population (panmitic). Together the GA and the covering operator make up the discovery mechanism of MCS. The GA operates as previously described where on each "explore" iteration, there is a probability (g) of GA invocation. This probability is only applied to "explore" iterations where action selection is performed randomly. Parent rules are selected from the population using roulette wheel selection. Offspring are produced using a mutation rate of (μ) (with a wildcard rate of $(p_\#)$) and a *single point* crossover rate of (χ). New rules having undergone mutation inherit their parent's fitness values, while those that have undergone crossover inherit the average fitness of the parents. New rules replace old ones as previously described. MCS is iterated in this manner over a user defined number of generations.

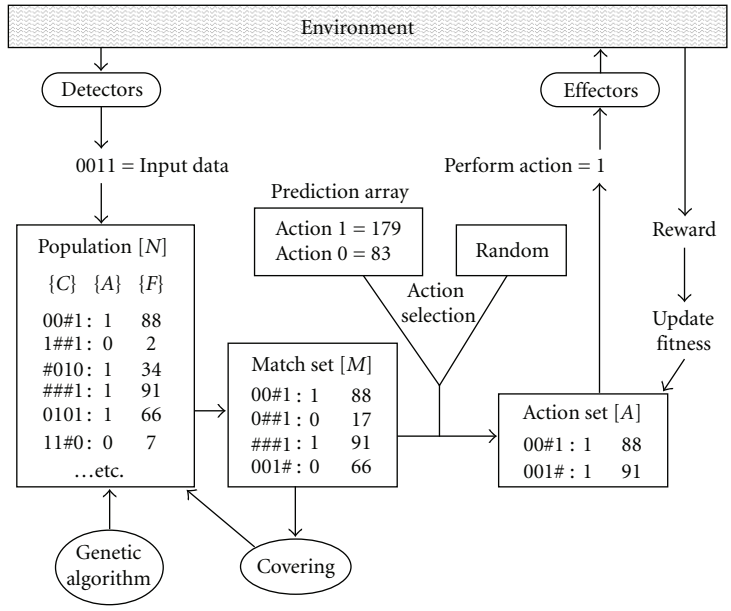

Figure 3: MCS algorithm—an example iteration.

5. Historical Perspective

The LCS concept, now three decades old, has inspired a wealth of research aimed at the development, comparison, and comprehension of different LCSs. The vast majority of this work is based on a handful of key papers [3, 19, 22–24] which can be credited with founding the major branches of LCS. These works have become the founding archetypes for an entire generation of LCS algorithms which seek to improve algorithmic performance when applied to different problem domains. As a result, many LCS algorithms are defined by an expansion, customization, or merger of one of the founding algorithms. Jumping into the literature, it is important to note that the naming convention used to refer to the LCS algorithm has undergone a number of changes since its infancy. John Holland, who formalized the original LCS concept [25] based around his more well-known invention, the Genetic Algorithm (GA) [6], referred to his proposal simply as a classifier system, abbreviated either as (CS), or (CFS) [26]. Since that time, LCSs have also been referred to as adaptive agents [1], cognitive systems [3], and genetics-based machine learning systems [2, 8]. On occasion they have quite generically been referred to as either production systems [6, 27] or genetic algorithms [28] which in fact describes only a part of the greater system. The now standard designation of a "learning classifier system" was not adopted until the late 80s [29] after Holland added a reinforcement component to the CS architecture [30, 31]. The rest of this section provides a synopsis of some of the most popular LCSs to have emerged, and the contributions they made to the field. This brief history is supplemented by Table 1 which chronologically identifies noted LCS algorithms/platforms and details some of the defining features of each. This table includes the LCS style (Michigan $\{M\}$, Pittsburgh $\{P\}$, Hybrid $\{H\}$, and Anticipatory $\{A\}$), the primary fitness basis, a summary of the learning style or credit assignment scheme, the manner in which rules are represented, the position in the algorithm at which the GA is invoked (panmitic $[P]$, match set $[M]$, action set $[A]$, correct set $[C]$, local neighborhood LN, and modified LN (MLN) and the problem domain(s)) on which the algorithm was designed and/or tested.

5.1. The Early Years. Holland's earliest CS implementation, called Cognitive System One (CS-1) [3] was essentially the first learning classifier system, being the first to merge a credit assignment scheme with a GA in order to evolve a population of rules as a solution to a problem who's environment only offered an infrequent payoff/reward. An immediate drawback to this and other early LCSs was the inherent complexity of the implementation and the lack of comprehension of the systems operation [8]. The CS-1 archetype, having been developed at the University of Michigan, would later inspire a whole generation of LCS implementations. These "Michigan-style" LCSs are characterized by a population of rules where the GA operates at the level of individual rules and the solution is represented by the entire rule population. Smith's 1980 dissertation from the University of Pittsburgh [23] introduced LS-1, an alternative implementation that founded the fundamentally different "Pittsburgh-style" LCS. Also referred to as the "Pitt-approach", the LS-1 archetype is characterized by a population of variable length rule-sets (each rule-set is a potential solution) where the GA typically operates at the level of an entire rule-set. An early advantage of the Pitt-approach came from its credit assignment scheme, where reward is assigned to entire rule-sets as opposed to individual rules. This allows Pitt-style systems such as LS-1 to circumvent the potential problem of having to

share credit amongst individual rules. But, in having to evolve multiple rule sets simultaneously, Pitt-style systems suffer from heavy computational requirements. Additionally, because Pitt systems learn iteratively from sets of problem instances, they can only work offline, whereas Michigan systems are designed to work online, but can engage offline problems as well. Of the two styles, the Michigan approach has drawn the most attention as it can be applied to a broader range of problem domains and larger, more complex tasks. As such, it has largely become what many consider to be the standard LCS framework. All subsequent systems mentioned in this review are of Michigan-style unless explicitly stated otherwise. Following CS-1, Holland's subsequent theoretical and experimental investigations [30–40] advocated the use of the *bucket brigade* credit assignment scheme (see Section 9.4.1). The bucket brigade algorithm (BBA), inspired by Samuel [41] and formalized by Holland [38] represents the first learning/credit assignment scheme to be widely adopted by the LCS community [20, 42, 43]. Early work by Booker on a CS-1 based system suggested a number of modifications including the idea to replace the panmictically acting GA with a niche-based one (i.e., the GA acts on $[M]$ instead of $[P]$) [42]. The reason for this modification was to eliminate undesirable competition between unrelated classifiers, and to encourage more useful crossovers between classifiers of a common "environmental niche". This in turn would help the classifier population retain diversity and encourage inclusion of problem subdomains in the solution. However, it should be noted that niching has the likely impact of making the GA more susceptible to local maxima, a disadvantage for problems with a solution best expressed as a single rule. In 1985, Stewart Wilson implemented an Animat CS [44] that utilized a simplified version of the bucket brigade, referred to later as an *implicit bucket brigade* [8]. Additionally, the Animat system introduced a number of concepts which persist in many LCSs today including *covering* (via a "create" operator), the formalization of an action set $[A]$, an estimated time-to-payoff parameter incorporated into the learning scheme, and a general progression towards a simpler CS architecture [44, 45]. In 1986 Holland published what would become known as the standard CS for years to come [30]. This implementation incorporated a strength-based fitness parameter and BBA credit assignment as described in [38]. While still considered to be quite complex and susceptible to a number of problems [45], the design of this hallmark LCS is to this day a benchmark for all other implementations. The next year, Wilson introduced BOOLE, a CS developed specifically to address the problem of learning Boolean functions [43]. Characteristic of the Boolean problem, classifiers are immediately rewarded in response to performing actions. As a result BOOLE omits sequential aspects of the CS, such as the BBA which allows reward to be delayed over a number of time steps, instead relying on a simpler "one-step" CS. Bonelli et. al. [46] later extended BOOLE to a system called NEWBOOLE in order to improve the learning rate. NEWBOOLE introduced a "symmetrical payoff-penalty" (SPP) algorithm (reminiscent of supervised learning) which replaced $[A]$ with a correct set $[C]$, and not-correct set $Not\ [C]$. In 1989, Booker continued his work with GOFER-1 [47] adding a fitness function based on both payoff and nonpayoff information (e.g., strength and specificity), and further pursued the idea of a "niching" GA. Another novel system which spawned its own lineage of research is Valenzuela's fuzzy LCS which combined fuzzy logic with the concept of a rule-based LCSs [48]. The fuzzy LCS represents one of the first systems to explore a rule representation beyond the simple ternary system. For an introduction to fuzzy LCS we refer readers to [49]. An early goal for LCSs was the capacity to learn and represent more complex problems using "internal models" as was originally envisioned by Holland [34, 38]. Work by Rick Riolo addressed the issues of forming "long action chains" and "default hierarchies" which had been identified as problematic for the BBA [29, 50, 51]. A long action chain refers to a series of rules which must sequentially activate before ultimately receiving some environmental payoff. They are challenging to evolve since "*there are long delays before rewards are received, with many unrewarded steps between some stage setting actions and the ultimate action those actions lead to*" [52]. Long chains are important for modeling behavior which require the execution of many actions before the receipt of a reward. A default hierarchy is a set of rules with increasing levels of specificity, where the action specified by more general rules is selected by "default" except in the case where overriding information is able to activate a more specific rule. "*Holland has long argued that default hierarchies are an efficient, flexible, easy-to-discover way to categorize observations and structure models of the world*" [52]. Reference [53] describes hierarchy formation in greater detail. Over the years a number of methods have been introduced in order to allow the structuring of internal models. Examples would include internal message lists, non-message-list memory mechanisms, "corporate" classifier systems, and enhanced rule syntax and semantics [52]. Internal message lists, part of the original CS-1 [3] exist as a means to handle all input and output communication between the system and the environment, as well as providing a makeshift memory for the system. While the message list component can facilitate complex internal structures, its presence accounts for much of the complexity in early LCS systems. The tradeoff between complexity and comprehensibility is a theme which has been revisited throughout the course of LCS research [45, 52, 54]. Another founding system is Riolo's CFCS2 [55], which addressed the particularly difficult task of performing "latent learning" or "look-ahead planning" where "*actions are based on predictions of future states of the world, using both current information and past experience as embodied in the agent's internal models of the world*" [52]. This work would later inspire its own branch of LCS research: anticipatory classifier systems (ACS) [24]. CFCS2 used "tags" to represent internal models, claiming a reduction in the learning time for general sequential decision tasks. Additionally, this system is one of the earliest to incorporate a Q-learning-like credit assignment technique (i.e., a nonbucket brigade temporal difference method). Q-learning-based credit assignment would later become a central component of the most popular LCS implementation to date.

5.2. The Revolution. From the late 80s until the mid-90s the interest generated by these early ideas began to diminish as researchers struggled with LCS's inherent complexity and the failure of various systems to reliably obtain the behavior and performance envisioned by Holland. Two events have repeatedly been credited with the revitalization of the LCS community, namely the publication of the "Q-Learning" algorithm in the RL community, and the advent of a significantly simplified LCS architecture as found in the ZCS and XCS (see Table 1). The fields of RL and LCSs have evolved in parallel, each contributing to the other. RL has been an integral component of LCSs from the very beginning [3]. While the founding concepts of RL can be traced back to Samuel's checker player [41], it was not until the 80s that RL became its own identifiable area of machine learning research [159]. Early RL techniques included Holland's BBA [38] and Sutton's *temporal difference* (TD) method [160] which was followed closely by Watkins's Q-Learning method [161]. Over the years a handful of studies have confirmed the basic equivalence of these three methods, highlighting the distinct ties between the two fields. To summarize, the BBA was shown to be one kind of TD method [160], and the similarity between all three methods were noted by Watkins [161] and confirmed by Liepins, Dorigo, and Bersini [162, 163]. This similarity across fields paved the way for the incorporation of Q-learning-based techniques into LCSs. To date, Q-learning is the most well-understood and widely-used RL algorithm available. In 1994, Wilson's pursuit of simplification culminated in the development of the "zeroth-level" classifier system (ZCS) [19], aimed at increasing the understandability and performance of an LCS. ZCS differed from the standard LCS framework in that it removed the rule-bidding and internal message list, both characteristic of the original BBA (see Section 9.3). Furthermore, ZCS was able to disregard a number of algorithmic components which had been appended to preceding systems in an effort to achieve acceptable performance using the original LCS framework (e.g., heuristics [44] and operators [164]). New to ZCS, was a novel credit assignment strategy that merged elements from the BBA and Q-Learning into the "QBB" strategy. This hybrid strategy represents the first attempt to bridge the gap between the major LCS credit assignment algorithm (i.e., the BBA) and other algorithms from the field of RL. With ZCS, Wilson was able to achieve similar performance to earlier, more complex implementations demonstrating that Holland's ideas could work even in a very simple framework. However, ZCS still exhibited unsatisfactory performance, attributed to the proliferation of over-general classifiers. The following year, Wilson introduced an eXtended Classifier System (XCS) [22] noted for being able to reach optimal performance while evolving accurate and maximally general classifiers. Retaining much of the ZCS architecture, XCS can be distinguished by the following key features: an accuracy based fitness, a niche GA (acting in the action set [A]), and an adaptation of standard Q-Learning as credit assignment. Probably the most important innovation in XCS was the separation of the credit assignment component from the GA component, based on accuracy. Previous LCSs typically relied on a *strength* value allocated to each rule (reflecting the reward the system can expect if that rule is fired; a.k.a. reward prediction). This one strength value was used both as a measure of fitness for GA selection, and to control which rules are allowed to participate in the decision making (i.e., predictions) of the system. As a result, the GA tends to eliminate classifiers from the population that have accumulated less reward than others, which can in turn remove a low-predicting classifier that is still well suited for its environmental niche. "*Wilson's intuition was that prediction should estimate how much reward might result from a certain action, but that the evolution learning should be focused on most reliable classifiers, that is, classifiers that give a more precise (accurate) prediction*" [165]. With XCS, the GA fitness is solely dependent on rule accuracy calculated separately from the other parameter values used for decision making. Although not a new idea [3, 25, 166], the accuracy-based fitness of XCS represents the starting point for a new family of LCSs, termed "accuracy-based" which are distinctly separable from the family of "strength-based" LCSs epitomized by ZCS (see Table 1). XCS is also important, because it successfully bridges the gap between LCS and RL. RL typically seeks to learn a value function which maps out a complete representation of the state/action space. Similarly, the design of XCS drives it to form an all-inclusive and accurate representation of the problem space (i.e., a *complete map*) rather than simply focusing on higher payoff niches in the environment (as is typically the case with strength-based LCSs). This latter methodology which seeks a rule set of efficient generalizations tends to form a *best action map* (or a *partial map*) [102, 167]. In the wake of XCS, it became clear that RL and LCS are not only linked but inherently overlapping. So much so, that analyses by Lanzi [168] led him to define LCSs as RL systems endowed with a generalization capability. "*This generalization property has been recognized as the distinguishing feature of LCSs with respect to the classical RL framework*" [9]. "*XCS was the first classifier system to be both general enough to allow applications to several domains and simple enough to allow duplication of the presented results*" [54]. As a result XCS has become the most popular LCS implementation to date, generating its own following of systems based directly on or heavily inspired by its architecture.

5.3. In the Wake of XCS. Of this following, three of the most prominent will be discussed: ACS, XCSF, and UCS. In 1998 Stolzmann introduced ACS [24] and in doing so formalized a new LCS family referred to as "anticipation-based". "[ACS] *is able to predict the perceptual consequences of an action in all possible situations in an environment. Thus the system evolves a model that specifies not only what to do in a given situation but also provides information of what will happen after a specific action was executed*" [103]. The most apparent algorithmic difference in ACS is the representation of rules in the form of a condition-action-effect as opposed to the classic condition-action. This architecture can be used for multi-step problems, planning, speeding up learning, or disambiguating perceptual aliasing (where the same observation is obtained in distinct states

Table 1: A summary of noted LCS algorithms

System	Year	Author/cite	Style	Fitness	Learning/credit assignment	Rule rep.	GA	Problem
CS-1	1978	Holland [3]	M	Accuracy	Epochal	Ternary	[P]	Maze Navigation
LS-1	1980	Smith [23]	P	Accuracy	Implicit Critic	Ternary	[P]	Poker Decisions
CS-1 (based)	1982	Booker [42]	M	Strength	Bucket Brigade	Ternary	[M]	Environment Navigation
Animat CS	1985	Wilson [44]	M	Strength	Implicit Bucket Brigade	Ternary	[P]	Animat Navigation
LS-2	1985	Schaffer [56]	P	Accuracy	Implicit Critic	Ternary	[P]	Classification
Standard CS	1986	Holland [30]	M	Strength	Bucket Brigade	Ternary	[P]	Online Learning
BOOLE	1987	Wilson [43]	M	Strength	One-Step Payoff-Penalty	Ternary	[P]	Boolean Function Learning
ADAM	1987	Greene [57]	P	Accuracy	Custom	Ternary	[P]	Classification
RUDI	1988	Grefenstette [58]	H	Strength	Bucket-Brigade and Profit-Sharing Plan	Ternary	[P]	Generic Problem Solving
GOFER	1988	Booker [59]	M	Strength	Payoff-Sharing	Ternary	[M]	Environment Navigation
GOFER-1	1989	Booker [47]	M	Strength	Bucket-Brigade-like	Ternary	[M]	Multiplexer Function
SCS	1989	Goldberg [8]	M	Strength	AOC	Trit	[P]	Multiplexer Function
SAMUEL	1989–1997	Grefenstette [60–62]	H	Strength	Profit-Sharing Plan	Varied	[P]	Sequential Decision Tasks
NEWBOOLE	1990	Bonelli [46]	M	Strength	Symmetrical Payoff-Penalty	Ternary	[P]	Classification
CFCS2	1991	Riolo [55]	M	Strength/ Accuracy	Q-Learning-Like	Ternary	[P]	Maze Navigation
HCS	1991	Shu [63]	H	Strength	Custom	Ternary	[P]	Boolearn Function Learning
Fuzzy LCS	1991	Valenzuela-Rendon [48]	M	Strength	Custom Bucket-Brigade	Binary - Fuzzy Logic	[P]	Classification
ALECSYS	1991–1995	Dorigo [64, 65]	M	Strength	Bucket Brigade	Ternary	[P]	Robotics
GABIL	1991-1993	De Jong [66, 67]	P	Accuracy	Batch - Incremental	Binary - CNF	[P]	Classification
GIL	1991–1993	Janikow [68, 69]	P	Accuracy	Supervised Learning - Custom	Multi-valued logic (VL1)	[P]	Multiple Domains
GARGLE	1992	Greene [70]	P	Accuracy	Custom	Ternary	[P]	Classification
COGIN	1993	Greene [71]	M	Accuracy/ Entropy	Custom	Ternary	[P]	Classification, Model Induction
REGAL	1993	Giordana [72–74]	H	Accuracy	Custom	Binary - First Order Logic	[P]	Classification
ELF	1993–1996	Bonarini [75–77]	H	Strength	Q-Learning-Like	Binary - Fuzzy Logic	[P]	Robotics, Cart-Pole Problem
ZCS	1994	Wilson [19]	M	Strength	Implicit Bucket Brigade	Ternary	[P]	Environment Navigation
ZCSM	1994	Cliff [78]	M	Strength	Implicit Bucket Brigade - Memory	Ternary	[P]	Environment Navigation
XCS	1995	Wilson [22]	M	Accuracy	Q-Learning-Like	Ternary	[A]	Mulitplexor Function and Environment Navigation

Table 1: Continued.

System	Year	Author/cite	Style	Fitness	Learning/credit assignment	Rule rep.	GA	Problem
GA-Miner	1995–1996	Flockhart [79, 80]	H	Accuracy	Custom	Symbolic Functions	LN	Classification, Data Mining
BOOLE++	1996	Holmes [81]	M	Strength	Symmetrical Payoff-Penalty	Ternary	[P]	Epidemiologic Classification
EpiCS	1997	Holmes [82]	M	Strength	Symmetrical Payoff-Penalty	Ternary	[P]	Epidemiologic Classification
XCSM	1998	Lanzi [83, 84]	M	Accuracy	Q-Learning-Like	Ternary	[A]	Environment Navigation
ZCCS	1998–1999	Tomlinson [85, 86]	H	Strength	Implicit Bucket Brigade	Ternary	[P]	Environment Navigation
ACS	1998–2000	Stolzmann [24, 87]	A	Strength/ Accuracy	Bucket-Brigade-like (reward or anticipation learning)	Ternary	—	Environment Navigation
iLCS	1999-2000	Browne [88, 89]	M	Strength/ Accuracy	Custom	Real-Value Alphabet	[P]	Industrial Applications - Hot Strip Mill
XCSMH	2000	Lanzi [90]	M	Accuracy	Q-Learning-Like	Ternary	[A]	Non-Markov Environment Navigation
CXCS	2000	Tomlinson [91]	H	Accuracy	Q-Learning-Like	Ternary	[A]	Environment Navigation
XCSR	2000	Wilson [92]	M	Accuracy	Q-Learning-Like	Interval Predicates	[A]	Real-Valued Multiplexor Problems
ClaDia	2000	Walter [93]	M	Strength	Supervised Learning - Custom	Binary - Fuzzy Logic	[P]	Epidemiologic Classification
OCS	2000	Takadama [94]	O	Strength	Profit Sharing	Binary	[P]	Non-Markov Multiagent Environments
XCSI	2000-2001	Wilson [95, 96]	M	Accuracy	Q-Learning-Like	Interval Predicates	[A]	Integer-Valued Data Mining
MOLeCS	2000-2001	Bernado-Mansilla [97, 98]	M	Accuracy	Multi- objective Learning	Binary	[P]	Multiplexor Problem
YACS	2000–2002	Gerard [99, 100]	A	Accuracy	Latent Learning	Tokens	—	Non-Markov Environment Navigation
SBXCS	2001-2002	Kovacs [101, 102]	M	Strength	Q-Learning-Like	Ternary	[A]	Multiplexor Function
ACS2	2001-2002	Butz [103, 104]	A	Accuracy	Q-Learning-Like	Ternary	—	Environment Navigation
ATNoSFERES	2001–2007	Landau and Picault [105–109]	P	Accuracy	Custom	Graph-Based Binary-Tokens	[P]	Non-Markov Environment Navigation
GALE	2001-2002	Llora [110, 111]	P	Accuracy	Custom	Binary	LN	Classification, Data Mining
GALE2	2002	Llora [112]	P	Accuracy	Custom	Binary	MLN	Classification, Data Mining
XCSF	2002	Wilson [113]	M	Accuracy	Q-Learning-Like	Interval Predicates	[A]	Function Approximation

Table 1: Continued.

System	Year	Author/cite	Style	Fitness	Learning/credit assignment	Rule rep.	GA	Problem
AXCS	2002	Tharakunnel [114]	M	Accuracy	Q-Learning-Like	Ternary	[A]	Multi-step Problems Environmental Navigation
TCS	2002	Hurst [115]	M	Strength	Q-Learning-Like	Interval Predicates	[P]	Robotics
X-NCS	2002	Bull [116]	M	Accuracy	Q-Learning-Like	Neural Network	[A]	Multiple Domains
X-NFCS	2002	Bull [116]	M	Accuracy	Q-Learning-Like	Fuzzy - Neural Network	[A]	Function Approximation
UCS	2003	Bernado-Mansilla [117]	M	Accuracy	Supervised Learning - Custom	Ternary	[C]	Classification - Data Mining
XACS	2003	Butz [118]	A	Accuracy	Generalizing State Value Learner	Ternary	—	Blocks World Problem
XCSTS	2003	Butz [119]	M	Accuracy	Q-Learning-Like	Ternary	[A]	Multiplexor Problem
MOLCS	2003	Llora [120]	P	Multi-objective	Custom	Ternary	[P]	Classification - LED Problem
YCS	2003	Bull [121]	M	Accuracy	Q-Learning-Like Widrow-Hoff	Ternary	[P]	Accuracy Theory - Multiplexor Problem
XCSQ	2003	Dixon [122]	M	Accuracy	Q-Learning-Like	Ternary	[A]	Rule-set Reduction
YCSL	2004	Bull [123]	A	Accuracy	Latent Learning	Ternary	—	Environment Navigation
PICS	2004	Gaspar [124, 125]	P	Accuracy	Custom - Artificial Immune System	Ternary	[P]	Multiplexor Problem
NCS	2004	Hurst [126]	M	Strength	Q-Learning-Like	Neural Network	[P]	Robotics
MCS	2004	Bull [127]	M	Strength	Q-Learning-Like Widrow-Hoff	Ternary	[P]	Strength Theory - Multiplexor Problem
GAssist	2004–2007	Bacardit [128–131]	P	Accuracy	ILAS	ADI - Binary	[P]	Data Mining UCI Problems
MACS	2005	Gerard [132]	A	Accuracy	Latent Learning	Tokens	—	Non-Markov Environment Navigation
XCSFG	2005	Hamzeh [133]	M	Accuracy	Q-Learning-Like	Interval Predicates	[A]	Function Approximation
ATNoSFERES-II	2005	Landau [134]	P	Accuracy	Custom	Graph-Based Integer-Tokens	[P]	Non-Markov Environment Navigation
GCS	2005	Unold [135, 136]	M	Accuracy	Custom	Context-Free Grammar CNF	[P]	Learning Context-Free Languages
DXCS	2005	Dam [137–139]	M	Accuracy	Q-Learning-Like	Ternary	[A]	Distributed Data Mining
LCSE	2005–2007	Gao [140–142]	M	Strength and Accuracy	Ensemble Learning	Interval Predicates	[A]	Data Mining UCI Problems

Table 1: Continued.

System	Year	Author/cite	Style	Fitness	Learning/credit assignment	Rule rep.	GA	Problem
EpiXCS	2005–2007	Holmes [143–145]	M	Accuracy	Q-Learning-Like	Ternary	[A]	Epidemiologic Data Mining
XCSFNN	2006	Loiacono [146]	M	Accuracy	Q-Learning-Like	Feedforward Multilayer Neural Network	[A]	Function Approximation
BCS	2006	Dam [147]	M	Bayesian	Supervised Learning - Custom	Ternary	[C]	Multiplexor Problem
BioHEL	2006	Bacardit [148, 149]	P	Accuracy	Custom	ADI - Binary	[P]	Larger Problems -Multiplexor, Protein Structure Prediction
XCSFGH	2006	Hamzeh [150]	M	Accuracy	Q-Learning-Like	Binary Polynomials	[A]	Function Approximation
XCSFGC	2007	Hamzeh [151]	M	Accuracy	Q-Learning-Like	Interval Predicates	[A]	Function Approximation
XCSCA	2007	Lanzi [152]	M	Accuracy	Supervised Learning - Custom	Interval Predicates	[M]	Environmental Navigation
LCSE	2007	Gao [142]	M	Accuracy	Q-Learning-Like	Interval Predicates	[A]	Medical Data Mining - Ensemble Learning
CB-HXCS	2007	Gershoff [153]	M	Accuracy	Q-Learning-Like	Ternary	[A]	Multiplexor Problem
MILCS	2007	Smith [154]	M	Accuracy	Supervised Learning - Custom	Neural Network	[C]	Multiplexor, Protein Structure
rGCS	2007	Cielecki [155]	M	Accuracy	Custom	Real-Valued Context-Free Grammar Based	[P]	Checkerboard Problem
Fuzzy XCS	2007	Casilas [156]	M	Accuracy	Q-Learning-Like	Binary - Fuzzy Logic	[A]	Single Step Reinforcement Problems
Fuzzy UCS	2007	Orriols-Puig [157]	M	Accuracy	Supervised Learning - Custom	Binary - Fuzzy Logic	[C]	Data Mining UCI Problems
NAX	2007	Llora [158]	P	Accuracy	Custom	Interval Predicates	[P]	Classification - Large Data Sets
NLCS	2008	Dam [2]	M	Accuracy	Supervised Learning - Custom	Neural Network	[C]	Classification

requiring different actions). Contributing heavily to this branch of research, Martin Butz later introduced ACS2 [103] and developed several improvements to the original model [104, 118, 169–171] . For a more in depth introduction to ACS we refer the reader to [87, 172]. Another brainchild of Wilson's was XCSF [113]. The complete action mapping of XCS made it possible to address the problem of function approximation. "*XCSF evolves classifiers which represent piecewise linear approximations of parts of the reward surface associated with the problem solution*" [54]. To accomplish this, XCSF introduces the concept of computed prediction, where the classifier's prediction (i.e., predicted reward) is no longer represented by a scalar parameter value, but is instead a function calculated as a linear combination of the classifier's inputs (for each dimension) and a weight vector maintained by each classifier. In addition to systems based on fuzzy logic, XCSF is of the minority of systems able to support continuous-valued actions. In complete contrast to the spirit of ACS, the sUpervised Classifier System (UCS) [117] was designed specifically to address single-step problem domains such as classification and data mining where delayed reward is not a concern. While XCS and the vast majority of other LCS implementations rely on RL, UCS trades this strategy for supervised learning. Explicitly, classifier prediction was replaced by accuracy in order to reflect the nature of a problem domain where the system is trained, knowing the correct prediction in advance. UCS demonstrates that a best action map can yield effective generalization, evolve

more compact knowledge representations, and can converge earlier in large search spaces.

5.4. Revisiting the Pitt. While there is certainly no consensus as to which style LCS (Michigan or Pittsburgh) is "better", the advantages of each system in the context of specific problem domains are becoming clearer [173]. Some of the more successful Pitt-style systems include GABIL [66], GALE [110], ATNoSFERES [106], MOLCS [120], GAssist [128], BioHEL [148] (a descendant of GAssist), and NAX [158] (a descendant of GALE). All but ATNoSFERES were designed primarily to address classification/data mining problems for which Pitt-style systems seem to be fundamentally suited. NAX and BioHEL both received recent praise for their human-competitive performance on moderately complex and large tasks. Also, a handful of "hybrid" systems have been developed, which merge Michigan and Pitt-style architectures (e.g., REGAL [72], GA-Miner [79], ZCCS [85], and CXCS [91]).

5.5. Visualization. There is an expanding wealth of literature beyond what we have discussed in this brief history [174]. One final innovation, which will likely prove to be of great significance to the LCS community is the design and application of visualization tools. Such tools allow researchers to follow algorithmic progress by (1) tracking online performance (i.e., by graphing metrics such as error, generality, and population size), (2) visualizing the current classifier population as it evolves (i.e., condition visualization), and (3) visualizing the action/prediction (useful in function approximation to visualize the current prediction surface) [144, 154, 175]. Examples include Holmes's EpiXCS Workbench geared towards knowledge discovery in medical data [144], and Butz and Stalph's cutting-edge XCSF visualization software geared towards function approximation [175, 176] and applied to robotic control in [177]. Tools such as these will advance algorithmic understandability and facilitate solution interpretation, while simultaneously fueling a continued interest in the LCS algorithm.

6. Problem Domains

The range of problem domains to which LCS has been applied can be broadly divided into three categories: function approximation problems, classification problems, and reinforcement learning problems [178]. All three domains are generally tied by the theme of optimizing prediction within an environment. *Function approximation problems* seek to accurately approximate a function represented by a partially overlapping set of approximation rules (e.g., a piecewise linear solution for a sine function). *Classification problems* seek to find a compact set of rules that classify all problem instances with maximal accuracy. Such problems frequently rely on supervised learning where feedback is provided instantly. A broad subdomain of the classification problem includes "data mining" which is the process of sorting through large amounts of data to extract or model useful patterns. Classification problems may also be divided into either Boolean or real-valued problems based on the problem type being respectively discrete, or continuous in nature. Examples of classification problems include Boolean function learning, medical diagnosis, image classification (e.g., letter recognition), pattern recognition, and game analysis. *RL problems* seek to find an optimal behavioral policy represented by a compact set of rules. These problems are typically distinguished by inconsistent environmental reward often requiring multiple actions before such reward is obtained (i.e., multi-step RL problem or sequential decision task). Examples of such problems would include robotic control, game strategy, environmental navigation, modeling time-dependant complex systems (e.g., stock market), and design optimization (e.g., engineering applications). Some RL problems are characterized by providing immediate reward feedback about the accuracy of a chosen class (i.e., single-step RL problem), which essentially makes it similar to a classification problem. RL problems can be partitioned further based on whether they can be modeled as a Markov decision process (MDP) or a partially observable Markov decision process (POMDP) . In short, for Markov problems the selection of the optimal action at any given time depends only on the current state of the environment and not on any past states. On the other hand, Non-Markov problems may require information on past states to select the optimal action. For a detailed introduction to this concept we refer readers to [9, 179, 180].

7. Biological Applications

One particularly demanding and promising domain for LCS application involves biological problems (e.g., epidemiology, medical diagnosis, and genetics). In order to gain insight into complex biological problems researchers often turn to algorithms which are themselves inspired by biology (e.g., genetic programming [181], ant colony optimization [182], artificial immune systems [183], and neural networks [184]). Similarly, since the mid 90s biological LCS studies have begun to appear that deal mainly with classification-type problems. One of the earliest attempts to apply an LCS algorithm to such a problem was [28]. Soon after, John Holmes initiated a lineage of LCS designed for epidemiological surveillance and knowledge discovery which included BOOLE++ [81], EpiCS [82], and most recently EpiXCS [143]. Similar applications include [93, 95, 130, 142, 185–187], all of which examined the Wisconsin breast cancer data taken from the UCI repository [188]. LCSs have also been applied to protein structure prediction [131, 149, 154], diagnostic image classification [158, 189], and promoter region identification [190].

8. Optimizing LCS

There are a number of factors to consider when trying to select or develop an "effective" LCS. The ultimate value of an LCS might be gauged by the following: (1) *performance*—the quality of the evolved solution (rule set), (2) *scalability*—how rapidly the learning time or system size grows as the problem

complexity increases, (3) *adaptivity*—the ability of online learning systems to adapt to rapidly changing situations, and/or (4) *speed*—the time it takes an offline learning system to reach a "good" solution. Much of the field's focus has been placed on optimizing performance (as defined here). The challenge of this task is in balancing algorithmic pressures designed to evolve the population of rules towards becoming what might be considered an optimal rule set. The definition of an optimal rule set is subjective, depending on the problem domain, and the system architecture. Kovacs discusses the properties of an optimal XCS rule set [O] as being correct, complete, minimal (compact), and non-overlapping [191]. Even for the XCS architecture it is not clear that these properties are always optimal (e.g., discouraging overlap prevents the evolution of default hierarchies, too much emphasis on correctness may lead to overfitting in training, and completeness is only important if the goal is to evolve a complete action map). Some of the tradeoffs are discussed in [192, 193]. Instead, researchers may use the characteristics of *correctness, completeness, compactness*, and *overlap* as metrics with which to track evolutionary learning progress. LCS, being a complex multifaceted algorithm is subject to a number of different pressures driving the rule-set evolution. Butz and Pelikan discuss 5 pressures that specifically influence XCS performance, and provide an intuitive visualization of how these pressures interact to evolve the intended complete, accurate, and maximally general problem representation [194, 195]. These include *set pressure* (an intrinsic generalization pressure), *mutation pressure* (which influences rule specificity), *deletion pressure* (included in set pressure), *subsumption pressure* (decreases population size), and *fitness pressures* (which generate a major drive towards accuracy). Other pressures have also been considered, including parsimony pressure for discouraging large rule sets (i.e., bloat) [196], and crowding (or niching) pressure for allocating classifiers to distinct subsets of the problem domain [42]. In order to ensure XCS success, Butz defines a number of learning bounds which address specific algorithmic pitfalls [197–200]. Broadly speaking, the number of studies addressing LCS theory are few in comparison to applications-based research. Further work in this area would certainly benefit the LCS community.

9. Component Roadmap

The following section is meant as a summary of the different LCS algorithmic components. Figure 2 encapsulates the primary elements of a generic LCS framework (heavily influenced by ZCS, XCS, and other Michigan-style systems). Using this generalized framework we identify a number of exchangeable methodologies, and direct readers towards the studies that incorporate them. Many of these elements have been introduced in Section 5, but are put in the context of the working algorithm here. It should be kept in mind that some outlying LCS implementations stray significantly from this generalized framework, and while we present these components separately, the system as a whole is dependent on the interactions and overlaps which connect them. Elements that do not obviously fit into the framework of Figure 2 will be discussed in Section 9.6. Readers interested in a simple summary and schematic of the three most renowned systems (including Holland's standard LCS, ZCS, and XCS) are referred to [201].

9.1. Detectors and Effectors. The first and ultimately last step of an LCS iteration involves interaction with the environment. This interaction is managed by detectors and effectors [30]. Detectors sense the current state of the environment and encode it as a standard message (i.e., formatted input data). The impact of how sensors are encoded has been explored [202]. Effectors, on the other hand, translate action messages into performed actions that modify the state of the environment. For supervised learning problems, the action is supplanted by some prediction of class, and the job of effectors is simply to check that the correct prediction was made. Depending on the efficacy of the systems' predicted action or class, the environment may eventually or immediately reward the system. As mentioned previously, the environment is the source of input data for the LCS algorithm, dependant on the problem domain being examined. "*The learning capabilities of LCS rely on and are constrained by the way the agent perceives the environment, e.g., by the detectors the system employs*" [52]. Also, the format of the input data may be binary, real-valued, or some other customized representation. In systems dealing with batch learning, the dataset that makes up the environment is often divided into a training and a testing set (e.g., [82]) as part of a cross-validation strategy to assess performance and ensure against overfitting.

9.2. Population. Modifying the knowledge representation of the population can occur on a few levels. First and foremost is the difference in overall population structure as embodied by the Michigan and Pitt-style families. In Michigan systems the population is made up of a single rule-set which represents the problem solution, and in Pitt systems the population is a collection of multiple competing rule-sets, each which represent a potential problem solution (see Figure 4). Next, is the overall structure of an individual rule. Most commonly, a rule is made up of a condition, an action, and one or more parameter values (typically including a prediction value and/or a fitness value) [1, 19, 22], but other structures have been explored e.g., the condition-action-effect structure used by ACSs [24]. Also worth mentioning are rule-structure-induced mechanisms, proposed to encourage the evolution of rule dependencies and internal models. Examples include: bridging-classifiers (to aid the learning of long action chains) [38, 50], tagging (a form of implicitly linking classifiers) [1, 203, 204], and classifier-chaining (a form of explicitly linking classifiers and the defining feature of a "corporate" classifier system) [85, 91]. The most basic level of rule representation is the *syntax* which depicts how either the condition or action is actually depicted. Many different syntaxes have been examined for representing a rule condition. The first, and probably most commonly used syntax for condition

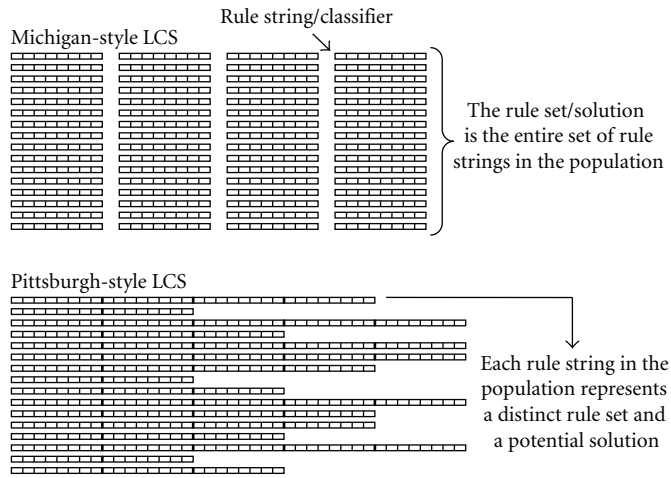

Figure 4: Michigan versus Pitt-style systems.

representation was fixed length bit-strings of the ternary alphabet $(0, 1, \#)$ corresponding with the simple binary encoding of input data [1, 3, 19, 22]. Unfortunately, it has been shown that this type of encoding can introduce bias as well as limit the system's ability to represent a problem solution [205]. For problems involving *real-valued* inputs the following condition syntaxes have been explored: real-valued alphabet [88], center-based interval predicates [92], min-max interval predicates [95], unordered-bound interval predicates [206], min-percentage representation [207], convex hulls [208], real-valued context-free grammar [155], ellipsoids [209], and hyper-ellipsoids [210]. Other condition syntaxes include: partial matching [211], value representation [203], tokens, [99, 105, 132, 134], context-free grammar [135], first-order logic expressions [212], messy conditions [213], GP-like conditions (including s-expressions) [79, 214–218], neural networks [2, 116, 219, 220], and fuzzy logic [48, 75, 77, 156, 157, 221]. Overall, advanced representations tend to improve generalization and learning, but require larger populations to do so. Action representation has seen much less attention. Actions are typically encoded in binary or by a set of symbols. Recent work has also begun to explore the prospect of computed actions, also known as computed prediction, which replaces the usual classifier action parameter with a function (e.g., XCSF function approximation) [113, 152]. Neural network predictors have also been explored [146]. Backtracking briefly, in contrast to Michigan-style systems, Pitt-style implementations tend to explore different rule *semantics* and typically rely on a simple binary syntax. Examples of this include: VL1 [68], CNF [66], and ADI [128, 148]. Beyond structural representation, other issues concerning the population include: (1) population initialization, (2) deciding whether to bound the population size (N), and if it is bound, (3) what value of (N) to select [129, 200].

9.3. Performance Component and Selection. This section will discuss different performance component structures and the selection mechanisms involved in covering, action selection, and the GA. The *message list*, a component found in many early LCSs (not included on Figure 2), is a kind of blackboard that documents the current state of the system. Acting as an interface, the message list temporarily stores all communications between the system and the environment (i.e., inputs from the detector, and classifier-posted messages that culminate as outputs to the effector) [30, 37, 59, 201]. One potential benefit of using the message list is that the LCS *"can emulate memory mechanisms when a message is kept on the list over several time steps"* [9]. The role of message lists will be discussed further in the context of the BBA in Section 9.4. While the match set $[M]$ is a ubiquitous component of Michigan-style systems the action set $[A]$ only appeared after the removal of the internal message list. $[A]$ provided a physical location with which to track classifiers involved in sending action messages to the effector. Concurrently, a *previously active action set* $[A]_{t-1}$ was implemented to keep track of the last set of rules to have been placed in $[A]$. This temporary storage allows reward to be implicitly passed up the activating chain of rules and was aptly referred to as an implicit bucket brigade. For LCSs designed for supervised learning (e.g., NEWBOOLE [46] and UCS [117]), the sets of the performance component take on a somewhat different appearance, with $[A]$ being replaced with a correct set $[C]$, and not-correct set $Not\ [C]$ to accommodate the different learning style. Going beyond the basic set structure, XCS also utilized a *prediction array* added to modify both action selection and credit assignment [22]. In brief, the prediction array calculates a system prediction $P(a_j)$ for each action a_j represented in $[M]$. $P(a_j)$ represents the strength (the likely benefit) of selecting the given a_j based on the collective knowledge of all classifiers in $[M]$ that advocate a_j. Its purpose will become clearer in Section 9.4. Modern LCS selection mechanisms serve three main functions: (1) using the classifiers to make an action decision, (2) choosing parent rules for GA "mating", and (3) picking out classifiers to be deleted. Four selection mechanisms are frequently implemented to perform these functions. They include: (1) *purely stochastic* (random)

selection, (2) *deterministic selection*—the classifier with the largest fitness or prediction (in the case of action selection) is chosen, (3) *proportionate selection* (often referred to as roulette-wheel selection)—where the chances of selection are proportional to fitness, and (4) *tournament selection*—a number of classifiers (s) are selected at random and the one with the largest fitness is chosen. Recent studies have examined selection mechanisms and noted the advantages of tournament selection [119, 222–225]. It should be noted that when selecting classifiers for deletion, any fitness-based selection will utilize the inverse of the fitness value so as to remove less-fit classifiers. Additionally, when dealing with action selection, selection methods will rely on the prediction parameter instead of fitness. Also, it is not uncommon, especially in the case of action selection, to alternate between different selection mechanisms (e.g., MCS alternates between stochastic and deterministic schemes from one iteration to the next). Sometimes this method is referred to as the pure explore/exploit scheme [19]. While action selection occurs once per iteration, and different GA triggering mechanisms are discussed in Section 9.5, deletion occurs under the following circumstances; the global population (N) is bound, and new classifiers are being added to a population that has reached (N). At this point, a corresponding number of classifiers must be deleted. This may occur following covering (explained in Section 4) or after the GA has been triggered. Of final note is a bidding mechanism. Bidding was used by Holland's LCS to select and allow the strongest n classifiers in $[M]$ to post their action messages to the message list. Additionally either bidding or a conflict resolution module [52] may be advocated for action selection from the message list. A classifier's "bid" is proportional to the product of its strength and specificity. The critical role of bidding in the BBA is discussed in the next section.

9.4. Reinforcement Component. Different LCS credit assignment strategies are bound by a similar objective (to distribute reward), but the varying specifics regarding where/when they are called, what parameters are included and updated, and what formulas are used to perform those updates have lead to an assortment of methodologies, many of which have only very subtle differences. As a result, the nomenclature used to describe an LCS credit assignment scheme is often vague (e.g., Q-Learning-Based [22]) and occasionally absent. Therefore to understand the credit assignment used in a specific system, we refer readers to the relevant primary source. Credit assignment can be as simple as updating a single value (as is implemented in MCS), or it may require a much more elaborate series of steps (e.g., BBA). We briefly review two of the most historically significant credit assignment schemes, that is, the BBA and XCS's Q-Learning-based strategy.

9.4.1. Bucket Brigade. "*The bucket brigade* [BBA] *may most easily be viewed as an information economy where the right to trade information is bought and sold by classifiers. Classifiers form a chain of middlemen from information manufacturer* ((detectors of) *the environment) to information consumer (the effectors)*"—*Goldberg*. [8] The BBA, as described in the following lines, involves both performance and reinforcement components. The following steps outline its progression over a single time iteration (t): It should be noted that within a given (t), the message list can receive only a limited number of input messages as well as a limited number of classifier postings. Also, when a classifier posts a message to the current message list it is said to have been "activated" during (t).

(1) Post one or more messages from the detector to the current message list $[ML]$.

(2) Compare all messages in $[ML]$ to all conditions in $[P]$ and record all matches in $[M]$.

(3) Post "action" messages of the highest bidding classifiers of $[M]$ onto $[ML]$.

(4) Reduce the strengths of these activated classifiers $\{C\}$ by the amount of their respective bids $\mathrm{B}(t)$ and place those collective bids in a "bucket" B_{total}. (paying for the privilege of posting a new message).

(5) Distribute B_{total} evenly over the previously activated classifiers $\{C'\}$. (suppliers $\{C'\}$ are rewarded for setting up a situation usable by $\{C\}$).

(6) Replace messages in $\{C'\}$ with those in $\{C\}$ and clear $\{C\}$. (updates record of previously activated classifiers).

(7) $[ML]$ is processed through the output interface(effector) to provoke an action.

(8) This step occurs if a reward is returned by the environment. The reward value is added to the strength of all classifiers in $\{C\}$ (the most recently activated classifiers receive the reward).

"*Whenever a classifier wins a bidding competition, it initiates a transaction in which it pays out part of its strength to its suppliers and then receives similar payments from its consumers.* [This] *strength is a kind of capital. If a classifier receives more from its consumers than it paid out, it has made a profit, that is its strength is increased*"—*Holland* [30]. The update for any given classifier can be summarized by the following equation where $S(t)$ is classifier strength, $B(t)$ is the bid of the classifier (see step 4), $P(t)$ is the sum of all payments made to this classifier by $\{C\}$ (see step 5), and $R(t)$ is any reward received (see step 8):

$$S(t+1) = S(t) - B(t) + P(t) + R(t). \tag{2}$$

The desired effect of this cycle is to enable classifiers to pass reward (when received) along to classifiers that may have helped make that reward possible. See [8, 38] for more details.

9.4.2. Q-Learning-Based. The Q-learning-based strategy used by XCS is an archetype of modern credit assignment. First off, it should be noted that the performance component of XCS is similar to that described for MCS (although XCS adds a prediction array and an $[A]_{t-1}$, both imperative to the

credit assignment strategy). Each classifier (j) in XCS tracks four parameters: prediction (p), prediction error (ϵ), fitness (F), and experience (e). The update of these parameters takes place in $[A]_{t-1}$ as follows.

(1) Each rule's ϵ is updated: $\epsilon_j \leftarrow \epsilon_j + \beta(|P - p_j|) - \epsilon_j)$.

(2) Rule predictions are updated: $p_j \leftarrow p_j + \beta(P - p_j)$.

(3) Each rule's accuracy is determined: $\kappa_j = \exp[(\ln \alpha)(\epsilon_j - \epsilon_0)/\epsilon_0)]$ for $\epsilon_j > \epsilon_0$ otherwise 1.

(4) A relative accuracy κ'_j, is determined for each rule: $\kappa'_j = \kappa_j / \Sigma\kappa_{[A]t-1}$.

(5) Each rule's F is updated using κ'_j: $F_j \leftarrow F_j + \beta(\kappa'_j - F_j)$.

(6) Increment e for all classifiers in $[A]$.

β is a learning rate constant ($0 \leq \beta \leq 1$) while ϵ_0 and α are accuracy function parameters. The procedure used to calculate p, ϵ, and F is the widely implemented Widrow-Hoff formula [226] (also known as Least Mean Square) seen here:

$$x \longleftarrow x + \beta(y - x). \tag{3}$$

An important caveat is that initially p, ϵ, and F are actually updated by respectively averaging together their current and previous values. It is only after a classifier has been adjusted at least $1/\beta$ times that the Widrow-Hoff procedure takes over parameter updates. This technique, referred to as "moyenne adaptive modifee" [227], is used to make early parameter values move more quickly to their "true" average values in an attempt at avoiding the arbitrary nature of early parameter values. The direct influence of Q-learning on this credit assignment scheme is found in the update of p_j, which takes the maximum prediction value from the prediction array, discounts it by a factor, and adds in any external reward received in the previous time. The resulting value, which Wilson calls P (see steps 1 and 2), is somewhat analogous to Q-Learning's Q-values. Also observe that a classifier's fitness is dependent on its ability to make accurate predictions, but is not proportional to the prediction value itself. For further perspective on basic modern credit assignment strategy see [19, 22, 228].

9.4.3. More Credit Assignment. Many other credit assignment schemes have been implemented. For example Pitt-style systems track credit at the level of entire rule sets as opposed to assigning parameters to individual rules. Supervised learning systems like UCS have basically eliminated the reinforcement component (as it is generally understood) and instead maintains and updates a single accuracy parameter [117]. Of course, many other credit assignment and parameter update strategies have been suggested and implemented. Here we list some of these strategies: epochal [3], implicit bucket brigade [44], one-step payoff-penalty [43], symmetrical payoff-penalty [46], hybrid bucket brigade-backward averaging (BB-BA) algorithm [229], nonbucket brigade temporal difference method [55], action-oriented credit assignment [230, 231], QBB [19], average reward [114], gradient descent [232, 233], eligibility traces [234], Bayesian update [147], least squares update [235], and Kalman filter update [235].

9.5. Discovery Components. A standard discovery component is comprised of a GA and a covering mechanism. The primary role of the covering mechanism is to ensure that there is at least one classifier in $[P]$ that can handle the current input. A new rule is generated by adding some number of #'s (wild cards) at random to the input string and then selecting a random action (i.e., the new rule "0#110#0-01" might be generated from the input string 0011010) [44]. The random action assignment has been noted to aid in escaping *loops* [22]. The parameter value(s) of this newly generated classifier are set to the population average. Covering might also be used to initialize classifier populations on the fly, instead of starting the system with an initialized population of maximum size. The covering mechanism can be implemented differently by modifying the frequency at which #'s are added to the new rule [22, 43, 44], altering how a new rule's parameters are calculated [19], and expanding the instances in which covering is called (e.g., ZCS will "cover" when the total strength of $[M]$ is less than a fraction of the average seen in $[P]$ [19]). Covering does more than just handle an unfamiliar input by assigning a random action. "*Covering allows the system* [to] *test a hypothesis (the condition-action relation expressed by the created classifier) at the same time*" [19]. The GA discovers rules by building upon knowledge already in the population (i.e., the fitness). The vast majority of LCS implementations utilize the GA as its primary discovery component. Specifically, LCSs typically use *steady state* GAs, where rules are changed in the population individually without any defined notion of a generation. This differs from *generational* GAs where all or an important part of the population is renewed from one generation to the next [9]. GAs implemented independent of an LCS are typically *generational*. In selecting an algorithm to address a given problem, an LCS algorithm that incorporates a GA would likely be preferable to a straightforward GA when dealing with more complex decision making tasks, specifically ones where a single rule cannot effectively represent the solution, or in problem domains where adaptive solutions are needed. Like the covering mechanism, the specifics of how a GA is implemented in an LCS may vary from system to system. Three questions seem to best summarize these differences: (1) Where is the GA applied? (2) When is GA fired? and (3) What operators does it employ? The set of classifiers to which the GA is applied can have a major impact on the evolutionary pressure it produces. While early systems applied the GA to $[P]$ [3], the concepts of *restricted mating* and niching [42] moved its action to $[M]$ and then later to $[A]$, where it is typically applied in modern systems (see Table 1). For more on niching see [22, 42, 236, 237]. The firing of the GA can simply be controlled by a parameter (g) which represents the probability of firing the GA on a given time step (t), but in order to more fairly allocate the application of the GA to different developing niches, it can be triggered by a tracking parameter [22, 47]. Crossover and mutation are the two most recognizable operators of the GA. Both mechanisms are controlled by parameters representing their respective probabilities of being called. Historically, most early LCSs used simple one-point crossover, but interest

in discovering complex "building blocks" [8, 238, 239] has led to examining two-point, uniform, and informed crossover (based on estimation of distribution algorithms) as well [239]. Additionally, a smart crossover operator for a Pitt-style LCS has also been explored [240]. The GA is a particularly important component of Pitt-style systems which relies on it as its only adaptive process. Oftentimes it seems more appropriate to classify Pitt-style systems simply as an evolutionary algorithm as opposed to what is commonly considered to be a modern LCS [9, 52]. Quite differently, the GA is absent from ACSs, instead relying on nonevolutionary discovery mechanisms [24, 99, 103, 118, 132].

9.6. Beyond the Basics. This section briefly identifies LCS implementation themes that extend beyond the basic framework such as the addition of memory, multilearning classifier systems, multiobjectivity, and data concerns. While able to deal optimally with Markov problems, the major drawback of simpler systems like ZCS and XCS was their relative inability to handle non-Markov problems. One of the methodologies that were developed to address this problem was the addition of memory via an internal register (i.e., a non-message-list memory mechanism) which can store a limited amount of information reguarding a previous state. Systems adopting memory include ZCSM [78], XCSM [83], and XCSMH [90]. Another area that has drawn attention is the development of what we will call "multilearning classifier systems" (M-LCSs) (i.e., multiagent LCSs, ensemble LCSs, and distributed LCSs), which run more than one LCS at a time. Multiagent LCSs were designed to model multiagent systems that intrinsically depend on the interaction between multiple intelligent agents (e.g., game-play) [94, 241, 242]. Ensemble LCSs were designed to improve algorithmic performance and generalization via parallelization [140–142, 153, 243–245]. Distributed LCSs were developed to assimilate distributed data (i.e. data coming from different sources) [137, 246, 247]. Similar to the concept of M-LCS, Ranawana and Palade published a detailed review and roadmap on *multiclassifier systems* [248]. Multiobjective LCSs discussed in [249] explicitly address the goals implicitly held by many LCS implementations (i.e., accuracy, completeness, minimalism) [97, 120]. A method that has been explored to assure minimalism is the application of a rule compaction algorithm for the removal of redundant or strongly overlapping classifiers [96, 122, 250, 251]. Some other dataset issues which have come up especially in the context of data mining include missing data [130, 252], unbalanced data [253], dataset size [254], and noise [120]. Some other interesting algorithmic innovations include partial matching [211], endogenous fitness [255], self-adapted parameters [219, 256–259], abstraction, [260], and macroclassifiers [22].

10. Conclusion

"Classifier Systems are a quagmire—a glorious, wondrous, and inventing quagmire, but a quagmire nonetheless"—Goldberg [261]. This early perspective was voiced at a time when LCSs were still quite complex and nebulously understood. Structurally speaking, the LCS algorithm is an interactive merger of other stand-alone algorithms. Therefore, its performance is dependent not only on individual components but also on the interactive implementation of the framework. The independent advancement of GA and learning theory (in and outside the context of LCS) has inspired an inovative generation of systems, that no longer merrit the label of a "quagmire". The application of LCSs to a spectrum of problem domains has generated a diversity of implementations. However it is not yet obvious which LCSs are best suited to address a given domain, nor how to best optimize performance. The basic XCS architecture has not only revitalized interest in LCS research, but has become the model framework upon which many recent modifications or adaptations have been built. These expansions are intended to address inherent limitations in different problem domains, while sticking to a trusted and recognizable framework. But will this be enough to address relevant real-world applications? One of the greatest challenges for LCS might inevitably be the issue of scalability as the problems we look to be solved increase exponentially in size and complexity. Perhaps, as was seen pre-empting the development of ZCS and XCS, the addition of heuristics to an accepted framework might again pave the way for some novel architecture(s). Perhaps there will be a return to Holland-style architectures as the limits of XCS-based systems are reached. A number of theoretical questions should also be considered: What are the limits of the LCS framework? How will LCS take advantage of advancing computational technology? How can we best identify and interpret an evolved population (solution)? And how can we make using the algorithm more intuitive and/or interactive? Beginning with a gentle introduction, this paper has described the basic LCS framework, provided a historical review of major advancements, and provided an extensive roadmap to the problem domains, optimization, and varying components of different LCS implementations. It is hoped that by organizing many of the existing components and concepts, they might be recycled into or inspire new systems which are better adapted to a specific problem domain. The ultimate challenge in developing an optimized LCS is to design an implementation that best arranges multiple interacting components, operating in concert, to evolve an accurate, compact, comprehensible solution, in the least amount of time, making efficient use of computational power. It seems likely that LCS research may culminate in one of two ways. Either there will be some dominant core platform, flexibly supplemented by a variety of problem specific modifiers, or will there be a handful of fundamentally different systems that specialize to different problem domains. Whatever the direction, it is likely that LCS will continue to evolve and inspire methodologies designed to address some of the most difficult problems ever presented to a machine.

11. Resources

For various perspectives on the LCS algorithm, we refer readers to the following review papers [4, 7, 9, 45, 52, 54, 165, 178, 262]. Together, [45, 165] represent two decade-long consecutive summaries of current systems, unsolved

problems and future challenges. For comparative system discussions see [102, 192, 263, 264]. For a detailed summary of LCS community resources as of (2002) see [265]. For a detailed examination of the design and analysis of LCS algorithms see [266].

References

[1] J. Holland, *Hidden Order: How Adaptation Builds Complexity*, Addison-Wesley, Reading, Mass, USA, 1996.

[2] H. H. Dam, H. A. Abbass, C. Lokan, and X. Yao, "Neural-based learning classifier systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, no. 1, pp. 26–39, 2008.

[3] J. Holland and J. Reitman, "Cognitive systems based on adaptive agents," in *Pattern-Directed Inference Systems*, D. A. Waterman and F. Inand, Eds., Hayes-Roth, 1978.

[4] J. H. Holmes, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, "Learning classifier systems: new models, successful applications," *Information Processing Letters*, vol. 82, no. 1, pp. 23–30, 2002.

[5] M. Minsky, "Steps toward artificial intelligence," *Proceedings of the IRE*, vol. 49, no. 1, pp. 8–30, 1961.

[6] J. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, Mich, USA, 1975.

[7] L. Bull and T. Kovacs, *Foundations of Learning Classifier Systems*, Springer, Berlin, Germany, 2005.

[8] D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Longman, Boston, Mass, USA, 1989.

[9] O. Sigaud and S. Wilson, "Learning classifier systems: a survey," *Soft Computing*, vol. 11, no. 11, pp. 1065–1078, 2007.

[10] J. Holmes, "Learning classifier systems applied to knowledge discovery in clinical research databases," in *Learning Classifier Systems, from Foundations to Applications*, pp. 243–262, 2000.

[11] D. Goldberg, *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.

[12] P. Langley, *Elements of Machine Learning*, Morgan Kaufmann, San Francisco, Calif, USA, 1996.

[13] M. B. Harries, C. Sammut, and K. Horn, "Extracting hidden context," *Machine Learning*, vol. 32, no. 2, pp. 101–126, 1998.

[14] S. Ben-David, E. Kushilevitz, and Y. Mansour, "Online learning versus offline learning," *Machine Learning*, vol. 29, no. 1, pp. 45–63, 1997.

[15] R. S. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, Mass, USA, 1998.

[16] M. Harmon and S. Harmon, "Reinforcement Learning: A Tutorial," December 1996.

[17] J. Wyatt, "Reinforcement learning: a brief overview," in *Foundations of Learning Classifier Systems*, pp. 179–202, 2005.

[18] L. Bull, "Two simple learning classifier systems," in *Foundations of Learning Classifier Systems*, pp. 63–89, 2005.

[19] S. W. Wilson, "ZCS: a zeroth level classifier system," *Evolutionary Computation*, vol. 2, no. 1, pp. 1–18, 1994.

[20] D. Goldberg, *Computer-aided gas pipeline operation using genetic algorithms and machine learning*, Ph.D. thesis, Department Civil Engineering, University of Michigan, Ann Arbor, Mich, USA, 1983.

[21] J. Baker, "Reducing bias and inefficiency in the selection algorithm," in *Proceedings of the 2nd International Conference on Genetic Algorithms on Genetic Algorithms and Their Application*, pp. 14–21, 1987.

[22] S. W. Wilson, "Classifier fitness based on accuracy," *Evolutionary Computation*, vol. 3, no. 2, pp. 149–175, 1995.

[23] S. Smith, *A learning system based on genetic adaptive algorithms*, Ph.D. thesis, University of Pittsburgh, Pittsburgh, Pa, USA, 1980.

[24] W. Stolzmann, "Anticipatory classifier systems," in *Proceedings of the 3rd Annual Genetic Programming Conference*, pp. 658–664, 1998.

[25] J. Holland, "Adaptation," *Progress in Theoretical Biology*, vol. 4, pp. 263–293, 1976.

[26] G. G. Robertson and R. L. Riolo, "A tale of two classifier systems," *Machine Learning*, vol. 3, no. 2-3, pp. 139–159, 1988.

[27] S. Smith, "Flexible learning of problem solving heuristics through adaptive search," in *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pp. 422–425, 1983.

[28] C. Congdon, *A comparison of genetic algorithms and other machine learning systems of a complex classification task from common disease research*, Ph.D. thesis, University of Michigan, 1995.

[29] R. Riolo, *Empirical studies of default hierarchies and sequences of rules in learning classifier systems*, Doctoral dissertation, University of Michigan, Ann Arbor, Mich, USA, 1988.

[30] J. Holland, "Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems," *Machine Learning*, vol. 2, pp. 593–623, 1986.

[31] J. Holland, "A mathematical framework for studying learning in classifier systems," *Physica D*, vol. 2, no. 1–3, pp. 307–317, 1986.

[32] J. H. Holland, "Adaptive algorithms for discovering and using general patterns in growing knowledge bases," *International Journal of Policy Analysis and Information Systems*, vol. 4, no. 3, pp. 245–268, 1980.

[33] J. Holland, "Adaptive knowledge acquisition," unpublished research proposal, 1980.

[34] J. Holland, "Genetic algorithms and adaptation," Tech. Rep. 34, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor, Mich, USA, 1981.

[35] J. Holland, "Induction in artificial intelligence," Tech. Rep., Department of Computer and Communication Sciences, University of Michigan, Ann Arbor, Mich, USA, 1983.

[36] J. Holland, "A more detailed discussion of classifier systems," Tech. Rep., Department of Computer and Communication Sciences, University of Michigan, Ann Arbor, Mich, USA, 1983.

[37] J. Holland, "Genetic algorithms and adaptation," in *Adaptive Control of Ill-Defined Systems*, pp. 317–333, 1984.

[38] J. Holland, "Properties of the Bucket brigade," in *Proceedings of the 1st International Conference on Genetic Algorithms*, pp. 1–7, 1985.

[39] J. Holland, "A mathematical framework for studying learning in classifier systems," Research Memo RIS-25r, The Rowland Institute for Science, Cambridge, Mass, USA, 1985.

[40] J. Holland, "Genetic algorithms and classifier systems: foundations and future directions," in *Proceedings of the 2nd International Conference on Genetic Algorithms and Their Application*, pp. 82–89, 1987.

[41] A. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, pp. 211–232, 1959.

[42] L. B. Booker, "Intelligent behavior as an adaptation to the task environment," 1982.

[43] S. W. Wilson, "Classifier systems and the animat problem," *Machine Learning*, vol. 2, no. 3, pp. 199–228, 1987.

[44] S. W. Wilson, "Knowledge growth in an artificial animal," in *Proceedings of the 1st International Conference on Genetic Algorithms and Their Application*, pp. 16–23, 1985.

[45] S. W. Wilson and D. Goldberg, "A critical review of classifier systems," in *Proceedings of the 3rd International Conference on Genetic Algorithms and Their Application*, pp. 244–255, 1989.

[46] P. Bonelli, A. Parodi, S. Sen, and S. Wilson, "NEWBOOLE: a fast GBML system," in *Proceedings of the 7th International Conference on Machine Learning*, pp. 153–159, 1990.

[47] L. Booker, "Triggered rule discovery in classifier systems," in *Proceedings of the 3rd International Conference on Genetic Algorithms*, pp. 265–274, 1989.

[48] M. Valenzuela-Rendon, "The fuzzy classifier system: a classifier system for continuously varying variables," in *Proceedings of the 4th International Conference on Genetic Algorithm*, pp. 346–353, 1991.

[49] A. Bonarini, "An introduction to learning fuzzy classifier systems," in *Proceedings of the International Workshop on Learning Classifier Systems (IWLCS '00)*, vol. 1813 of *Lecture Notes in Artificial Intelligence*, pp. 83–104, 2000.

[50] R. Riolo, "Bucket brigade performance: I. Long sequences of classifiers," in *Proceedings of the 2nd International Conference on Genetic Algorithms and Their Application*, pp. 184–195, Lawrence Erlbaum, Mahwah, NJ, USA, 1987.

[51] R. Riolo, "Bucket brigade performance: II. Default hierarchies," in *Proceedings of the 2nd International Conference on Genetic Algorithms and Their Application*, pp. 196–201, Lawrence Erlbaum, Mahwah, NJ, USA, 1987.

[52] P. Lanzi and R. Riolo, "Recent trends in learning classifier systems research," in *Advances in Evolutionary Computing: Theory and Applications*, Natural Computing Series, pp. 955–988, 2003.

[53] A. Barry, "Hierarchy formation within classifier systems: a review," in *Proceedings of the 1st International Conference on Evolutionary Algorithms and Their Application (EVCA '96)*, pp. 195–211, 1996.

[54] P. L. Lanzi, "Learning classifier systems: then and now," *Evolutionary Intelligence*, vol. 1, no. 1, pp. 63–82, 2008.

[55] R. Riolo, "Lookahead planning and latent learning in a classifier system," in *Proceedings of the 1st International Conference on Simulation of Adaptive Behavior on from Animals to Animats*, pp. 316–326, 1991.

[56] J. Schaffer and J. Grefenstette, "Multi-objective learning via genetic algorithms," in *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pp. 593–595, 1985.

[57] D. Greene, "Automated knowledge acquisition: overcoming the expert system bottleneck," in *Proceedings of the 8th International Conference on Information Systems*, J. DeGross and C. Kriebel, Eds., pp. 107–117, Pittsburgh, Pa, USA, 1987.

[58] J. J. Grefenstette, "Credit assignment in rule discovery systems based on genetic algorithms," *Machine Learning*, vol. 3, no. 2-3, pp. 225–245, 1988.

[59] L. B. Booker, "Classifier systems that learn internal world models," *Machine Learning*, vol. 3, no. 2-3, pp. 161–192, 1988.

[60] J. Grefenstette, "Incremental learning of control strategies with genetic algorithms," in *Proceedings of the 6th International Workshop on Machine Learning*, pp. 340–344, 1989.

[61] J. Grefenstette, "The evolution of strategies for multiagent environments," *Adaptive Behavior*, vol. 1, no. 1, p. 65, 1992.

[62] J. Grefenstette, *The User's Guide to SAMUEL-97: An Evolutionary Learning System*, Navy Center for Applied Research in Artificial Intelligence, Naval Research Laboratory, Washington, DC, USA, 1997.

[63] L. Shu and J. Schaeffer, "HCS: adding hierarchies to classifier systems," in *Proceedings of the 4th International Conference on Genetic Algorithms and Their Application*, pp. 339–345, 1991.

[64] M. Dorigo and E. Sirtori, "Alecsys: a parallel laboratory for learning classifier systems," in *Proceedings of the 4th International Conference on Genetic Algorithms*, 1991.

[65] M. Dorigo, "Alecsys and the autonoMouse: learning to control a real robot by distributed classifier systems," *Machine Learning*, vol. 19, no. 3, pp. 209–240, 1995.

[66] K. De Jong and W. Spears, "Learning concept classification rules using genetic algorithms," in *Proceedings of the 12th International Conference on Artificial Intelligence (IJCAI '91)*, vol. 2, 1991.

[67] K. A. De Jong, W. M. Spears, and D. F. Gordon, "Using genetic algorithms for concept learning," *Machine Learning*, vol. 13, no. 2-3, pp. 161–188, 1993.

[68] C. Janikow, *Inductive Learning of decision rules in attribute-based examples: a knowledge-intensive genetic algorithm approach*, Ph.D. thesis, University of North Carolina, 1991.

[69] C. Z. Janikow, "A knowledge-intensive genetic algorithm for supervised learning," *Machine Learning*, vol. 13, no. 2-3, pp. 189–228, 1993.

[70] D. Greene, *Inductive knowledge acquisition using genetic adaptive search*, Ph.D. thesis, 1992.

[71] D. P. Greene and S. F. Smith, "Competition-based induction of decision models from examples," *Machine Learning*, vol. 13, no. 2-3, pp. 229–257, 1993.

[72] A. Giordana and L. Saitta, "REGAL: an integrated system for learning relations using genetic algorithms," in *Proceedings of the 2nd International Workshop on Multistrategy Learning*, pp. 234–249, 1993.

[73] A. Giordana, L. Saitta, and F. Zini, "Learning disjunctive concepts with distributed genetic algorithms," in *Proceedings of the 1st IEEE Conference on Evolutionary Computation*, vol. 1, pp. 115–119, Orlando, Fla, USA, June 1994.

[74] A. Giordana and F. Neri, "Search-intensive concept induction," *Evolutionary Computation*, vol. 3, no. 4, pp. 375–416, 1995.

[75] A. Bonarini, "ELF: learning incomplete fuzzy rule sets for an autonomous robot," in *Proceedings of the ELITE Foundation (EUFIT '93)*, pp. 69–75, Aachen, Germany, 1993.

[76] A. Bonarini, "Some methodological issues about designing autonomous agents which learn their behaviors: the ELF experience," in *Proceedings of the Cybernetics and Systems Research*, R. Trappl, Ed., pp. 1435–1442, 1994.

[77] A. Bonarini, "Evolutionary learning of fuzzy rules: competition and cooperation," in *Fuzzy Modelling: Paradigms and Practice*, pp. 265–284, 1996.

[78] D. Cliff and S. Ross, "Adding memory to ZCS," *Adaptive Behavior*, vol. 3, no. 2, pp. 101–150, 1994.

[79] I. Flockhart and N. Radcliffe, "GA-MINER: parallel data mining with hierarchical genetic algorithms-final report," EPCC AIKMS GA-Miner-Report 1.

[80] I. Flockhart, N. Radcliffe, E. Simoudis, J. Han, and U. Fayyad, "A genetic algorithm-based approach to data mining," in *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD '96)*, pp. 299–302, 1996.

[81] J. Holmes, "A genetics-based machine learning approach to knowledge discovery in clinical data," in *Proceedings of the AMIA Anual Symposim*, pp. 883–883, 1996.

[82] J. Holmes, "Discovering risk of disease with a learning classifier system," in *Proceedings of the 7th International Conference on Genetic Algorithms (ICGA '97)*, pp. 426–433, 1997.

[83] P. Lanzi, "Adding memory to XCS," in *Proceedings of IEEE Conference on Evolutionary Computation (ICEC '98)*, pp. 609–614, 1998.

[84] P. Lanzi, "An analysis of the memory mechanism of XCSM," *Genetic Programming*, vol. 98, pp. 643–651, 1998.

[85] A. Tomlinson and L. Bull, "A corporate classifier system," Lecture Notes in Computer Science, pp. 550–559, 1998.

[86] A. Tomlinson and L. Bull, "A zeroth level corporate classifier system," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '99)*, pp. 306–313, 1999.

[87] W. Stolzmann, "An introduction to anticipatory classifier systems," in *Learning Classifier Systems: From Foundations to Applications*, Lecture Notes in Computer Science, pp. 175–194, 2000.

[88] W. Browne, *The Development of an Industrial Learning Classifier System for Application to a Steel Hot Strip Mill*, Ph.D. thesis, Division of Mechanical Engineering and Energy Studies, University of Wales, Cardiff, UK, 1999.

[89] W. N. L. Browne, K. M. Holford, C. J. 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*, vol. 13, no. 1, pp. 25–36, 2000.

[90] P. Lanzi and S. Wilson, "Toward optimal classifier system performance in non-Markov environments," *Evolutionary Computation*, vol. 8, no. 4, pp. 393–418, 2000.

[91] A. Tomlinson and L. Bull, "A corporate XCS," in *Proceedings of the International Workshop on Learning Classifier Systems*, Lecture Notes in Computer Science, pp. 195–208, 2000.

[92] S. W. Wilson, "Get real! XCS with continuous-valued inputs," in *Proceedings of the 3rd International Workshop on Advances in Learning Classifier Systems*, Lecture Notes in Computer Science, pp. 209–222, 2000.

[93] D. Walter and C. K. Mohan, "ClaDia: a fuzzy classifier system for disease diagnosis," in *Proceedings of the IEEE Conference on Evolutionary Computation (ICEC '00)*, vol. 2, pp. 1429–1435, 2000.

[94] K. Takadama, T. Terano, and K. Shimohara, "Learning classifier systems meet multiagent environments," in *Proceedings of the 3rd International Workshop on Learning Classifier Systems (IWLCS '00)*, L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds., pp. 192–210, Springer, 2000.

[95] S. W. Wilson, "Mining oblique data with XCS," in *Proceedings of the 3rd International Workshop on Advances in Learning Classifier Systems*, pp. 158–176, 2000.

[96] S. W. Wilson, "Compact rulesets from XCSI," in *Proceedings of the 4th International Workshop on Advances in Learning Classifier Systems*, pp. 197–210, 2001.

[97] E. Bernado-Mansilla and J. Garrell-Guiu, "MOLeCS: a multiObjective learning classifier system," in *Proceedings of the Conference on Genetic and Evolutionary Computation*, vol. 1, 2000.

[98] E. Mansilla and J. Guiu, "MOLeCS: using multiobjective evolutionary algorithms for learning," in *Proceedings of the 1st International Conference on Evolutionary Multi-Criterion Optimization*, Lecture Notes in Computer Science, pp. 696–710, 2001.

[99] P. Gérard and O. Sigaud, "YACS: combining dynamic programming with generalization in classifiersystems," in *Proceedings of the 3rd International Workshop on Advances in Learning Classifier Systems (IWLCS '00)*, pp. 52–69, 2000.

[100] P. Gérard, W. Stolzmann, and O. Sigaud, "YACS : a new learning classifier system using anticipation," *Soft Computing-A*, vol. 6, no. 3, pp. 216–228, 2002.

[101] T. Kovacs, *A comparison of strength and accuracy-based fitness in learning classifier systems*, Ph.D. thesis, University of Birmingham, Birmingham, UK, 2001.

[102] T. Kovacs, "Two views of classifier systems," in *Advances in Learning Classifier Systems*, Lecture Notes in Computer Science, pp. 74–87, 2002.

[103] M. Butz, "Biasing exploration in an anticipatory learning classifier system," in *Proceedings of the 4th International Workshop on Advances in Learning Classifier Systems*, vol. 2321 of *Lecture Notes in Computer Science*, pp. 3–22, 2001.

[104] M. V. Butz and J. Hoffmann, "Anticipations control behavior: animal behavior in an anticipatory learning classifier system," *Adaptive Behavior*, vol. 10, no. 2, pp. 75–96, 2002.

[105] S. Picault and S. Landau, "ATNoSFERES: a Darwinian evolutionary model for individual or collective agent behavior," Tech. Rep., LIP6, Paris, France, 2001.

[106] S. Landau, S. Picault, and A. Drogoul, "ATNoSFERES: a model for evolutive agent behaviors," in *Proceedings of the Symposium on Adaptive Agents and Multi-Agent Systems (AISB '01)*, vol. 1, 2001.

[107] S. Landau, S. Picault, O. Sigaud, and P. Gérard, "A comparison between ATNoSFERES and XCSM," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 926–933, 2002.

[108] S. Landau, S. Picault, O. Sigaud, and P. Gerard, "Further comparison between ATNoSFERES and XCSM," in *Proceedings of the 5th International Workshop on Learning Classifier Systems*, vol. 2661 of *Lecture Notes in Computer Science*, pp. 99–117, 2003.

[109] S. Landau, O. Sigaud, S. Picault, and P. Gerard, "An experimental comparison between ATNoSFERES and ACS," in *Proceedings of the International Workshop on Learning Classifier Systems*, vol. 4399 of *Lecture Notes in Computer Science*, pp. 144–160, 2007.

[110] X. Llorà, J. Garrell, et al., "Knowledge-independent data mining with fine-grained parallel evolutionary algorithms," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '01)*, pp. 461–468, Morgan Kaufmann, San Francisco, Calif, USA, 2001.

[111] T. Kovacs, *Genetic based machine learning using fine-grained parallelism for data mining*, Ph.D. thesis, Enginyeria i Arquitectura La Salle, Ramon Llull University, 2002.

[112] X. Llorà and J. Guiu, "Coevolving different knowledge representations with fine-grained parallel learning classifier systems," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '02)*, pp. 934–941, Morgan Kaufmann, San Francisco, Calif, USA, 2002.

[113] S. W. Wilson, "Classifiers that approximate functions," *Natural Computing*, vol. 1, no. 2, pp. 211–234, 2002.

[114] K. Tharakunnel and D. Goldberg, "XCS with average reward criterion in multi-step environment," Tech. Rep., Illinois Genetic Algorithms Laboratory (IlliGAL), Department of General Engineering, University of Illinois at Urbana-Champaign, 2002.

[115] J. Hurst, L. Bull, and C. Melhuish, "TCS learning classifier system controller on a real robot," in *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature (PPSN '02)*, Lecture Notes in Computer Science, pp. 588–600, Granada, Spain, September 2002.

[116] L. Bull and T. O'Hara, "Accuracy-based neuro and neuro-fuzzy classifier systems," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '02)*, pp. 905–911, 2002.

[117] E. Bernadó-Mansilla and J. M. Garrell-Guiu, "Accuracy-based learning classifier systems: models, analysis and applications to classification tasks," *Evolutionary Computation*, vol. 11, no. 3, pp. 209–238, 2003.

[118] M. Butz and D. Goldberg, "Generalized state values in an anticipatory learning classifier system," in *Proceedings of the 7th International Conference on Simulation of Adaptive Behavior in Anticipatory Learning Systems*, Lecture Notes in Computer Science, pp. 282–302, 2003.

[119] M. Butz, K. Sastry, and D. Goldberg, "Tournament selection: stable fitness pressure in XCS," in *Proceedings of the Genetic and Evolutionary Computation Conference*, Lecture Notes in Computer Science, pp. 1857–1869, 2003.

[120] X. Llorà and D. E. Goldberg, "Bounding the effect of noise in multiobjective learning classifier systems," *Evolutionary Computation*, vol. 11, no. 3, pp. 279–298, 2003.

[121] L. Bull, "A simple accuracy-based learning classifier system," Learning Classifier Systems Group Technical Report UWELCSG03-005, University of the West of England, Bristol, UK, 2003.

[122] P. W. Dixon, D. W. Corne, and M. J. Oates, "A ruleset reduction algorithm for the XCS learning classifier system," in *Learning Classifier Systems*, vol. 2661 of *Lecture Notes in Computer Science*, pp. 20–29, 2003.

[123] L. Bull, "Lookahead and latent learning in a simple accuracy-based classifier system," in *Proceedings of the 8th International Conference on Parallel Problem Solving from Nature*, Lecture Notes in Computer Science, pp. 1042–1050, 2004.

[124] A. Gaspar and B. Hirsbrunner, "PICS: Pittsburgh immune classifier system," in *Proceedings of the AISB Symposium on the Immune System and Cognition*, Leeds, UK, March 2004.

[125] A. Gaspar and B. Hirsbrunner, "From optimization to learning in changing environments: the Pittsburgh immune classifier system," in *Proceedings of the 1st International Conference on Artificial Immune Systems (ICARIS '02)*, September 2002.

[126] J. Hurst and L. Bull, "A self-adaptive neural learning classifier system with constructivism for mobile robot control," in *Proceedings of the 8th International Conference on Parallel Problem Solving from Nature (PPSN '04)*, Lecture Notes in Computer Science, pp. 942–951, Birmingham, UK, September 2004.

[127] L. Bull, "A simple payoff-based learning classifier system," in *Proceedings of the 8th International Conference on Parallel Problem Solving from Nature*, Lecture Notes in Computer Science, pp. 1032–1041, 2004.

[128] J. Bacardit, *Pittsburgh genetic-based machine learning in the data mining era: representations, generalization, and run-time*, Ph.D. thesis, Enginyeria i Arquitectura La Salle, Ramon Llull University, Barcelona, European Union, Catalonia, Spain, 2004.

[129] J. Bacardit, "Analysis of the initialization stage of a Pittsburgh approach learning classifier system," in *Proceedings of the Conference on Genetic and Evolutionary Computation*, pp. 1843–1850, 2005.

[130] J. Bacardit and M. Butz, "Data mining in learning classifier systems: comparing XCS with GAssist," in *Proceedings of the International Workshop on Learning Classifier Systems (IWLCS '07)*, vol. 4399 of *Lecture Notes in Computer Science*, pp. 282–290, 2007.

[131] M. Stout, J. Bacardit, J. D. Hirst, R. E. Smith, and N. Krasnogor, "Prediction of topological contacts in proteins using learning classifier systems," *Soft Computing*, vol. 13, no. 3, pp. 245–258, 2009.

[132] P. Gérard, J.-A. Meyer, and O. Sigaud, "Combining latent learning with dynamic programming in the modular anticipatory classifier system," *European Journal of Operational Research*, vol. 160, no. 3, pp. 614–637, 2005.

[133] A. Hamzeh and A. Rahmani, "An evolutionary function approximation approach to compute prediction in XCSF," in *Proceedings of the 16th European Conference on Machine Learning (ECML '05)*, vol. 3720 of *Lecture Notes in Computer Science*, pp. 584–592, Porto, Portugal, October 2005.

[134] S. Landau, O. Sigaud, and M. Schoenauer, "ATNoSFERES revisited," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '05)*, pp. 1867–1874, 2005.

[135] O. Unold, "Context-free grammar induction with grammar-based classifier system," *Archives of Control Science*, vol. 15, no. 4, p. 681, 2005.

[136] O. Unold and L. Cielecki, "Grammar-based classifier system," in *Issues in Intelligent Systems: Paradigms*, pp. 273–286, EXIT, Warsaw, Poland, 2005.

[137] H. Dam, H. Abbass, and C. Lokan, "DXCS: an XCS system for distributed data mining," in *Proceedings of the Conference on Genetic and Evolutionary Computation*, pp. 1883–1890, 2005.

[138] H. Dam, H. Abbass, and C. Lokan, "Investigation on DXCS: an XCS system for distribution data mining, with continuous-valued inputs in static and dynamic environments," in *Proceedings of the IEEE Cogress on Evolutionary Computation*, 2005.

[139] H. Dam, H. Abbass, and C. Lokan, "The performance of the DXCS system on continuous-valued inputs in stationary and dynamic environments," in *Proceedings of the IEEE Congress on Evolutionary Computation*, vol. 1, 2005.

[140] Y. Gao, J. Huang, H. Rong, and D. Gu, "Learning classifier system ensemble for data mining," in *Proceedings of the Workshops on Genetic and Evolutionary Computation*, pp. 63–66, 2005.

[141] Y. Gao, L. Wu, and J. Huang, "Ensemble learning classifier system and compact ruleset," in *Proceedings of the 6th International Conference Simulated Evolution and Learning (SEAL '06)*, vol. 4247 of *Lecture Notes in Computer Science*, pp. 42–49, Hefei, China, October 2006.

[142] Y. Gao, J. Z. Huang, H. Rong, and D.-Q. Gu, "LCSE: learning classifier system ensemble for incremental medical instances," in *Proceedings of the International Workshop on Learning Classifier Systems (IWLCS '07)*, vol. 4399 of *Lecture Notes in Computer Science*, pp. 93–103, 2007.

[143] J. H. Holmes and J. A. Sager, "Rule discovery in epidemiologic surveillance data using EpiXCS: an evolutionary computation approach," in *Proceedings of the 10th Conference on Artificial Intelligence in Medicine (AIME '05)*, vol. 3581 of *Lecture Notes in Computer Science*, pp. 444–452, Aberdeen, Scotland, July 2005.

[144] J. H. Holmes and J. A. Sager, "The EpiXCS workbench: a tool for experimentation and visualization," in *Proceedings of the International Workshop on Learning Classifier Systems (IWLCS '07)*, vol. 4399 of *Lecture Notes in Computer Science*, pp. 333–344, 2007.

[145] J. H. Holmes, "Detection of sentinel predictor-class associations with XCS: a sensitivity analysis," in *Proceedings of the International Workshop on Learning Classifier Systems (IWLCS '07)*, vol. 4399 of *Lecture Notes in Computer Science*, pp. 270–281, 2007.

[146] D. Loiacono and P. Lanzi, "Evolving neural networks for classifier prediction with XCSF," in *Proceedings of the Workshop on Evolutionary Computation (ECAI '06)*, pp. 36–40, 2006.

[147] H. Dam, H. Abbass, and C. Lokan, "BCS: a Bayesian learning classifier system," Tech. Rep. TR-ALAR-200604005, The Artificial Life and Adaptic Robotics Laboratory, School of Information Technology and Electrical Engineering, University of New South Wales, Cardiff, UK, 2006.

[148] J. Bacardit and N. Krasnogor, "Biohel: bioinformatics-oriented hierarchical evolutionary learning (Nottingham ePrints)," Tech. Rep., University of Nottingham, Nottingham, UK, 2006.

[149] J. Bacardit, M. Stout, J. D. Hirst, K. Sastry, X. Llorà, and N. Krasnogor, "Automated alphabet reduction method with evolutionary algorithms for protein structure prediction," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '07)*, pp. 346–353, ACM Press, New York, NY, USA, 2007.

[150] A. Hamzeh and A. Rahmani, "Extending XCSFG beyond linear approximation," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '06)*, pp. 2246–2253, Vancouver, Canada, July 2006.

[151] A. Hamzeh and A. Rahmani, "A new architecture of XCS to approximate real-valued functions based on high order polynomials using variable-length GA," in *Proceedings of the 3rd International Conference on Natural Computation (ICNC '07)*, vol. 3, pp. 515–519, Haikou, China, August 2007.

[152] P. Lanzi and D. Loiacono, "Classifier systems that compute action mappings," in *Proceedings of the 9th Genetic and Evolutionary Computation Conference (GECCO '07)*, pp. 1822–1829, ACM Press, New York, NY, USA, 2007.

[153] M. Gershoff and S. Schulenburg, "Collective behavior based hierarchical XCS," in *Proceedings of the Conference on Genetic and Evolutionary Computation Conference (GECCO '07)*, pp. 2695–2700, ACM Press, New York, NY, USA, 2007.

[154] R. E. Smith and M. K. Jiang, "MILCS: a mutual information learning classifier system," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '07)*, pp. 2945–2952, ACM Press, New York, NY, USA, 2007.

[155] L. Cielecki and O. Unold, "GCS with real-valued input," in *Proceedings of the 2nd International Work-Conference on The Interplay between Natural and Artificial Computation*, vol. 4527 of *Lecture Notes in Computer Science*, pp. 488–497, 2007.

[156] J. Casillas and L. Bull, "Fuzzy-XCS: a Michigan genetic fuzzy system," *IEEE Transactions on Fuzzy Systems*, vol. 15, no. 4, pp. 536–550, 2007.

[157] A. Orriols-Puig, J. Casillas, and E. Bernadó-Mansilla, "Fuzzy-UCS: preliminary results," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '07)*, pp. 2871–2874, 2007.

[158] X. Llorà, R. Reddy, B. Matesic, and R. Bhargava, "Towards better than human capability in diagnosing prostate cancer using infrared spectroscopic imaging," in *Proceedings of the 9th Genetic and Evolutionary Computation Conference (GECCO '07)*, pp. 2098–2105, ACM Press, New York, NY, USA, 2007.

[159] R. S. Sutton, "Introduction: the challenge of reinforcement learning," *Machine Learning*, vol. 8, no. 3-4, pp. 225–227, 1992.

[160] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, no. 1, pp. 9–44, 1988.

[161] C. Watkins, Learning from Delayed Rewards, 1989.

[162] G. Liepins, M. Hilliard, M. Palmer, and G. Rangarajan, "Alternatives for classifier system credit assignment," in *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI '89)*, pp. 756–761, 1989.

[163] M. Dorigo and H. Bersini, "A comparison of Q-learning and classifier systems," in *Proceedings of the 3rd International Conference on Simulation of Adaptive Behavior: From Animals to Animats 3*, pp. 248–255, MIT Press, Cambridge, Mass, USA, 1994.

[164] M. Dorigo, "Genetic and non-genetic operators in ALECSYS," *Evolutionary Computation*, vol. 1, no. 2, pp. 151–164, 1993.

[165] P. Lanzi and R. Riolo, "A roadmap to the last decade of learning classifier system research(from 1989 to 1999)," in *Proceedings of the International Workshop on Learning Classifier Systems*, Lecture Notes in Computer Science, pp. 33–61, 2000.

[166] P. W. Frey and D. J. Slate, "Letter recognition using Holland-style adaptive classifiers," *Machine Learning*, vol. 6, no. 2, pp. 161–182, 1991.

[167] T. Kovacs, *A comparison of strength and accuracy-based fitness in learning classifier systems*, Ph.D. thesis, University of Birmingham, Birmingham, UK, 2002.

[168] P. Lanzi, "Learning classifier systems from a reinforcement learning perspective," *Soft Computing*, vol. 6, no. 3, pp. 162–170, 2002.

[169] M. Butz, D. Goldberg, and W. Stolzmann, "Introducing a genetic generalization pressure to the anticipatory classier system: part 1-theoretical approach," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '00)*, 2000.

[170] M. Butz, D. Goldberg, and W. Stolzmann, "Investigating generalization in the anticipatory classifier system," in *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*, Lecture Notes in Computer Science, pp. 735–744, 2000.

[171] M. Butz, D. Goldberg, and W. Stolzmann, "Probability-enhanced predictions in the anticipatory classifier system," in *Proceedings of the International Workshop on Learning Classifier Systems (IWLCS '00)*, pp. 37–51, Springer, 2000.

[172] M. Butz, *Anticipatory Learning Classifier Systems*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.

[173] J. Bacardit and M. Butz, "Data mining in learning classifier systems: comparing XCS with GAssist," in *Proceedings of the 7th International Workshop on Learning Classifier Systems (IWLCS '04)*, 2004.

[174] T. Kovacs and P. Lanzi, "A learning classifier systems bibliography," Tech. Rep., CSR Centre, School of Computer Science Research, University of Birmingham, Birmingham, UK, 1999.

[175] P. Stalph and M. Butz, Documentation of XCSF-Ellipsoids Java plus Visualization.

[176] M. Butz, "Documentation of XCSFJava 1.1 plus visualization," MEDAL Report 2007008, 2007.

[177] M. Butz and O. Herbort, "Context-dependent predictions and cognitive arm control with XCSF," in *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, pp. 1357–1364, ACM, New York, NY, USA, 2008.

[178] M. Butz, "Combining gradient-based with evolutionary online learning: an introduction to learning classifier systems," in *Proceedings of the 7th International Conference on Hybrid Intelligent Systems (HIS '07)*, pp. 12–17, 2007.

[179] S. Russell, P. Norvig, J. Canny, J. Malik, and D. Edwards, *Artificial Intelligence: A Modern Approach*, Prentice-Hall, Englewood Cliffs, NJ, USA, 1995.

[180] M. Butz, *Rule-based Evolutionary Online Learning Systems: A Principled Approach to LCS Analysis And Design*, Springer, Berlin, Germany, 2006.

[181] J. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, Mass, USA, 1992.

[182] M. Dorigo and T. Stützle, *Ant Colony Optimization*, MIT Press, Cambridge, Mass, USA, 2004.

[183] L. De Castro and J. Timmis, *Artificial Immune Systems: A New Computational Intelligence Approach*, Springer, New York, NY, USA, 2002.

[184] S. Haykin, *Neural Networks: A Comprehensive Foundation*, Prentice-Hall, Upper Saddle River, NJ, USA, 1998.

[185] E. Bernado, X. Llorà, and J. Garrell, "XCS and GALE: a comparative study of two learning classifier systems with six other learning algorithms on classification tasks," in *Proceedings of the 4th International Workshop on Learning Classifier Systems (IWLCS '01)*, pp. 337–341, 2001.

[186] F. Kharbat, L. Bull, and M. Odeh, "Mining breast cancer data with XCS," in *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation Conference (GECCO '07)*, pp. 2066–2073, 2007.

[187] O. Unold and K. Tuszyński, "Mining knowledge from data using anticipatory classifier system," *Knowledge-Based Systems*, vol. 21, no. 5, pp. 363–370, 2008.

[188] C. Blake and C. Merz, "UCI repository of machine learning databases," 1998.

[189] S. Alayón, J. I. Estévez, J. Sigut, J. L. Sánchez, and P. Toledo, "An evolutionary Michigan recurrent fuzzy system for nuclei classification in cytological images using nuclear chromatin distribution," *Journal of Biomedical Informatics*, vol. 39, no. 6, pp. 573–588, 2006.

[190] O. Unold, "Grammar-based classifier system for recognition of promoter regions," in *Proceedings of the 8th International Conference on Adaptive and Natural Computing Algorithms (ICANNGA '07)*, vol. 4431 of *Lecture Notes in Computer Science*, pp. 798–805, 2007.

[191] T. Kovacs, *Evolving optimal populations with XCS classier systems*, M.S. thesis, School of Computer Science, University of Birmingham, Birmingham, UK, 1996.

[192] T. Kovacs, "What should a classifier system learn?" in *Proceedings of the Congress on Evolutionary Computation*, vol. 2, 2001.

[193] T. Kovacs, "What should a classifier system learn and how should we measure it?" *Soft Computing-A*, vol. 6, no. 3, pp. 171–182, 2002.

[194] M. Butz and M. Pelikan, "Analyzing the evolutionary pressures in XCS," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '01)*, pp. 935–942, 2001.

[195] M. V. Butz, T. Kovacs, P. L. Lanzi, and S. W. Wilson, "Toward a theory of generalization and learning in XCS," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 1, pp. 28–46, 2004.

[196] J. Bassett and K. De Jong, "Evolving behaviors for cooperating agents," in *Proceedings of the 12th International Symposium on Foundations of Intelligent Systems (ISMIS '00)*, vol. 1932 of *Lecture Notes in Computer Science*, pp. 157–165, 2000.

[197] M. Butz and D. Goldberg, "Bounding the population size in XCS to ensure reproductive opportunities," in *Proceedings of the Conference on Genetic and Evolutionary Computation*, Lecture Notes in Computer Science, pp. 1844–1856, 2003.

[198] M. Butz, D. Goldberg, P. Lanzi, and K. Sastry, "Bounding the population size to ensure niche support in XCS," IlliGAl Report 2004033, July 2004.

[199] M. Butz, D. Goldberg, and P. Lanzi, "Bounding learning time in XCS," in *Proceedings of Genetic and Evolutionary Computation Conference (GECCO '04)*, pp. 739–750, Seattle,Wash, USA, June 2004.

[200] M. Butz, *Rule-based evolutionary online learning systems: learning bounds, classification, and prediction*, Ph.D. thesis, 2004.

[201] L. Bull, "Learning classifier systems: a brief introduction," Applications of Learning Classifier Systems, 2004.

[202] L. Booker, "Representing attribute-based concepts in a classifier system," *Foundations of Genetic Algorithms*, pp. 115–127, 1991.

[203] S. Sen, "A tale of two representations," in *Proceedings of the 7th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pp. 245–254, Gordon and Breach, 1994.

[204] R. Riolo, "The emergence of coupled sequences of classifiers," in *Proceedings of the 3rd International Conference on Genetic Algorithms and Their Application*, pp. 256–264, Morgan Kaufmann, San Francisco, Calif, USA, 1989.

[205] D. Schuurmans and J. Schaeffer, *Representational Difficulties with Classifier Systems*, Department of Computing Science, University of Alberta, Edmonton, Canada, 1988.

[206] C. Stone and L. Bull, "For real! XCS with continuous-valued inputs," *Evolutionary Computation*, vol. 11, no. 3, pp. 299–336, 2003.

[207] H. Dam, H. Abbass, and C. Lokan, "Be real! XCS with continuous-valued inputs," in *Proceedings of the Workshops on Genetic and Evolutionary Computation*, pp. 85–87, ACM, New York, NY, USA, 2005.

[208] P. Lanzi and S. Wilson, "Using convex hulls to represent classifier conditions," in *Proceedings of the 8th Genetic and Evolutionary Computation Conference (GECCO '06)*, vol. 2, pp. 1481–1488, ACM Press, New York, NY, USA, 2006.

[209] M. Butz, "Kernel-based, ellipsoidal conditions in the real-valued XCS classifier system," in *Proceedings of the Conference on Genetic and Evolutionary Computation*, pp. 1835–1842, 2005.

[210] M. Butz, P. Lanzi, and S. Wilson, "Hyper-ellipsoidal conditions in XCS: rotation, linear approximation, and solution structure," in *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, pp. 1457–1464, ACM, New York, NY, USA, 2006.

[211] L. Booker, "Improving the performance of genetic algorithms in classifier systems," in *Proceedings of the 1st International Conference on Genetic Algorithms*, pp. 80–92, Lawrence Erlbaum Associates, Mahwah, NJ, USA, 1985.

[212] D. Mellor, "A first order logic classifier system," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '05)*, pp. 1819–1826, ACM Press, New York, NY, USA, 2005.

[213] P. 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)*, vol. 1, pp. 337–344, 1999.

[214] P. Tufts, "Dynamic classifiers: genetic programming and classifier systems," in *Proceedings of the Genetic Programming*, pp. 114–119, 1995.

[215] M. Ahluwalia, L. Bull, W. Banzhaf, et al., "A genetic programming-based classifier system," in *Proceedings of the Genetic and Evolutionary Computation Conference*, vol. 1, pp. 11–18, 1999.

[216] P. Lanzi and A. Perrucci, "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)*, vol. 1, pp. 345–352, 1999.

[217] P. Lanzi, "Mining interesting knowledge from data with the XCS classier system," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '01)*, pp. 958–965, 2001.

[218] P. Lanzi, "An analysis of generalization in XCS with symbolic conditions," in *Proceedings of IEEE Congress on Evolutionary Computation (CEC '07)*, pp. 2149–2156, 2007.

[219] L. Bull and J. Hurst, "A neural learning classifier system with self-adaptive constructivism," in *Proceedings of the Congress on Evolutionary Computation (CEC '03)*, vol. 2, 2003.

[220] T. O'Hara and L. Bull, "A memetic accuracy-based neural learning classifier system," in *Proceedings of IEEE Congress on Evolutionary Computation (CEC '05)*, vol. 3, pp. 2040–2045, 2005.

[221] B. Carse and T. Fogarty, "A fuzzy classifier system using the Pittsburgh approach," in *Proceedings of the International Conference on Evolutionary Computation, the 3rd Conference on Parallel Problem Solving from Nature*, Jerusalem, Israel, October 1994.

[222] M. Butz, K. Sastry, and D. Goldberg, "Tournament selection in XCS," in *Proceedings of the 5th Genetic and Evolutionary Computation Conference (GECCO '02)*, vol. 1869, 2002.

[223] M. V. Butz, D. E. Goldberg, and K. Tharakunnel, "Analysis and improvement of fitness exploitation in XCS: bounding models, tournament selection, and bilateral accuracy," *Evolutionary Computation*, vol. 11, no. 3, pp. 239–277, 2003.

[224] F. Kharbat, L. Bull, and M. Odeh, "Revisiting genetic selection in the XCS learning classifier system," in *Proceedings of the IEEE Congress on Evolutionary Computation*, vol. 3, 2005.

[225] M. V. Butz, K. Sastry, and D. E. Goldberg, "Strong, stable, and reliable fitness pressure in XCS due to tournament selection," *Genetic Programming and Evolvable Machines*, vol. 6, no. 1, pp. 53–77, 2005.

[226] B. Widrow and M. E. Hoff, "Adaptive switching circuits," in *IRE WESCON Convention Record*, vol. 4, pp. 709–717, 1960.

[227] G. Venturini, *Apprentissage adaptatif et apprentissage supervise par algorithme genetique*, These de Docteur en Science, Universite de Paris-Sud, Paris, France, 1994.

[228] M. Butz, T. Kovacs, P. Lanzi, and S. Wilson, "How XCS evolves accurate classifiers," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '01)*, pp. 927–934, 2001.

[229] G. Liepins and L. Wang, "Classifier system learning of Boolean concepts," in *Proceedings of the 4th International Conference on Genetic Algorithms*, pp. 318–323, Morgan Kaufmann, San Francisco, Calif, USA, 1991.

[230] G. Weiss, *The Action oriented Bucket Brigade*, Institut für Informatik, 1991.

[231] G. Weiss, "Ah action-oriented perspective of learning in classifier systems," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 8, no. 1, pp. 43–62, 1996.

[232] M. V. Butz, D. E. Goldberg, and P. L. Lanzi, "Gradient descent methods in learning classifier systems: improving XCS performance in multistep problems," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 5, pp. 452–473, 2005.

[233] P. Lanzi, M. V. Butz, and D. E. Goldberg, "Empirical analysis of generalization and learning in XCS with gradient descent," in *Proceedings of of the 9th Genetic and Evolutionary Computation Conference (GECCO '07)*, pp. 1814–1821, ACM Press, New York, NY, USA, 2007.

[234] J. Drugowitsch and A. M. Barry, "XCS with eligibility traces," in *Proceedings of the Conference on Genetic and Evolutionary Computation Conference (GECCO '05)*, pp. 1851–1858, ACM, New York, NY, USA, 2005.

[235] P. Lanzi, D. Loiacono, S. W. Wilson, and D. E. Goldberg, "Prediction update algorithms for XCSF: RLS, Kalman filter, and gain adaptation," in *Proceedings of the 8th Genetic and Evolutionary Computation Conference (GECCO '06)*, vol. 2, pp. 1505–1512, ACM Press, 2006.

[236] J. Horn, D. Goldberg, and K. Deb, "Implicit niching in a learning classifier system: nature's way," *Evolutionary Computation*, vol. 2, no. 1, pp. 37–66, 1994.

[237] J. Horn, D. Goldberg, J. Koza, D. Goldberg, D. Fogel, and R. Riolo, "Natural niching for cooperative learning in classifier systems," in *Proceedings of the 1st Annual Conference on Genetic Programming*, pp. 553–564, MIT Press, 1996.

[238] M. V. Butz, M. Pelikan, X. Llorà, and D. E. Goldberg, "Extracted global structure makes local building block processing effective in XCS," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '05)*, pp. 655–662, ACM, New York, NY, USA, 2005.

[239] M. V. Butz, M. Pelikan, X. Llorà, and D. E. Goldberg, "Automated global structure extraction for effective local building block processing in XCS," *Evolutionary Computation*, vol. 14, no. 3, pp. 345–380, 2006.

[240] J. Bacardit and N. Krasnogor, "Smart crossover operator with multiple parents for a pittsburgh learning classifier system," in *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (GECCO '06)*, vol. 2, pp. 1441–1448, ACM Press, New York, NY, USA, 2006.

[241] F. Serendynski, P. Cichosz, and G. Klebus, "Learning classifier systems in multi-agent environments," in *Proceedings of the 1st International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications (GAESIA '95)*, pp. 287–292, 1995.

[242] S. Sen and M. Sekaran, "Multiagent coordination with learning classifier systems," in *Proceedings of the Adaption and Learning in Multi-Agent Systems*, Lecture Notes in Computer Science, pp. 218–233, 1996.

[243] L. Bull, M. Studley, T. Bagnall, and I. Whittley, "On the use of rule-sharing in learning classifier system ensembles," in *Proceedings of the Congress on Evolutionary Computation (CEC '05)*, vol. 1, 2005.

[244] L. Bull, M. Studley, A. Bagnall, and I. Whittley, "Learning classifier system ensembles with rule-sharing," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 4, pp. 496–502, 2007.

[245] J. Bacardit and N. Krasnogor, "Empirical evaluation of ensemble techniques for a Pittsburgh learning classifier

system," in *Proceedings of the 9th International Workshop on Learning Classifier Systems (IWLCS '08)*, vol. 4998, pp. 255–268, 2008.

[246] H. H. Dam, P. Rojanavasu, H. A. Abbass, and C. Lokan, "Distributed learning classifier systems," *Studies in Computational Intelligence*, vol. 125, pp. 69–91, 2008.

[247] C. Lokan, "Distributed learning classifier systems," in *Learning Classifier Systems in Data Mining*, 2008.

[248] R. Ranawana and V. Palade, "Multi-classifier systems: review and a roadmap for developers," *International Journal of Hybrid Intelligent Systems*, vol. 3, no. 1, pp. 35–61, 2006.

[249] E. Bernado-Mansilla, X. Llorà, and I. Traus, "Multiobjective learning classifier systems: an overview," Tech. Rep., University of Illinois at Urbana Champaign, Urbana, Ill, USA, 2005.

[250] C. Fu and L. Davis, "A modified classifier system compaction algorithm," in *Proceedings of the Conference on Genetic and Evolutionary Computation Conference (GECCO '02)*, pp. 920–925, 2002.

[251] M. V. Butz, P. L. Lanzi, and S. W. Wilson, "Function approximation with XCS: hyperellipsoidal conditions, recursive least squares, and compaction," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 3, pp. 355–376, 2008.

[252] J. Holmes, J. Sager, and W. Bilker, "A comparison of three methods for covering missing data in XCS," in *Proceedings of the 7th International Workshop on Learning Classifier Systems (IWLCS '04)*, Seattle, Wash, USA, June 2004.

[253] A. Orriols-Puig and E. Bernadó-Mansilla, "Bounding XCS's parameters for unbalanced datasets," in *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation Conference (GECCO '06)*, vol. 2, pp. 1561–1568, ACM, New York, NY, USA, 2006.

[254] H. Dam, K. Shafi, and H. Abbass, "Can evolutionary computation handle large dataset?" Tech. Rep. TR-ALAR-2005070001, 2005, http://seal.itee.adfa.edu.au/~alar/techreps.

[255] L. Booker, "Classier systems, endogenous fitness, and delayed rewards: a preliminary investigation," in *Proceedings of the International Workshop on Learning Classifier Systems (IWLCS '00) in the Joint Workshops of SAB*, 2000.

[256] J. Hurst and L. Bull, "A self-adaptive classifier system," in *Proceedings of the 3rd International Workshop on Advances in Learning Classifier Systems*, pp. 70–79, Springer, 2000.

[257] L. Bull and J. Hurst, "Self-adaptive mutation in ZCS controllers," in *Proceedings of the Real-World Applications of Evolutionary Computing, EvoWorkshops*, Lecture Notes in Computer Science, pp. 339–346, 2000.

[258] L. Bull, J. Hurst, and A. Tomlinson, "Self-adaptive mutation in classifier system controllers," in *From Animals to Animats 6: Proceedings of the 6th International Conference on Simulation of Adaptive Behavior*, MIT Press, 2000.

[259] J. Hurst and L. Bull, "A self-adaptive XCS," in *Proceedings of the 4th International Workshop on Advances in Learning Classifier Systems*, Lecture Notes in Computer Science, pp. 57–73, 2002.

[260] W. Browne, "Improving Evolutionary Computation Based Data-Mining for the Process Industry: The Importance of Abstraction," Learning Classifier Systems in Data Mining, 2008.

[261] D. Goldberg, J. Horn, and K. Deb, "What makes a problem hard for a classifier system?" Tech. Rep., Santa Fe Working Paper, 1992.

[262] L. B. Booker, D. E. Goldberg, and J. Holland, "Classifier systems and genetic algorithms," in *Machine Learning: Paradigms and Methods*, pp. 235–282, 1989.

[263] J. Holland, L. Booker, M. Colombetti, et al., "What is a learning classifier system?" in *Learning Classifier Systems, from Foundations to Applications*, Lecture Notes in Computer Science, pp. 3–32, 2000.

[264] S. W. Wilson, "State of XCS classifier system research," in *Proceedings of the 3rd International Workshop on Advances in Learning Classifier Systems*, Lecture Notes in Computer Science, pp. 63–82, 2000.

[265] T. Kovacs, "Learning classifier systems resources," *Soft Computing-A*, vol. 6, no. 3, pp. 240–243, 2002.

[266] J. Drugowitsch, *Design and Analysis of Learning Classifier Systems: A Probabilistic Approach*, Springer, Berlin, Germany, 2008.