A Brief Introduction to GMBL Systems and its Divisions Implementations

Shokoofeh Hoseini1, MohammadAli MehdiAghaei2

1Kharazmi University, Faculty of Management,

2Science and Research Branch Islamic Azad University, Faculty Of Management

hoseini.shokoofeh@gmail.com, shabar.aghaei@gmail.com

***Abstract*:** In this paper we introduce a brief history of Genetic Base Machine Learning as Evolutionary Computing paradigm and lecture some features of each implementations under two points of view, Michigan-style LCS and Pittsburgh-style LCS and distinguish each divisions among implementations. Keeping in mind that LCS is a paradigm for genetic-based machine learning rather than a specific method. Finally we introduce KEEL as tools of GBML systems.

***Keywords*:** learning classifier systems, genetic based machine learning, Michigan approach, Pittsburgh approach.

# Introduction

[Learning] Classifier Systems are a kind of rule-based system with general mechanisms for processing rules in parallel, for adaptive generation of new rules, and for testing the effectiveness of existing rules. These mechanisms make possible performance and learning without the “brittleness” characteristic of most expert systems in AI [[1](#Ref1)].

This rule-based machine learning methods combine a discovery component with a learning component. Learning classifier systems seek to identify a set of context-dependent rules that collectively store and apply knowledge in piecewise manner in order to make predictions. This approach allows complex solution spaces to be broken up into smaller, simpler parts. The application of Genetic Algorithms (GA) to classification problems is usually known as Genetic Based Machine Learning (GBML) or Learning Classifier Systems (LCS), and traditionally it has been addressed from two different points of view: the Pittsburgh approach and the Michigan approach.

Section 2 introduce Evolutionary computations and its paradigm. Evolutionary learning is the application of evolutionary computation (EC) to learning tasks. Evolutionary computation is a field that gathers a large collection of techniques inspired in biological processes such as population-based evolution, natural selection and genetics. These techniques can be applied to several kind of tasks: search, optimization, scheduling, and, of course, machine learning*.* Section 3 introduce the divisions among LCS implementations. Section 4 introduce KEEL as a GBML tool which have implementation of those Michigan and Pittsburgh approach.

In order to understand this paper you have to know about LCS elements and LCS machine learning (see [[2](#Ref2)]). Because it is not the purpose of this paper, we just introduce a brief divisions implementations of LCSs and some famous implementation and a tool in which these implementation has been founded. We also don’t discuss strengthens and weaknesses of algorithms.

# Evolutionary Computing and its paradigm

Evolutionary computation (EC) techniques are optimization tools that solve problems using procedures inspired by natural processes. These techniques usually work by transforming a population of individuals, being each individual a candidate solution for our problem. This  
transformation process consists in the iterative application of a cycle of stages inspired in natural selection and also in the generation of new individuals by genetic recombination. This  
combination of selection and recombination produces a directed exploration of the search space, converging to the regions of the space where the best solutions are placed. Research in evolutionary computation started in the 1960’s, and the first major milestone was John Holland’s book, considered a foundation work in the field (Holland, 1975).

Using a classical classification, we can describe four main EC paradigms (Freitas, 2002) [[3](#Ref3)]:

**Evolution Strategies (ES)** (Rechenberg, 1973). These techniques typically use an individual representation consisting of a real-valued vector. Early ES emphasized mutation as the main exploratory search operator, but currently both mutation and crossover are used.  
An individual often represents not only real-valued variables of the problem being solved but also parameters controlling the mutation distribution, characterizing a self-adaptation  
of mutation parameters. The mutation operator usually modifies individuals according to a multivariate normal distribution, where small mutations are more likely than large mutations [[4](#Ref4)].

**Evolutionary Programming (EP)** (Fogel, 1964). Originally developed to evolve finite-state machines, but it is now often used to evolve individuals consisting of a real-valued vector.  
Unlike ES, in general it does not use crossover. Similar to ES, it also uses normally distributed mutations and self-adaptation of mutation parameters [[5](#Ref5)].

**Genetic Algorithms (GA)** (Holland, 1975; Goldberg, 1989a). This is the most popular paradigm of EC. GAs emphasize crossover as the main exploratory search operator and consider mutation as a minor operator, typically applied with a very low probability. In early (“classic”) GAs individuals were represented by binary strings, but nowadays more elaborate representations, such as real-valued strings, are also used [[6](#Ref6)].

**Genetic Programming (GP)** (Koza, 1992). This paradigm is often described as a variation of GAs rather than a mainstream EC paradigm in an of itself. Individuals being evolved in this paradigm are various kinds of computer programs, consisting not only of data structures but also of functions (or operations) applied to those data structures. These programs are usually represented using trees [[7](#Ref7)].

In recent years a new paradigm has been developed, which could be added to the previous list. It is known as *estimation of distribution algorithms (EDAs)* (Larranaga & Lozano, 2002). The main difference from the above stated paradigms are the recombination operators used: A statistical model is created from the individuals of the population, and the offspring are generated by sampling this model. Thus, the exploration process is less *blind* than the one used in the other EC paradigms. GBML systems use GA as Discovery components rather than any other discovery methods. (EDAs) [[8](#Ref8)].

# GBML Systems and its Implementations

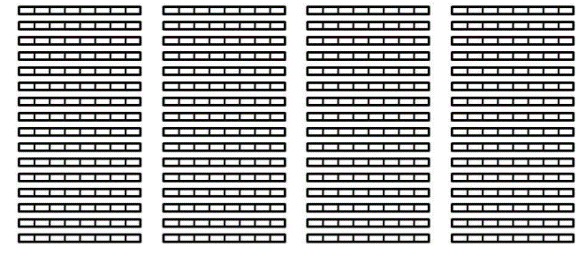
As stated previously in the introduction of this paper, usually only two models are described in the literature, called *Michigan approach* and *Pittsburgh approach*. However, in recent years a third approach, called *Iterative Rule Learning*, has risen in popularity. This third approach, first used in the *SIA* system (Venturini, 1993) [[9](#Ref9)] uses the separate-and-conquer methodology quite popular in the non-evolutionary rule induction field. In this approach an individual is a rule, like in *Michigan*, but the solution provided by the *GA* is the best individual of the population, like in *Pitt*, although the final solution is the concatenation of the rules obtained by running the *GA* several time.

**Pittsburgh approach**, which represents an extension of the traditional fixed-length chromosome approaches. Here, variable-length chromosomes are used to represent proposed solutions individually. This approach (LS for Learning System) was originally proposed by Smith (1980). This representation seems more naturally suited for the supervised learning, since each chromosome represents an independent solution [[10](#Ref10)].



**Figure 1.** Pittsburgh-Style LCS

**Michigan approach**, where the population still consists of fixed-length elements, but the solution is represented by a set of chromosomes from the population. This methodology, known as CS for classifier systems, along with a special "bucket brigade" mechanism for credit assignment, was originally developed by Holland and colleagues. Here, each chromosome, called a classifier, represents a structure composed of conditions and messages lists. The environment, together with the activated rules, provides a set of active messages, Theses, in turn activate other classifiers by satisfying their conditions. The chained actions of message-condition pairs cluster the rules together. Because of this chaining mechanism, this approach seems more suitable for planning than concept learning. In this approach the GA is not the central element, but only a part of the system used from time to time to discover new rules [[10](#Ref10)].



**Figure 2.** Michigan-Style LCS

**The Iterative Rule Learning**, first used in the *SIA* system (Venturini, 1993) uses the separate-and-conquer methodology to induce rules, using a *GA* to generate each rule. In this approach an individual is a rule, like in *Michigan*, but the solution provided by the *GA* is the best individual of the population, like in *Pitt*, although the final solution is the concatenation of the rules obtained by running the *GA* several time. This approach has been used extensively in genetic-fuzzy systems [[11](#Ref11)] but there are also some examples of application of this model to crisp presentations,   
like the *HIDER* system (Aguilar-Ruiz, Riquelme, & Toro, 2003).

The architecture and components of a given learning classifier system can be quite variable. It is useful to think of an LCS as a machine consisting of several interacting components. Components may be added or removed, or existing components modified/exchanged to suit the demands of a given problem domain or to make the algorithm flexible enough to function in many different problem domains. As a result, the LCS paradigm can be flexibly applied to many problem domains that call for machine learning [[2](#Ref2)]. The major divisions among LCS implementations are as follows: (1) Michigan-style architecture vs. Pittsburgh-style architecture, (2) reinforcement learning vs. supervised learning, (3) incremental learning vs. batch learning, (4) online learning vs. offline learning, (5) strength-based fitness vs. accuracy-based fitness, and (6) complete action mapping vs best action mapping. These divisions are not necessarily mutually exclusive. Let’s take a look at brief definition of these divisions categorizations.

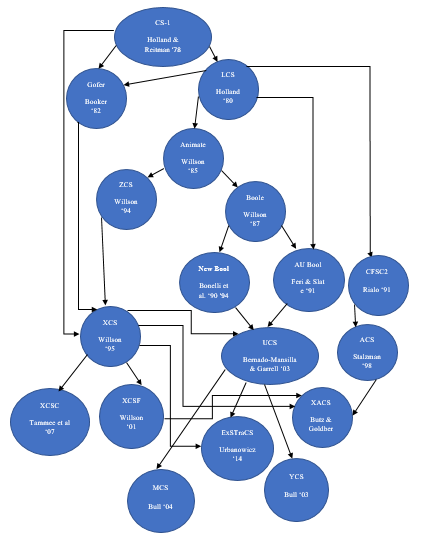
*Supervised learning* as the name indicates a presence of supervisor as teacher. Basically supervised learning is a learning in which we teach or train the machine using data which is well labeled that means some data is already tagged with correct answer. After that, machine is provided with new set of examples (data) so that supervised learning algorithm analyses the training data (set of training examples) and produces a correct outcome from labeled data [[12](#Ref12)].

*Unsupervised learning* is the training of machine using information that is neither classified nor labeled and allowing the algorithm to act on that information without guidance. Here the task of machine is to group unsorted information according to similarities, patterns and differences without any prior training of data. Unlike supervised learning, no teacher is provided that means no training will be given to the machine. Therefore machine is restricted to find the hidden structure in unlabeled data by our-self [[2](#Ref2)].

*Reinforcement learning* differs from the supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task. In the absence of training dataset, it is bound to learn from its experience [[13](#Ref13)]. *Incremental learning* meaning that machine gets one instance from environment at the time, the environment interacts the learning machine, passes one instance, completing the learning cycle and returning it to the environment and grabs the next instance. *Batch learning* is to train your classifier/ algorithm once using this complete training set of data [[2](#Ref2)]. In *online learning*, only the set of possible elements is known, a streaming set of data set in presented .In *offline learning*, the identity of the elements as well as the order in which they are presented is known to the learner which means you have a static data set [[14](#Ref14)].

LCS have traditionally been *strength-based*, meaning that the strength of a rule is also used as its fitness in the GA. In this method we calculate the strength of rules and treats them as predictions of the reward the system will receive if they are used. However, some implementations differs from these systems in its use of *accuracy-based* fitness, in which the fitness of a rule is based on the accuracy with which it predicts the reward the system will receive if it is used. We could say that this bases fitness on the consistency of a rule’s strength over time, as rules with consistent strengths have converged to predict a particular value, and so make accurate predictions of it. Rules with persistently varying strength are being successively updated towards different rewards (because they are over general), and so make inaccurate predictions of them [[15](#Ref15)].

A *complete action map* which predicts the rewards of all actions in each state and a *best action map* which in contrast covers only the highest-return action in each state. Conceptually, complete maps don't only capture what you should do, or what is correct, but also what you shouldn't do, or what's incorrect. Differently, most strength-based LCSs, or exclusively supervised learning LCSs seek a rule set of efficient generalizations in the form of a *best action map* or a *partial map* [[16](#Ref16)].



**Fig 3.** Michigan-Style Learning classifier system’s family tree

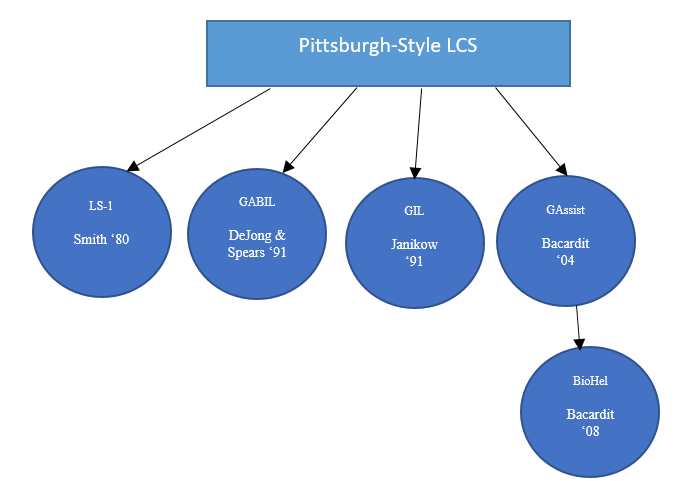
Figure 3 illustrate the family tree of Michigan-Style LCS. Let’s take a look at properties of each one. (See [[2](#Ref2), [17](#Ref17),  [18](#Ref18)] for a previous historical reviews).

[John Henry Holland](https://en.wikipedia.org/wiki/John_Henry_Holland) was best known for his work popularizing [genetic algorithms](https://en.wikipedia.org/wiki/Genetic_algorithm) (GA), through his ground-breaking book "Adaptation in Natural and Artificial Systems" [[1](#Ref1)] in 1975 and his formalization of [Holland's schema theorem](https://en.wikipedia.org/wiki/Holland%27s_schema_theorem). In 1976, Holland conceptualized an extension of the GA concept to what he called a "cognitive system", [[19](#Ref19)] and provided the first detailed description of what would become known as the first learning classifier system in the paper "Cognitive Systems based on Adaptive Algorithms" [[20](#Ref20)]. This first system, named Cognitive System One (CS-1) was conceived as a modeling tool, designed to model a real system (i.e. *environment*) with unknown underlying dynamics using a population of human readable rules. The goal was for a set of rules to perform [online machine learning](https://en.wikipedia.org/wiki/Online_machine_learning) to adapt to the environment based on infrequent payoff/reward (i.e. reinforcement learning) and apply these rules to generate a behavior that matched the real system. This early, ambitious implementation was later regarded as overly complex, yielding inconsistent results [[21](#Ref21)].

Beginning in 1980, [Kenneth de Jong](https://en.wikipedia.org/wiki/Kenneth_A_De_Jong) and his student Stephen Smith took a different approach to rule-based machine learning with (LS-1), where learning was viewed as an offline optimization process rather than an online adaptation process [[22](#Ref22), [23](#Ref23), [24](#Ref24)]. This new approach was more similar to a standard genetic algorithm but evolved independent sets of rules. In 1986, Holland developed what would be considered the standard Michigan-style LCS for the next decade [[25](#Ref25)].

Interest in learning classifier systems was reinvigorated in the mid-1990s largely due to two events; the development of the [Q-Learning](https://en.wikipedia.org/wiki/Q-learning) algorithm [[26](#Ref26)] for [reinforcement learning](https://en.wikipedia.org/wiki/Reinforcement_learning), and the introduction of significantly simplified Michigan-style LCS architectures by Stewart Wilson [[27](#Ref27), [28](#Ref28)]. Wilson's Zeroth-level Classifier System (ZCS) focused on increasing algorithmic understandability based on Holland’s standard LCS implementation [[25](#Ref25)]. This was done, in part, by removing rule-bidding and the internal message list, essential to the original Bucket Brigade Algorithm [[29](#Ref29)] credit assignment, and replacing it with a hybrid BBA/[Q-Learning](https://en.wikipedia.org/wiki/Q-learning) strategy. ZCS demonstrated that a much simpler LCS architecture could perform as well as the original, more complex implementations. However, ZCS still suffered from performance drawbacks including the proliferation of over-general classifiers.

In 1995, Wilson published his landmark paper, "Classifier fitness based on accuracy" in which he introduced the classifier system XCS, the best known and best studied LCS algorithm, is Michigan-style, was designed for reinforcement learning but can also perform supervised learning, applies incremental learning that can be either online or offline, applies accuracy-based fitness, and seeks to generate a complete action mapping. Back acronymed to be eXtended Classifier System [[27](#Ref27)]. XCS took the simplified architecture of ZCS and added an accuracy-based fitness, a niche GA (acting in the action set [A]), an explicit generalization mechanism called *subsumption*, and an adaptation of the [Q-Learning](https://en.wikipedia.org/wiki/Q-learning) credit assignment. XCS was popularized by its ability to reach optimal performance while evolving accurate and maximally general classifiers as well as its impressive problem flexibility (able to perform both [reinforcement learning](https://en.wikipedia.org/wiki/Reinforcement_learning) and [supervised learning](https://en.wikipedia.org/wiki/Supervised_learning)). ZCS alternatively became synonymous with *strength-based LCS*. XCS is also important, because it successfully bridged the gap between LCS and the field of [reinforcement learning](https://en.wikipedia.org/wiki/Reinforcement_learning). Following the success of XCS, LCS were later described as reinforcement learning systems endowed with a generalization capability [[30](#Ref30)]. [Reinforcement learning](https://en.wikipedia.org/wiki/Reinforcement_learning) typically seeks to learn a value function that 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 focusing on high payoff niches in the environment (as was the case with strength-based LCS). XCS inspired the development of a whole new generation of LCS algorithms and applications. In 1995, Congdon was the first to apply LCS to real world [epidemiological](https://en.wikipedia.org/wiki/Epidemiology) investigations of disease [[31](#Ref31)] followed closely by Holmes who developed the BOOLE**++** [[32](#Ref32)],EpiCS[[33](#Ref33)], and later EpiXCS[[34](#Ref34)] for [epidemiological](https://en.wikipedia.org/wiki/Epidemiology) classification. These early works inspired later interest in applying LCS algorithms to complex and large-scale [data mining](https://en.wikipedia.org/wiki/Data_mining) tasks epitomized by [bioinformatics](https://en.wikipedia.org/wiki/Bioinformatics) applications. In 1998, Stolzmann introduced anticipatory classifier systems (ACS) which included rules in the form of 'condition-action-effect, rather than the classic 'condition-action' representation [[35](#Ref35)]. ACS was designed to predict the perceptual consequences of an action in all possible situations in an environment. In other words, 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 will be executed. This family of LCS algorithms is best suited to multi-step problems, planning, and speeding up learning, or disambiguating perceptual aliasing (i.e. where the same observation is obtained in distinct states but requires different actions). Butz later pursued this anticipatory family of LCS developing a number of improvements to the original method [[36](#Ref36)]. In 2002, Wilson introduced XCSF, adding a computed action in order to perform function approximation [[37](#Ref37)]. In 2003, Bernado-Mansilla introduced a sUpervised Classifier System (UCS), which specialized the XCS algorithm to the task of [supervised learning](https://en.wikipedia.org/wiki/Supervised_learning), single-step problems, and forming a best action set. UCS removed the [reinforcement learning](https://en.wikipedia.org/wiki/Reinforcement_learning) strategy in favor of a simple, accuracy-based rule fitness as well as explore /exploit learning phases, characteristic of many reinforcement learners. Bull introduced a simple accuracy-based LCS (YCS) [[38](#Ref38)] and a simple strength-based LCS Minimal Classifier System (MCS) [[39](#Ref39)] in order to develop a better theoretical understanding of the LCS framework.



**Fig 4.**  Pittsburgh-Style LCS Implementation

Bacardit introduced GAssist [[3](#Ref3)] Genetic clASSIfier system, Accuracy-based Pittsburgh learning classifier system with default action, incremental update and novel representation. Now superseded by BioHEL. BioHEL, Bioinformatics-oriented Hierarchical Evolutionary Learning - BioHEL is designed to handle large-scale, e.g. bioinformatics, datasets using a meta-representation Pittsburgh-style LCSs designed for [data mining](https://en.wikipedia.org/wiki/Data_mining) and [scalability](https://en.wikipedia.org/wiki/Scalability) to large datasets in [bioinformatics](https://en.wikipedia.org/wiki/Bioinformatics) applications [[40](#Ref40)]. In 2008, Drugowitsch published the book titled "Design and Analysis of Learning Classifier Systems" including some theoretical examination of LCS algorithms [[41](#Ref41)]. Butz introduced the first rule online learning visualization within a [GUI](https://en.wikipedia.org/wiki/Graphical_user_interface) for XCSF [[42](#Ref42)]. Urbanowicz extended the UCS framework and introduced ExSTraCS**,** Extended Supervised Tracking and Classifying System, adding expert knowledge-guided learning, attribute tracking for heterogeneous subgroup identification, and a number of other heuristics to handle complex, noisy, and larger-scale (e.g. bioinformatics) data mining. Explicitly designed for [supervised learning](https://en.wikipedia.org/wiki/Supervised_learning) in noisy problem domains (e.g. epidemiology and bioinformatics) [[43](#Ref43)]. ExSTraCS integrated (1) expert knowledge to drive covering and genetic algorithm towards important features in the data [[44](#Ref44)],(2) a form of long-term memory referred to as attribute tracking [[45](#Ref45)], allowing for more efficient learning and the characterization of heterogeneous data patterns, and (3) a flexible rule representation similar to Bacardit's mixed discrete-continuous attribute list representation.

Both Bacardit and Urbanowicz explored statistical and visualization strategies to interpret LCS rules and perform knowledge discovery for data mining. Browne and Iqbal explored the concept of reusing building blocks in the form of code fragments and were the first to solve the 135-bit multiplexer benchmark problem by first learning useful building blocks from simpler multiplexer problems [[46](#Ref46)]. ExSTraCS 2.0 was later introduced to improve Michigan-style LCS scalability, successfully solving the 135-bit multiplexer benchmark problem for the first time directly. The n-bit [multiplexer](https://en.wikipedia.org/wiki/Multiplexer) problem is highly [epistatic](https://en.wikipedia.org/wiki/Epistasis) and [heterogeneous](https://en.wikipedia.org/wiki/Homogeneity_and_heterogeneity), making it a very challenging [machine learning](https://en.wikipedia.org/wiki/Machine_learning) task [[47](#Ref47)].

In this section we considered a history of LCS and a few variants.

# GBML Tool

We found a practical tool called “KEEL” (Knowledge Extraction based on Evolutionary Learning) is an open source Java software tool that can be used for a large number of different knowledge data discovery tasks. KEEL provides a simple GUI based on data flow to design experiments with different datasets and computational intelligence algorithms (paying special attention to evolutionary algorithms) in order to assess the behavior of the algorithms. It contains a wide variety of classical knowledge extraction algorithms, preprocessing techniques (training set selection, feature selection, discretization, and imputation methods for missing values, among others), computational intelligence based learning algorithms, hybrid models, statistical methodologies for contrasting experiments and so forth. It allows to perform a complete analysis of new computational intelligence proposals in comparison to existing ones. Moreover, KEEL has been designed with a two-fold goal: research and education. You can visit their site (keel.es) to download the software and manuals [[48](#Ref48), [49](#Ref49)].

**References**

1. Holland J H. Adaptation in natural and artificial systems.  
   *University of Michigan Press*, Ann Arbor, 1975.
2. Urbanowicz, Ryan J., Moore, Jason H. [Learning Classifier Systems: A Complete Introduction, Review, and Roadmap](http://www.hindawi.com/archive/2009/736398/). *Journal of Artificial Evolution and Applications*. 2009.
3. Jaume Bacardit Penarroya. Pittsburgh Genetic-Based Machine Learning in the Data Mining era: Representations, generalization, and run-time, PHD diss, Ramon Llull University, Barcelona, Spain, 2004.
4. Rechenberg, I. Evolutions strategie: ptimierung technischer Systeme nach Prinzipien der biologischen Evolution. Stuttgart: *Fromman-Holzboog Verlag*, 1973.
5. Fogel, L. J. On the organization of intellect. Doctoral dissertation, *University of California*, Los Angeles, 1964.
6. Goldberg, D. E. Genetic algorithms in search, optimization and machine learning. *Addison-Wesley Publishing Company, Inc*, 1989a.
7. Koza, J. R. Genetic programming. *Cambridge, Massachusetts: The MIT Press*, 1992.
8. Larranaga, P., & Lozano, J. (Eds.). Estimation of Distribution Algorithms, A New Tool for Evolutionary Computation. Genetic Algorithms and Evolutionary Computation. *Kluwer Academic Publishers*. 2002.
9. Venturini, G. Sia: A supervised inductive algorithm with genetic search for learning attributes based concepts. In Brazil, P. B. (Ed.), Machine Learning: ECML-93 - *Proc. of the European Conference on Machine Learning,* pp. 280–296, *Berlin, Heidelberg: Springer Verlag*, 1993.
10. CEZARY Z. JANIKOW. A Knowledge-Intensive Genetic Algorithm for Supervised Learning. *Kluwer Academic Publishers*, Boston, Manufactured in The Netherlands, 1993.
11. Cord´on, O., Herrera, F., Hoffmann, F., & Magdalena, L. Genetic fuzzy systems. Evolutionary tuning and learning of fuzzy knowledge bases. *World Scientific,* 2001.
12. Wilson, Stewart W. [Classifier systems and the animate problem](https://link.springer.com/article/10.1007/BF00058679)*. Machine Learning, Springer*, 1987.
13. Lanzi, P L. Learning classifier systems from a reinforcement learning perspective. *Springer Verlag*, 2002.
14. Ben-David, Shai; Kushilevitz, Eyal; Mansour, Yishay. [Online Learning versus Offline Learning](https://link.springer.com/article/10.1023/A:1007465907571), Journal of Machine Learning, 1997.
15. Tim Kovacs. Strength or Accuracy? Fitness Calculation in Learning Classifier Systems. *International work shop on computer science*, 2000.
16. Nakata, Masaya & Lanzi, Pier Luca & Kovacs, Tim & Takadama, Keiki. Complete action map or best action map in accuracy-based reinforcement learning classifier systems. *Proceedings of the 2014 Genetic and Evolutionary Computation Conference*, 2014.
17. Lanzi P-L, Riolo R. A roadmap to the last decade of  
    learning classifier system research. In: Lanzi P-L, Stolzmann, Wilson SW (eds) Learning classifier systems: from foundations to applications. Springer, New York, pp 33–62, 2000.
18. Larry Bull, A brief history of learning classifier systems: from CS-1 to XCS and its variants. *Springer-Verlag,* Berlin Heidelberg, 2015.
19. Holland JH. Adaptation. In: Rosen R, Snell F (eds) Progress in theoretical biology, vol 4. *Academic Press*, New York, pp 263–293, 1976.
20. Holland JH, Reitman JS. Cognitive systems based on adaptive algorithms Reprinted in: Evolutionary computation. *The fossil record. In: David BF* (ed) IEEE Press, New York 1998, 1978.
21. Lanzi, Pier Luca[. Learning classifier systems: then and now](https://link.springer.com/article/10.1007/s12065-007-0003-3), Evolutionary Intelligence, *Springer, vo1*, 2008.
22. Smith S. A learning system based on genetic adaptive algorithms. Ph.D. thesis, Department of Computer Science*, University of Pittsburgh*, 1980.
23. Smith S. [Flexible learning of problem solving heuristics through adaptive search](https://www.researchgate.net/profile/Stephen_Smith14/publication/220815785_Flexible_Learning_of_Problem_Solving_Heuristics_Through_Adaptive_Search/links/0deec52c18dbd0dd53000000.pdf). In: *Eighth international joint conference on artificial intelligence*. Morgan Kaufmann, Los Altos, pp 421–425, 1983.
24. De Jong KA. Learning with genetic algorithms: an overview. *Mach Learn*, 1988.

1. [Holland, John H. Escaping brittleness: the possibilities of general purpose learning algorithms applied to parallel rule-based system.](http://dl.acm.org/citation.cfm?id=216016)*[Machine learning,](http://dl.acm.org/citation.cfm?id=216016)* [1986.](http://dl.acm.org/citation.cfm?id=216016)
2. Watkins, Christopher John Cornish Hellaby. Learning from delayed rewards. PhD diss., *University of Cambridge*, 1989.
3. Wilson, Stewart W. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, vol3, 1995.
4. Wilson, Stewart W. [ZCS: A Zeroth Level Classifier System](https://dx.doi.org/10.1162/evco.1994.2.1.1). *Evolutionary Computation*, vol2, 1994.
5. Holland, John H. [Properties of the Bucket Brigade](http://dl.acm.org/citation.cfm?id=645511.657087). *Proceedings of the 1st International Conference on Genetic Algorithms*. Hillsdale, NJ, USA: L.Erbaum Associates Inc, 1985
6. Lanzi, P. L. [Learning classifier systems from a reinforcement learning perspective](https://link.springer.com/article/10.1007/s005000100113). *Soft Computing*, vol6, Springer Verlag, 2002.
7. Congdon, Clare Bates. A comparison of genetic algorithms and other machine learning systems on a complex classification task from common disease research. PhD diss*, The University of Michigan*, 1995.
8. Holmes, John H. [A Genetics-Based Machine Learning Approach to Knowledge Discovery in Clinical Data](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2233061). *Proceedings of the AMIA Annual Fall Symposium*, 1996.
9. Holmes, John H. [Discovering Risk of Disease with a Learning Classifier System](https://pdfs.semanticscholar.org/71e4/eb6c630dee4b762e74b2970f6dc638a351ab.pdf). *In ICGA*, pp. 426-433. 1997.
10. Holmes, John H., and Jennifer A. Sager. [Rule discovery in epidemiologic surveillance data using EpiXCS: an evolutionary computation approach](https://link.springer.com/10.1007%2F11527770_60). *In Conference on Artificial Intelligence in Medicine in Europe*, pp. 444-452. Springer Berlin Heidelberg, 2005.
11. W. Stolzmann. Anticipatory classifier systems. *In Proceedings of the 3rd annual Genetic Programming Conference*, pp.658-664, 1998.
12. Butz, Martin V. [Biasing exploration in an anticipatory learning classifier system](https://pdfs.semanticscholar.org/3572/7a56fcce7a73ccc43e5bfa19389780e6d436.pdf). *In International Workshop on Learning Classifier Systems*, pp. 3-22. Springer Berlin Heidelberg, 2001.
13. Wilson, Stewart W. [Classifiers that approximate functions](https://link.springer.com/article/10.1023/A:1016535925043)*. Natural Computing*, vol1, 2001.
14. Bull, Larry. [A simple accuracy-based learning classifier system](https://pdfs.semanticscholar.org/120c/8f5057995c36ee60ec320c2263b20af05444.pdf). *Learning Classifier Systems Group Technical Report* UWELCSG03-005, University of the West of England, Bristol, UK, 2003.
15. Bull, Larry. [A simple payoff-based learning classifier system](https://link.springer.com/chapter/10.1007/978-3-540-30217-9_104). In *International Conference on Parallel Problem Solving from Nature*, pp. 1032-1041. Springer Berlin Heidelberg, 2004.
16. Bacardit, Jaume; Burke, Edmund K.; Krasnogor, Natalio. [Improving the scalability of rule-based evolutionary learning](https://link.springer.com/article/10.1007/s12293-008-0005-4)*. Memetic Computing*, vol1, 2008.
17. Drugowitsch, Jan. Design and Analysis of Learning Classifier System. Springer. Studies in Computational Intelligence, 2008.
18. Stalph, Patrick O.; Butz, Martin V. JavaXCSF: The XCSF Learning Classifier System in java. SIGEV Olution, vol4, 2010.
19. Urbanowicz, Ryan J., Gediminas Bertasius, and Jason H. Moore. [An extended Michigan-style learning classifier system for flexible supervised learning, classification, and data mining](http://www.seas.upenn.edu/~gberta/uploads/3/1/4/8/31486883/urbanowicz_2014_exstracs_algorithm.pdf). *In International Conference on Parallel Problem Solving from Nature*, pp. 211-221. Springer International Publishing, 2014.
20. Urbanowicz, Ryan J., Delaney Granizo-Mackenzie, and Jason H. Moore. Using expert knowledge to guide covering and mutation in a Michigan style learning classifier system to detect epistatic and heterogeneity*. In International Conference on Parallel Problem Solving from Nature, pp. 266-275, Springer Berlin Heidelberg, 2012.*
21. Urbanowicz, Ryan; Granizo-Mackenzie, Ambrose; Moore, Jason. Instance-linked Attribute Tracking and Feedback for Michigan-style Supervised Learning Classifier Systems. Proceedings of the14th Annual Conference on Genetic and Evolutionary Computation, 2012.
22. Iqbal, Muhammad; Browne, Will N.; Zhang, Mengjie. Reusing Building Blocks of Extracted Knowledge to Solve Complex, Large-Scale Boolean Problems. IEEE Transactions on Evolutionary Computation, 2014.
23. Urbanowicz. Ryan J.; Moore, Jason H. [ExSTraCS 2.0: description and evaluation of a scalable learning classifier system](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4583133)*. Evolutionary Intelligence*, 2015.
24. J. Alcalá-Fdez, L. Sánchez, S. García, M.J. del Jesus, S. Ventura, J.M. Garrell, J. Otero, C. Romero, J. Bacardit, V.M. Rivas, J.C. Fernández, F. Herrera. KEEL: A Software Tool to Assess Evolutionary Algorithms to Data Mining Problems. *Soft Computing*, 2009.
25. J. Alcalá-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. García, L. Sánchez, F. Herrera. KEEL Data-Mining Software Tool: Data Set Repository, Integration of Algorithms and Experimental Analysis Framework. *Journal of Multiple-Valued Logic and Soft Computing*, 2011.