

# 30 Genetics-Based Machine Learning

*Tim Kovacs*

Department of Computer Science, University of Bristol, UK  
kovacs@cs.bris.ac.uk

<b>1</b>	<b><i>Introduction</i></b>	<b>938</b>
<b>2</b>	<b><i>A Framework for GBML</i></b>	<b>941</b>
<b>3</b>	<b><i>GBML Areas</i></b>	<b>947</b>
<b>4</b>	<b><i>Conclusions</i></b>	<b>972</b>

## Abstract

---

This is a survey of the field of genetics-based machine learning (GBML): the application of evolutionary algorithms (ES) to machine learning. We assume readers are familiar with evolutionary algorithms and their application to optimization problems, but not necessarily with machine learning. We briefly outline the scope of machine learning, introduce the more specific area of supervised learning, contrast it with optimization and present arguments for and against GBML. Next we introduce a framework for GBML, which includes ways of classifying GBML algorithms and a discussion of the interaction between learning and evolution. We then review the following areas with emphasis on their evolutionary aspects: GBML for subproblems of learning, genetic programming, evolving ensembles, evolving neural networks, learning classifier systems, and genetic fuzzy systems.

## 1 Introduction

---

Genetics-based machine learning (GBML) is the application of evolutionary algorithms (EAs) to machine learning. We assume readers are familiar with EAs, which are well documented elsewhere, and their application to optimization problems. In this introductory section we outline the scope of machine learning, introduce the more specific area of supervised learning, and contrast it with optimization. However, the treatment is necessarily brief and readers who desire to work in GBML are strongly advised to first gain a solid foundation in non-evolutionary approaches to machine learning. 🔗 [Sect. 2](#) describes a framework for GBML, which includes ways of classifying GBML algorithms and a discussion of the interaction between learning and evolution. 🔗 [Sect. 3](#) reviews the work of a number of GBML communities with emphasis on their evolutionary aspects. Finally, 🔗 [Sect. 4](#) concludes the chapter.

### What is Missing

Given the breadth of the field and the volume of the literature, the coverage herein is necessarily somewhat arbitrary and misses a number of significant subjects. These include a general introduction to machine learning including the structure of learning problems and their fitness landscapes (which we must exploit in order to learn efficiently), non-evolutionary algorithms (which constitute the majority of machine learning methods, and include both simple and effective methods), and theoretical limitations of learning (such as the *no free lunch* theorem for supervised learning (Wolpert 1996) and the *conservation law of generalization* (Schaffer 1994)). Also missing is coverage of GBML for clustering, reinforcement learning, Bayesian networks, artificial immune systems, artificial life, and application areas. Finally, some areas which have been touched on have been given an undeservedly cursory treatment, including EAs for data preparation (e.g., feature selection), coevolution, and comparisons between GBML and non-evolutionary alternatives. However, Freitas (2002a) contains good treatments of GBML for, among others, clustering and data preparation.

### 1.1 Machine Learning

---

Machine learning is concerned with machines that improve with experience and reason inductively or abductively in order to optimize, approximate, summarize, generalize from specific examples to general rules, classify, make predictions, find associations, propose

explanations, and propose ways of grouping things. For simplicity, we will restrict ourselves to classification and optimization problems.

### Inductive Generalization

Inductive generalization refers to the inference of unknown values from known values. Induction differs from deduction in that the unknown values are in fact *unknowable*, which gives rise to fundamental limitations in what can be learned. (If at a later time new data makes all such values known the problem ceases to be inductive.) Given that the unknown values are unknowable, we *assume* they are correlated with the known values and we seek to learn the correlations. We formulate our objective as maximizing a function of the unknown values. In evolutionary computation this objective is called the fitness function, whereas in other areas the analogous feedback signal may be known as the error function, or by other names. There is *no need for induction* if: (i) all values are known and (ii) there is enough time to process them. We consider two inductive problems: function optimization and learning. We will not deal with abduction.

#### 1-Max: A Typical Optimization Problem

The 1-max problem is to maximize the number of 1s in a binary string of length  $n$ . The optimal solution is trivial for humans although it is less so for EAs. The *representation* of this problem follows. Input: none. Output: bit strings of length  $n$ . *Data generation*: we can generate as many output strings as time allows, up to the point where we have enumerated the search space (in which case the problem ceases to be inductive). *Training*: the fitness of a string is the number of 1s it contains. We can evaluate a learning method on this task by determining how close it gets to the known optimal solution. In more realistic problems, the optimum is not known and we may not even know the maximum possible fitness. Nonetheless, for both toy and realistic problems, we can evaluate how much training was needed to reach a certain fitness and how a learning method compares to others.

#### Classification of Mushrooms: A Typical Learning Problem

Suppose we want to classify mushroom species as poisonous or edible given some training data consisting of features of each species (color, size and so on) including edibility. Our task is to learn a hypothesis, which will classify new species whose edibility is unknown. *Representation*: the input is a set of nominal attributes and the output is a binary label indicating edibility. *Data generation*: a fixed dataset of input/output examples derived from a book. Typically the dataset is far, far smaller than the set of possible inputs, and we partition it into train and test sets. *Training*: induce a hypothesis which maximizes classification accuracy on the train set. *Evaluation*: evaluate the accuracy of the induced hypothesis on the test set, which we take as an indication of how well a newly encountered species might be classified.

#### Terminology in Supervised Learning

Although many others exist, we focus on the primary machine learning paradigm: standard supervised learning (SL), of which the preceding mushroom classification task is a good example. In SL we have a dataset of labeled input/output pairs. Inputs are typically called instances or exemplars and are factored into attributes (also called features), while outputs are called classes (for classification tasks) or the output is called the dependent variable (for regression tasks).

### Comparison of Supervised Learning and Optimization

In SL we typically have limited training data and it is crucial to find a good inductive bias for later use on new data. Consequently, we *must* evaluate the generalization of the induced hypothesis from the train set to the previously unused test set. In contrast, in optimization we can typically generate as much data as time allows and we can typically evaluate any output. We are concerned with finding the optimum output in minimum time, and, specifically, inducing which output to evaluate next. As a result no test set is needed.

### Issues in Supervised Learning

A great many issues arise in SL including overfitting, underfitting, producing human readable results, dealing with class imbalances in the training data, asymmetric cost functions, noisy and nonstationary data, online learning, stream mining, learning from particularly small datasets, learning when there are very many attributes, learning from positive instances only, incorporating bias and prior knowledge, handling structured data, and using additional unlabeled data for training. None of these will be dealt with here.

## 1.2 Arguments For and Against GBML

---

GBML methods are a niche approach to machine learning and much less well-known than the main non-evolutionary methods, but there are many good reasons to consider them.

### Accuracy

Importantly, the classification accuracy of the best evolutionary and non-evolutionary methods are comparable (Freitas 2002a, Sect. 12.1.1).

### Synergy of Learning and Evolution

GBML methods exploit the synergy of learning and evolution, combining global and local search and benefitting from the Baldwin effect's smoothing of the fitness landscape ➤ Sect. 2.3.

### Epistasis

There is some evidence that the accuracy of GBML methods may not suffer from epistasis as much as typical non-evolutionary greedy search (Freitas 2002a, Sect. 12.1.1).

### Integrated Feature Selection and Learning

GBML methods can combine feature selection and learning in one process. For instance, feature selection is intrinsic in LCS methods ➤ Sect. 3.5.

### Adapting Bias

GBML methods are well-suited to adapting inductive bias. We can adapt representational bias by, for example, selecting rule condition shapes ➤ Sect. 3.5.3, and algorithmic bias by, for example, evolving learning rules ➤ Sect. 3.4.

### Exploiting Diversity

We can exploit the diversity of a population of solutions to combine and improve predictions (the ensemble approach, ➤ Sect. 3.3) and to generate Pareto sets for multi-objective problems.

**Dynamic Adaptation**

All the above can be done dynamically to improve accuracy, to deal with nonstationarity, and to minimize population size. This last is of interest in order to reduce overfitting, improve run-time, and improve readability.

**Universality**

Evolution can be used as a wrapper for *any* learner.

**Parallelization**

Population-based search is easily parallelized.

**Suitable Problem Characteristics**

From an optimization perspective, learning problems are typically large, non-differentiable, noisy, epistatic, deceptive, and multimodal (Miller et al. 1989). To this list we could add high-dimensional and highly constrained. EAs are a good choice for such problems. See Cantú-Paz and Kamath (2003) and ➤ Sect. 3.4 for more arguments in favor of and against GBML.

**Algorithmic Complexity**

GBML algorithms are typically more complex than their non-evolutionary alternatives. This makes them harder to implement and harder to analyze, which means there is less theory to guide parameterization and development of new algorithms.

**Increased Run-time**

GBML methods are generally much slower than the non-evolutionary alternatives.

**Suitability for a Given Problem**

No single learning method is a good choice for all problems. For one thing the bias of a given GBML method may be inappropriate for a given problem. Problems to which GBML methods are particularly prone include prohibitive run-time (or set-up time) and that simpler and/or faster methods may suffice. Furthermore, even where GBML methods perform better, the improvements may be marginal. See the strengths, weaknesses, opportunities, threats (SWOT) analysis of GBML in Orriols-Puig et al. (2008b) for more.

## 2 A Framework for GBML

---

The aim of the framework presented in this section is to structure the range of GBML systems into more specific categories about which we can make more specific observations than we could about GBML systems as a whole. We present two categorizations. In the first (➤ Sect. 2.1), GBML systems are classified by their role in learning; specifically their applications to i) subproblems of machine learning, ii) learning itself, or iii) meta-learning. In the second categorization (➤ Sect. 2.2), GBML systems are classified by their high-level algorithmic approach as either Pittsburgh or Michigan systems. Following this, in ➤ Sect. 2.3 we briefly review ways in which learning and evolution interact and in ➤ Sect. 2.4 we consider various models of GBML not covered earlier.

Before proceeding, we note that evolution can output a huge range of phenotypes, from scalar values to complex learning agents, and that agents can be more or less plastic (independent of evolution). For example, if evolution outputs a fixed hypothesis, that hypothesis has no plasticity. In contrast, evolution can output a neural net which, when trained with back-propagation, can learn much. (In the latter approach, evolution may specify the network structure while backpropagation adapts the network weights.)

### Structure of GBML Systems

We can divide any evolutionary (meta)-learning system into the following parts: (i) *Representation*, which consists of the genotype (the learner's genes) and phenotype (the learner itself, built according to its genes). In simple cases, the genotype and phenotype may be identical, for example with the simple ternary LCS rules of [Sect. 3.5.2](#). In other cases, the two are very different and the phenotype may be derived through a complex developmental process (as in nature); see [Sect. 3.4](#) on developmental encodings for neural networks. (ii) *Feedback*, which consists of the learner's objective function (e.g., the error function in supervised learning) and the fitness function which guides evolution. (iii) *The production system*, which applies the phenotypes to the learning problem. (iv) *The evolutionary system*, which adapts genes.

## 2.1 Classifying GBML Systems by Role

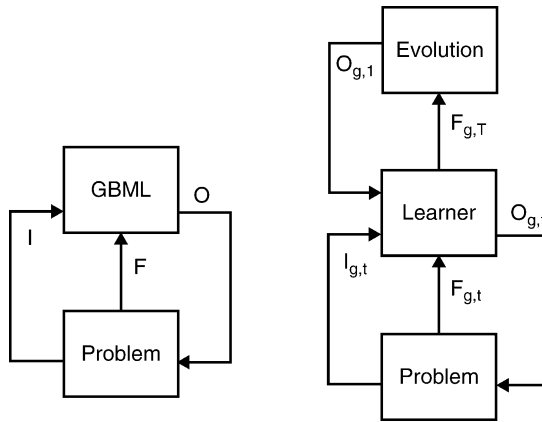
In order to contrast learning and meta-learning, we define learning as a process which outputs a fixed hypothesis. Accordingly, when evolution adapts hypotheses it is a learner and when it adapts learners it is a meta-learner. However, this distinction between learning and meta-learning should not be overemphasized; if evolution outputs a learner with little plasticity then evolution may be largely responsible for the final hypothesis, and in this case plays both a learning and a meta-learning role. Furthermore, both contribute to the ultimate goal of adaptation, and in [Sect. 2.3](#) we will see ways in which they interact.

Evolution as learning is illustrated in the left of [Fig. 1](#), which shows a GBML agent interacting directly with the learning problem. In contrast, the right of the figure shows GBML as meta-learning: the learner (or a set of learners) is the output of evolution, and the learner interacts directly with the learning problem while evolution interacts with it only through learners. At time step 1 of each generation, evolution outputs a learning agent and at the generation's final step,  $T$ , it receives an evaluation of the learner's fitness. During the intervening time steps the learner interacts with the problem. This approach to meta-learning is *universal* as any learner can be augmented by GBML, and is related to the wrapper approach to feature selection in [Sect. 3.1](#).

Meta-learning is a broad term with different interpretations but the essential idea is *learning about learning*. A meta-learner may optimize parameters of a learner, learn which learner to apply to a given input or a given problem, learn which representation(s) to use, optimize the update rules used to train learners, learn an algorithm which solves the problem, evolve an ecosystem of learners, and potentially be open ended. See Vilalta and Drissi (2002) and Giraud-Carrier and Keller (2002) on non-evolutionary meta-learning and Burke et al. (2003), Krasnogor (2004), Krasnogor and Gustafson (2004), and Burke and Kendall (2005) on the hyperheuristics (*heuristics to learn heuristics*) approach, of which a subset is evolutionary.

■ Fig. 1

(Left) GBML as learner. Input, Output, and Fitness shown. (Right) GBML as meta-learner. Subscripts denote generation and time step ( $1 \dots T$ ).



A third role for evolution is application to various subproblems of learning including feature selection, feature construction, and other optimization roles within learning agents. In these cases evolution neither outputs the final hypothesis nor outputs a learning agent which does so. ➤ [Section 3.1](#) deals with such applications.

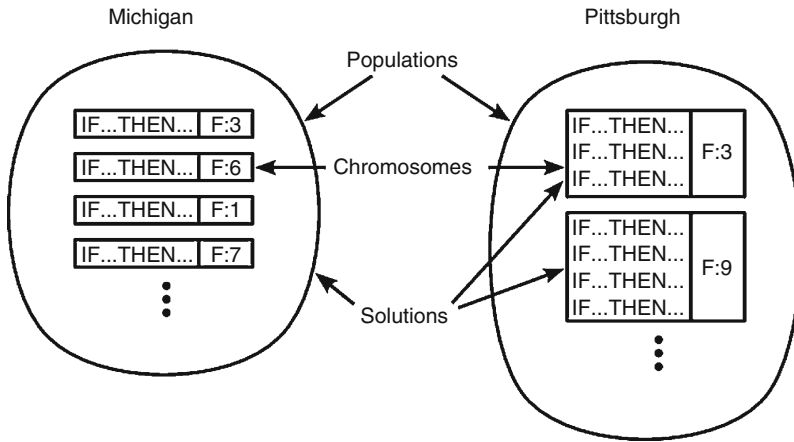
## 2.2 Classifying GBML Systems Algorithmically

In the Pittsburgh (Pitt) approach, one chromosome encodes one solution. We assume that fitness is assigned to chromosomes, so in Pitt systems it is assigned to solutions. This leaves a credit assignment problem: how did the chromosome's component genes contribute to the observed fitness of the chromosome? This is left to evolution as this is what EAs are designed to deal with. In the Michigan approach, one solution is (typically) represented by many chromosomes and so fitness is assigned to partial solutions. Credit assignment differs from the Pitt case as chromosomes not only compete for reproduction but may also complement and cooperate with each other. This gives rise to the issues of how to encourage cooperation, complementarity, and coverage of the inputs, all of which makes designing an effective fitness function more complex than in Pitt systems. In Michigan systems, the credit assignment problem is how to measure a chromosome's contributions to the overall solution, as reflected in the various aspects of fitness just mentioned. To sum up the difficulty in Michigan systems: the best set of chromosomes may not be the set of best (i.e., fittest) chromosomes (Freitas 2002a). To illustrate, ➤ [Fig. 2](#) depicts the representation used by Pitt and Michigan versions of the rule-based systems called learning classifier systems (LCS) (see ➤ [Sect. 3.5](#)). In a Pittsburgh LCS, a chromosome is a variable-length *set* of rules, while in a Michigan LCS, a chromosome is a single fixed-length rule.

Although the Pittsburgh and Michigan approaches are generally presented as two discrete cases, some hybrids exist (e.g., Wilcox (1995)).

■ Fig. 2

Michigan and Pittsburgh rule-based systems compared. The  $F:x$  associated with each chromosome indicates its fitness.



### Pittsburgh and Michigan Compared

Pittsburgh systems (especially naive implementations) are slower, since they evolve more complex structures and they assign credit at a less specific (and hence less informative) level. (See, however, Sect. 3.5.5 on the windowing approach to improving run-time and Bacardit et al. (2009a) for an approach which avoids performing matching operations between rule conditions and irrelevant features.) Additionally, their chromosomes and their genetic operators are more complex. On the other hand they face less complex credit assignment problems and hence are more robust, that is, more likely to adapt successfully. Michigan systems use a finer grain of credit assignment than the Pittsburgh approach, which means bad partial solutions can be deleted without restarting from scratch. This makes them more efficient and also more suitable for incremental learning. However, credit assignment is more complex in Michigan systems. Since the solution is a *set* of chromosomes: (i) the population must not converge fully, and (ii) as noted, the best set of chromosomes may not be the set of best chromosomes.

The two approaches also tend to be applied in different ways. Pitt systems are typically used offline and are algorithm-driven; the main loop processes each chromosome in turn and seeks out data to evaluate them (which is how a standard genetic algorithm (GA) works, although fitness evaluation is typically simpler in a GA). In contrast, Michigan systems are typically used online and are data-driven; the main loop processes each data input in turn and seeks out applicable chromosomes (see Fig. 3). As a result, Michigan systems are more often used as learners (though not necessarily more often as meta-learners) for reinforcement learning, which is almost always online. The Michigan approach has mainly been used with LCS. See Greene and Smith (1993), Janikow (1993), Wilcox (1995), Freitas (2002b) and Kovacs (2004) for comparison of the approaches.

### Iterative Rule Learning

IRL is a variation on the Michigan approach in which, as usual, one solution is represented by many chromosomes, but only the single best chromosome is selected after each run, which



■ Fig. 3

A basic Michigan algorithm.

On each time step:

1. Identify match set: subset of population which match current input
2. Compute support in match set for each class
3. Select class
4. Identify action set: subset of match set which advocate selected class
5. Update action set based on feedback
6. Optionally alter population

alters the coevolutionary dynamics of the system. The output of multiple runs is combined to produce the solution. The approach originated with SIA (Supervised Inductive Algorithm) (Venturini 1993; Juan Liu and Tin-Yau Kwok 2000), a supervised genetic rule learner.

### Genetic Cooperative-Competitive Learning

GCCL is another Michigan approach in which each generation is ranked by fitness and a *coverage-based filter* then allocates inputs to the first rule which correctly covers them. Inputs are only allocated to one rule per generation and rules which have no inputs allocated die at the end of a generation. The collective accuracy of the remaining rules is compared to the previous best generation, which is stored offline. If the new generation is more accurate (or the same but has fewer rules) it replaces the previous best. Examples include COGIN (Greene and Smith 1993, 1994), REGAL (Giordana and Neri 1995), and LOGENPRO (Wong and Leung 2000).

## 2.3 The Interaction of Learning and Evolution

This section briefly touches on the rich interactions between evolution and learning.

### Memetic Learning

We can characterize evolution as a form of global search, which is good at finding good basins of attraction, but poor at finding the optimum of those basins. In contrast, many learning methods are forms of local search and have the opposite characteristics. We can get the best of both by combining them, which generally outperforms either alone (Yao 1999). For example, evolving the initial weights of a neural network and then training them with gradient descent can be two orders of magnitude faster than using random initial weights (Floreano et al. 2008). Methods which combine global and local search are called *memetic* algorithms (Hart et al. 2004, 2005; Ong et al. 2006, 2007; Smith 2007; Ong et al. 2009; Rozenberg et al. 2012). See Krasnogor and Smith (2005) for a self-contained tutorial.

### Darwinian and Lamarckian Evolution

In Lamarckian evolution/inheritance, learning during an individual's lifetime directly alters the genes passed to offspring, so offspring inherit the result of their parents' learning. This does not occur in nature but can in computers and has the potential to be more efficient than Darwinian evolution since the results of learning are not thrown away. Indeed, Ackley and

Littman (1992) showed Lamarckian evolution was much faster on stationary learning tasks but Sasaki and Tokoro (1997) showed Darwinian evolution was generally better on nonstationary tasks. See also Whitley et al. (1994), Yamasaki and Sekiguchi (2000), Pereira and Costa (2001), and Whiteson and Stone (2006).

### The Baldwin Effect

The Baldwin effect is a two-part dynamic between learning and evolution which depends on *Phenotypic Plasticity* (PP): the ability to adapt (e.g., learn) during an individual's lifetime. The first aspect is this. Suppose a mutation would have no benefit except for PP. Without PP, the mutation does not increase fitness, but with PP it does. Thus PP helps evolution to adopt beneficial mutations; it effectively smooths the fitness landscape. A possible example from nature is lactose tolerance in human adults. At a recent point in human evolution a mutation occurred, which allowed adult humans to digest milk. Subsequently, humans learned to keep animals for milk, which in turn made the mutation more likely to spread. The smoothing effect on the fitness landscape depends on PP; the greater the PP the more potential there is for smoothing. All GBML methods exploit the Baldwin effect to the extent that they have PP. See Whiteson and Stone (2006, Sect. 7.2) for a short review of the Baldwin effect in reinforcement learning.

The second aspect of the Baldwin effect is genetic assimilation. Suppose PP has a cost (e.g., learning involves making mistakes). If PP can be replaced by new genes, it will be; for instance a learned behavior can become instinctive. This allows learned behaviors to become inherited without Lamarckian inheritance.

Turney (1996) has connected the Baldwin effect to inductive bias. All inductive algorithms have a bias and the Baldwin effect can be seen as a shift from weak to strong bias. When bias is weak, agents rely on learning; when bias is strong, agents rely on instinctive behavior.

### Evaluating Evolutionary Search by Evaluating Accuracy

Kovacs and Kerber (2004) point out that high classification accuracy does not imply effective genetic search. To illustrate, they initialized XCS (Wilson 1995) with random condition/action rules and disabled evolutionary search. Updates to estimates of rule utility, however, were made as usual. They found the system was still able to achieve very high training set accuracy on the widely used 6 and 11 multiplexer tasks since ineffective rules were simply given low weight in decision making, though neither removed nor replaced. Care is therefore warranted when attributing good accuracy to genetic search. A limitation of this work is that test set accuracy was not evaluated.

## 2.4 Other GBML Models

---

This section covers some models which are orthogonal to those discussed earlier.

### Online Evolutionary Computation

In many problems, especially sequential ones, feedback is very noisy and needs averaging. Whiteson and Stone (2006) allocated trials to chromosomes in proportion to their fitness with the following procedure. At each new generation, each chromosome is evaluated once only. Subsequent evaluations are allocated using a softmax distribution based on the initial fitnesses and the average fitness of a chromosome is recalculated after each evaluation. In nonstationary

problems a recency-weighted average of fitness samples is used. This approach is called *online evolutionary computation*. Its advantages are that less time is wasted evaluating weaker chromosomes, and in cases where mistakes matter, fewer mistakes are made by agents during fitness evaluations. However, the improvement is only on average; worst case performance is not improved. This is related to other work on optimizing noisy fitness functions (Stagge 1998; Beielstein and Markon 2002), except that they do not reduce online mistakes.

### Steady-State EAs

Whereas standard generational EAs replace the entire population each generation, steady state EAs replace a subset (e.g., only two in XCS). This approach is standard in Michigan LCS because they minimize disruption to the population, which is useful for online learning. Steady state EAs introduce selection for deletion as well as reproduction and this is typically biased toward lower fitness chromosomes or to reduce crowding.

### Co-evolving Learners and Problems

Another possibility not mentioned in our earlier classifications is to coevolve both learners and problems. When successful, this allows learners to gradually solve harder problems rather than tackling the most difficult problems from the start. It also allows us to search the space of problems to find those which are harder for a given learner, and to explore the dynamics between learners and problems.

## 3 GBML Areas

---

This section covers the main GBML research communities. These communities are more disjoint than the methods they use and the lines between them are increasingly blurring. For example, LCS often evolve neural networks and fuzzy rules, and some are powered by genetic programming. Nonetheless, the differences between the communities and their approaches is such that it seemed most useful to structure this section by community and not, for example, by phenotype or learning paradigm; such integrated surveys of GBML are left to the future. Many communities have reinvented the same ideas, yet each has its own focus and strengths and so each has much to learn from the others.

### 3.1 GBML for Subproblems of Learning

---

This section briefly reviews ways in which evolution has been used not for the primary task of learning – generating hypotheses – but for subproblems including data preparation and optimization within other learning methods.

#### Evolutionary Feature Selection

Some attributes (features) of the input are of little or no use in classification. We can simplify and speed learning by selecting only useful attributes to work with, especially when there are very many attributes and many contribute little. EAs are widely used in the wrapper approach to feature selection (John et al. 1994) in which the base learner (the one which generates hypotheses) is treated as a black box to be optimized by a search algorithm. In this, EAs usually give good results compared to non-evolutionary methods (Jain and Zongker 1997;

Sharpe and Glover 1999; Kudo and Skalsky 2000) but there are exceptions (Jain and Zongker 1997). In Cantú-Paz (2002) Estimation of Distribution Algorithms were found to give similar accuracy but run more slowly than a GA. More generally we can weight features (instead of making an all-or-nothing selection) and some learners can use weights directly, for example, weighted k-nearest neighbors (Raymer et al. 2000). The main drawback of EAs for feature selection is their slowness compared to non-evolutionary methods. See Martin-Bautista and Vila (1999) and Freitas (2002a, b) for overviews, and Stout et al. (2008), Bacardit et al. (2009b) for some recent real-world applications.

### Evolutionary Feature Construction

Some features are not very useful by themselves but can be when combined with others. We can leave the base learner to discover this or we can preprocess data to construct informative new features by combining existing ones, for example new feature  $f_{\text{new}} = f_1 \text{ AND } f_3 \text{ AND } f_8$ . This is also called constructive induction and there are different approaches. GP has been used to construct features out of the original attributes, for example, Hu (1998), Krawiec (2002), and Smith and Bull (2005). The original features have also been linearly transformed by evolving a vector of coefficients (Kelly and Davis 1991; Punch et al. 1993). Simultaneous feature transformation and selection has had good results (Raymer et al. 2000).

### Other Subproblems of Learning

EAs have been used in a variety of other ways. One is training set optimization in which we can partition the data into training sets (Romaniuk 1994), select the most useful training inputs (Ishibuchi and Nakashima 2000), and even generate synthetic inputs (Zhang and Veenker 1991; Cho and Cha 1996). EAs have also been used for optimization within a learner, for example, Kelly and Davis (1991) optimized weighted k-nearest neighbors with a GA, Cantú-Paz and Kamath (2003) optimized decision tree tests using a GA and an evolution strategy (ES) and Thompson (1998, 1999) optimized voting weights in an ensemble. Janikow (1993) replaced beam search in AQ with a genetic algorithm and similarly Tameddoni-Nezhad and Muggleton (2000, 2003), Divina and Marchiori (2002), and Divina et al. (2002, 2003) have investigated inductive logic programming driven by a GA.

## 3.2 Genetic Programming

Genetic Programming (GP) is a major evolutionary paradigm which evolves programs (Vanneschi and Poli 2012). The differences between GP and GAs are not precise but typically GP evolves variable-length structures, typically trees, in which genes can be functions. See Woodward (2003) for discussion. Freitas (2002a) discusses differences between GAs and GP which arise because GP representations are more complex. Among the pros of GP: (i) it is easier to represent complex languages, such as first-order logic in GP, (ii) it is easier to represent complex concepts compactly, and (iii) GP is good at finding novel and complex patterns overlooked by other methods. Among the cons of GP: (i) expressive representations have large search spaces, (ii) GP tends to overfit / does not generalize well, and (iii) variable-length representations suffer from bloat (see, e.g., Poli et al. (2008)).

While GAs are typically applied to function optimization, GP is widely applied to learning. To illustrate, there are many learning problems among the set of “typical GP problems” defined by Koza (1992), which have become more-or-less agreed benchmarks for the

GP community (Vanneschi and Poli 2012), there are many learning problems. These include the multiplexer and parity Boolean functions, symbolic regression of mathematical functions and the intertwined spirals problem, which involves the classification of two-dimensional points as belonging to one of the two spirals. GP usually follows the Pittsburgh approach. We cover the two representations most widely used for learning with GP: GP trees and decision trees.

### 3.2.1 GP Trees

#### GP Trees for Classification

Figure 4 shows the 3 multiplexer Boolean function as a truth table on the left and as a GP tree on the right. To classify an input with the GP tree: (i) instantiate the leaf variables with the input values, (ii) propagate values upward from leaves through the functions in the non-leaf nodes and (iii) output the value of the root (top) node as the classification.

#### GP Trees for Regression

In regression problems leaves may be constants or variables and non-leaves are mathematical functions. Figure 5 shows a real-valued function as an algebraic expression on the left and as

Fig. 4

Two representations of the 3 multiplexer function: truth table (left) and GP tree (right).

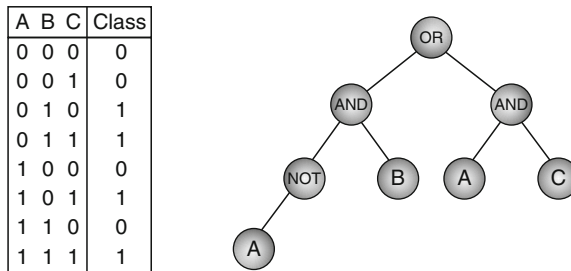
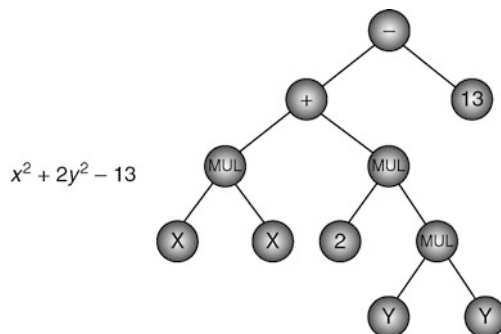


Fig. 5

Two representations of a real-valued function.



a GP tree on the right. (Note that  $x^2 + 2y^2 - 13 = ((x*x) + (2*(y*y))) - 13$ .) The output of the tree is computed in the same way as in the preceding classification example.

3.2.2 Decision Trees

Figure 6 shows the 3 multiplexer as a truth table and as a decision tree. To classify an input in such a tree: (i) start at the root (top) of tree, (ii) follow the branch corresponding to the value of the attribute in the input, (iii) repeat until a leaf is reached, and (iv) output the value of the leaf as the classification of the input.

Evolving First-Order Trees

First-order trees use both propositional and first-order internal nodes. Rouwhorst and Engelbrecht (2000) found first-order logic made trees more expressive and allowed much smaller solutions than found by the rule learner CN2 or the tree learner C4.5, with similar accuracy.

Oblique (Linear) Trees

Whereas conventional tree algorithms learn axis-parallel decision boundaries, oblique trees make tests on a linear combination of attributes. The resulting trees are more expressive but have a larger search space. See Bot and Langdon (2000).

Evolving Individual Nodes in Decision Trees

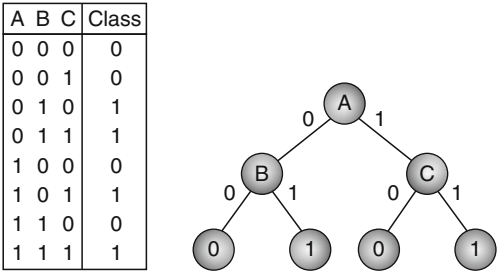
In most GP-based tree evolvers, an individual is a complete tree but in Marmelstein and Lamont (1998) each individual is a tree node. The tree is built incrementally: one GP run is made for each node. This is similar to IRL in Sect. 2.2 but results are added to a tree structure rather than a list.

3.2.3 Extensions to GP

Ensemble Methods and GP

Ensemble ideas have been used in two ways. First, to reduce fitness computation time and memory requirements by training on subsamples of the data. The bagging approach has been used in Folino et al. (2003) and Iba (1999) and the boosting approach in Song et al. (2005).

Fig. 6 Two representations of the 3 multiplexer function: truth table (left) and decision tree (right).



Although not an ensemble technique, the limited error fitness (LEF) method introduced in Gathercole and Ross (1997) as a way of reducing GP run-time works in a similar manner: in LEF, the proportion of the training set used to evaluate fitness depends on the individual's performance. The second ensemble approach improves accuracy by building an ensemble of GP trees. In Keijzer and Babovic (2000) and Paris et al. (2001) each run adds one tree to the ensemble and weights are computed with standard boosting.

### GP Hyperheuristics

Schmidhuber (1987) proposed a meta-GP system evolving evolutionary operators as a way of expanding the power of GP's evolutionary search. Instead of evolving decision rules Krasnogor proposes applying GP to the much harder task of evolving classification algorithms, represented using grammars (Krasnogor 2002, 2004; Krasnogor and Gustafson 2004). Freitas (2002a, Sect. 12.2.3) sketches a similar approach, which he calls *algorithm induction*, while (Pappa and Freitas 2010) goes into the subject in much more detail. Burke et al. (2009) also deal with GP hyperheuristics.

## 3.2.4 Conclusions

### Lack of Test Sets in GP

GP terminology follows a convention in the GA field since at least (Holland 1986) in which *brittleness* refers to overfitting or poor generalization to unseen cases, and *robustness* refers to good generalization. A feature of the GP literature is that GP is usually evaluated only on the training set (Kushchu 2002; Vanneschi and Poli 2012). Kushchu has also criticized the way in which test sets have been used (Kushchu 2002). Nonetheless GP has the same need for test sets to evaluate generalization as other methods (Kuschu 2002) and as a result, the ability of GP to perform inductive generalization is one of the open issues for GP identified in Vanneschi and Poli (2012). See Kushchu (2002) and Vanneschi and Poli (2012) for methods which have been used to encourage generalization in GP, many of which can be applied to other methods.

### Reading

See Koza's 1994 book (Koza 1994) for the basics of evolving decision trees with GP, Wong and Leung's 2000 book on data mining with grammar-based GP (Wong and Leung 2000), Freitas' 2002 book (Freitas 2002a) for a good introduction to GP, decision trees and both evolutionary and non-evolutionary learning, Poli, Langdon, and McPhee's free 2008 GP book (Poli et al. 2008), and Vanneschi and Poli's chapter on GP in this volume (Vanneschi and Poli 2012). The GP bibliography has over 5,000 entries (Langdon et al. 2009).

## 3.3 Evolving Ensembles

Ensembles, also called *multiple classifier systems* and *committee machines*, is the field which studies how to combine predictions from multiple sources. Ensemble methods are widely applicable to evolutionary systems where a population intrinsically provides multiple predictors. Ensemble techniques can be used with any learning method, although they are most useful for unstable learners, whose hypotheses are sensitive to small changes in their training. Ensembles can be heterogeneous (composed of different types of predictors) in which case they are called *hybrid* ensembles. Relatively few studies of hybrid systems exist (Brown et al. 2005)

but see for example Woods et al. (1997), Cho and Park (2001), and Chandra and Yao (2006). Ensembles enjoy good theoretical foundations (Brown et al. 1996; Tumer and Ghosh 1996), perform very well in practice (Caruana and Niculescu-Mizil 2006) and were identified by Dietterich as one of the four current directions for machine learning in 1998 (Dietterich 1998). While the key advantage of using ensembles is better test-set generalization, there are others: ensembles can perform more complex tasks than individual members, the overall system can be easier to understand and modify and ensembles are more robust/degrade more gracefully than individual predictors (Sharkey 1996).

Working with an ensemble raises a number of issues. How to create or select ensemble members? How many members are needed? When to remove ensemble members? How to combine their predictions? How to encourage diversity in members? There are many approaches to these issues, among the best known of which are bagging (Breiman 1996, 1998) and boosting (Chandra and Yao 2006a; Meir and Rätsch 2003).

Creating a good ensemble is an inherently multi-objective problem (Valentini and Masulli 2002). In addition to maximizing accuracy we also want to maximize diversity in errors; after all, having multiple identical predictors provides no advantage. On the other hand, an ensemble of predictors which make different errors is very useful since we can combine their predictions so that the ensemble output is at least as good on the training set as the average predictor (Krogh and Vedelsby 1995). Hence we want to create accurate predictors with diverse errors (Dietterich 1998; Hansen and Salamon 1990; Krogh and Vedelsby 1995; Opitz and Maclin 1999; Opitz and Slavlik 1996). In addition we may want to minimize ensemble size in order to reduce run-time and to make the ensemble easier to understand. Finally, evolving variable-length chromosomes without pressure toward parsimony results in bloat (Poli et al. 2008), in which case we have a reason to minimize the size of individual members.

### 3.3.1 Evolutionary Ensembles

Although most ensembles are non-evolutionary, evolution has many applications within ensembles. (i) *Classifier creation and adaptation*: providing the ensemble with a set of candidate members. (ii) *Voting*: Lam and Suen (1995), Thompson (1998, 1999), and Cho (1999) evolve weights for the votes of ensemble members. (iii) *Classifier selection*: the winners of evolutionary competition are added to the ensemble. (iv) *Feature selection*: generating diverse classifiers by training them on different features (see ▶ Sect. 3.1 and Kuncheva (2004, Sect. 8.1.4)). (v) *Data selection*: generating diverse classifiers by training on different data (see ▶ Sect. 3.1). All these approaches have non-evolutionary alternatives. We now go into more detail on two of the above applications.

#### Classifier Creation and Adaptation

Single-objective evolution is common in evolving ensembles. For example, Liu and Yao (1999) combines accuracy and diversity into a single objective. In comparison, multiobjective evolutionary ensembles are rare (Chandra and Yao 2006) but they are starting to appear for example Abbass (2003) and Chandra and Yao (2006a, b). In addition to upgrading GBML to multi-objective GBML, other measures can be taken to evolve diversity, for example, fitness sharing (Liu et al. 2000) and the coevolutionary fitness method we describe next. Gagné et al. (2007) compare boosting and coevolution of learners and problems – both gradually focus on



cases which are harder to learn – and argue that coevolution is less likely to overfit noise. Their coevolution-inspired fitness works as follows: Let  $Q$  be a set of reference classifiers. The hardness of a training input,  $x_i$ , is based on how many members of  $Q$  misclassify it. The fitness of a classifier is the sum of hardnesses of the inputs,  $x_i$ , it classifies correctly. This method results in accurate, yet error-diverse classifiers, since both are required to obtain high fitness. Gagné et al. exploit the population of classifiers to provide  $Q$ . They also introduce a greedy margin-based scheme for selection of ensemble members. They find that a simpler off-line version of their evolving ensemble learning (EEL) approach dominates their online version as the latter lacks a way to remove bad classifiers. Good results were obtained compared to Adaboost on six UCI (Asuncion and Newman 2009) datasets.

### Evolutionary Selection of Members

There are two extremes. Usually each run produces one member of the ensemble and many runs are needed. Sometimes, however, the entire population is eligible to join the ensemble, in which case only one run is needed. The latter does not resolve the *ensemble selection problem*: which candidates to use? There are many combinations possible from just a small pool of candidates, and, as with selecting a solution set from a Michigan population, the set of best individuals may not be the best set of individuals (that is, the best ensemble). The selection problem is formally equivalent to the feature selection problem (Gagné et al. 2007) (► Sect. 3.2). See, for example, Sirlantzis et al. (2001) and Ruta and Gabrys (2001) for evolutionary approaches.

## 3.3.2 Conclusions

Research directions for evolutionary ensembles include multiobjective evolution (Chandra and Yao 2006a), hybrid ensembles (Chandra and Yao 2006b), and minimizing ensemble complexity (Liu et al. 2000).

### Reading

Key works include Opitz and Shavlik's classic 1996 paper on evolving NN ensembles (Opitz and Shavlik (1996), Kuncheva's 2004 book on ensembles (Kuncheva 2004), Chandra and Yao's 2006 discussion of multiobjective evolution of ensembles (Chandra and Yao 2006a), Yao and Islam's 2008 review of evolving NN ensembles (Yao and Islam 2008) and Brown's 2005 and 2010 surveys of ensembles (Brown et al. 2005; Brown 2010). We cover evolving NN ensembles in ► Sect. 3.4.

## 3.4 Evolving Neural Networks

The study of neural networks (NNs) is a large and interdisciplinary area. The term artificial neural network (ANN) is often used to distinguish simulations from biological NNs, but having noted this we shall refer simply to NNs. When evolution is involved such systems may be called evolving artificial neural networks (EANNs) Yao (1999) or evolving connectionist systems (ECoSs) (Kasabov 2007).

A neural network consists of a set of nodes, a set of directed connections between a subset of nodes, and a set of weights on the connections. The connections specify inputs and outputs to and from nodes and there are three forms of nodes: input nodes (for input to the network from the outside world), output nodes, and hidden nodes, which only connect to other nodes.

Nodes are typically arranged in layers: the input layer, hidden layer(s), and output layer. Nodes compute by integrating their inputs using an activation function and passing on their activation as output. Connection weights modulate the activation they pass on and in the simplest form of learning weights are modified while all else remains fixed. The most common approach to learning weights is to use a gradient descent-based learning rule such as back-propagation. The *architecture* of a NN refers to the set of nodes, connections, activation functions, and the plasticity of nodes (that is, whether they can be updated or not). Most often all nodes use the same activation function and in virtually all cases all nodes can be updated. Evolution has been applied at three levels: weights, architecture, and learning rules. In terms of architecture, evolution has been used to determine connectivity, select activation functions, and determine plasticity.

### Representations

Three forms of representations have been used: (i) direct encoding (Yao 1999; Floreano et al. 2008) in which all details (connections and nodes) are specified, (ii) indirect encoding (Yao 1999; Floreano et al. 2008) in which general features are specified (e.g., number of hidden layers and nodes) and a learning process determines the details, and (iii) developmental encoding (Floreano et al. 2008) in which a developmental process is genetically encoded (Kitano 1990; Gruau 1995; Nolfi et al. 1994; Husbands et al. 1994; Pal and Bhandari 1994; Sziranyi 1996). Implicit and developmental representations are more flexible and tend to be used for evolving architectures, while direct representations tend to be used for evolving weights alone.

### Credit Assignment

Evolving NNs virtually always use the Pittsburgh approach although there are a few Michigan systems (Andersen and Tsoi 1993; Smith and Cribbs 1994; Smith and Cribbs 1997). In Michigan systems, each chromosome specifies only one hidden node, which raises issues. How should the architecture be defined? A simple method is to fix it in advance. How can we make nodes specialize? Two options are to encourage diversity during evolution, for example with fitness sharing, or, after evolution, by pruning redundant nodes (Andersen and Tsoi 1993).

### Adapting Weights

Most NN learning rules are based on gradient descent, including the best known: back-propagation (BP). BP has many successful applications, but gradient descent-based methods require a continuous and differentiable error function and often get trapped in local minima (Sutton 1986; Whitley et al. 1990).

An alternative is to evolve the weights which has the advantages that EAs do not rely on gradients and can work on discrete fitness functions. Another advantage of evolving weights is that the same evolutionary method can be used for different types of network (feedforward, recurrent, and higher order), which is a great convenience for the engineer (Yao 1999). Consequently, much research has been done on evolution of weights. Unsurprisingly fitness functions penalize NN error but they also typically penalize network complexity (number of hidden nodes) in order to control overfitting. The expressive power of a NN depends on the number of hidden nodes: fewer nodes = less expressive = fits training data less, while more nodes = more expressive = fits data better. As a result, if a NN has too few nodes it underfits, while with too many nodes it overfits. In terms of training rate, there is no clear winner between evolution and gradient descent; which is better depending on the problem

(Yao 1999). However, Yao (1999) states that evolving weights and architecture is better than evolving weights alone and that evolution seems better for reinforcement learning and recurrent networks. Floreano (2008) suggests evolution is better for dynamic networks. Happily we do not have to choose between the two approaches.

### **Evolving and Learning Weights**

Evolution is good at finding a good basin of attraction but poor at finding the optimum of the basin. In contrast, gradient descent has the opposite characteristics. To get the best of both (Yao 1999) we should evolve initial weights and then train them with gradient descent. Floreano (2008) claims that this can be two orders of magnitude faster than beginning with random initial weights.

### **Evolving Architectures**

Architecture has an important impact on performance and can determine whether a NN under- or over-fits. Designing architectures by hand is a tedious, expert, trial-and-error process. Alternatives include constructive NNs, which grow from a minimal network and destructive NNs, which shrink from a maximal network. Unfortunately, both can become stuck in local optima and can only generate certain architectures (Angeline et al. 1994). Another alternative is to evolve architectures. Miller et al. (1989) make the following suggestions (quoted from Yao (1999)) as to why EAs should be suitable for searching the space of architectures.

- ▶ The surface is infinitely large since the number of possible nodes and connections is unbounded.
- ▶ The surface is nondifferentiable since changes in the number of nodes or connections are discrete and can have a discontinuous effect on EANN's performance.
- ▶ The surface is complex and noisy, since the mapping from an architecture to its performance is indirect, strongly epistatic, and dependent on the evaluation method used.
- ▶ The surface is deceptive since similar architectures may have quite different performance.
- ▶ The surface is multimodal since different architectures may have similar performance.

There are good reasons to evolve architectures and weights simultaneously. If we learn with gradient descent there is a many-to-one mapping from NN genotypes to phenotypes (Yao and Liu 1997). Random initial weights and stochastic learning lead to different outcomes, which makes fitness evaluation noisy, and necessitates averaging over multiple runs, which means the process is slow. On the other hand, if we evolve architectures and weights simultaneously we have a one-to-one genotype to phenotype mapping, which avoids the problem above and results in faster learning. Furthermore, we can co-optimize other parameters of the network (Floreano 2008) at the same time. For example, Belew et al. (1992) found the best networks had a very high learning rate which may have been optimal due to many factors such as initial weights, training order, and amount of training. Without co-optimizing, architecture and weights evolution would not have been able to take all factors into account at the same time.

### **Evolving Learning Rules (Yao 1999)**

There is no one best learning rule for all architectures or problems. Selecting rules by hand is difficult and if we evolve the architecture then we do not know a priori what it will be.

A way to deal with this is to evolve the learning rule, but we must be careful: the architectures and problems used in learning the rules must be representative of those to which it will eventually be applied. To get general rules, we should train on general problems and architectures, not just one kind. On the other hand, to obtain a training rule specialized for a specific architecture or problem type, we should train just on that architecture or problem.

One approach is to evolve only learning rule parameters (Yao 1999) such as the learning rate and momentum in backpropagation. This has the effect of adapting a standard learning rule to the architecture or problem at hand. Non-evolutionary methods of adapting training rules also exist. Castillo et al. (2007), working with multi-layer perceptrons, found evolving the architecture, initial weights, and rule parameters together as good or better than evolving only the first two or the third.

We can also evolve new learning rules (Yao 1999; Radi and Poli 2003). Open-ended evolution of rules was initially considered impractical and instead Chalmers (1990) specified a generic, abstract form of update and evolved its parameters to produce different concrete rules. The generic update was a linear function of ten terms, each of which had an associated evolved real-valued weight. Four of the terms represented local information for the node being updated while the other six terms were the pairwise products of the first four. Using this method Chalmers was able to rediscover the delta rule and some of its variants. This approach has been used by a number of others and has been reported to outperform human-designed rules (Dasdan and Oflazer 1993). More recently, GP was used to evolve novel types of rules from a set of mathematical functions and the best new rules consistently outperformed standard backpropagation (Radi and Poli 2003). Whereas architectures are fixed, rules could potentially change over their lifetime (e.g., their learning rate could change) but evolving dynamic rules would naturally be much more complex than evolving static ones.

### 3.4.1 Ensembles of NNs

Most methods output a single NN (Yao and Islam 2008) but a population of evolving NNs is naturally treated as an ensemble and recent work has begun to do so. Evolving NNs is inherently multiobjective: we want accurate yet simple and diverse networks. Some works combine these objectives into one fitness function while others are explicitly multiobjective.

#### Single-Objective Ensembles

Yao and Liu (1998) used EPNet's (Yao and Liu 1997) population as an ensemble without modifying the evolutionary process. By treating the population as an ensemble, the result outperformed the population's best individual. Liu and Yao (1990) pursued accuracy and diversity in two ways. The first was to modify backpropagation to minimize error and maximize diversity using an approach they call negative correlation learning (NCL) in which the errors of members become negatively correlated and hence diverse. The second method was to combine accuracy and diversity in a single objective. Evolutionary ensembles for NCL (EENCL) (Liu et al. 2000) automatically determines the size of an ensemble. It encourages diversity with fitness sharing and NCL, and it deals with the ensemble member selection problem (➤ Sect. 3.3.1) with a cluster-and-select method (see Jin and Sendhoff (2004)). First we cluster candidates, based on their errors, on the training set so that clusters of

candidates make similar errors. Then we select the most accurate in each cluster to join the ensemble; the result is the ensemble can be much smaller than the population. Cooperative neural net ensembles (CNNE) (Islam et al. 2003) used a constructive approach to determine the number of individuals and how many hidden nodes each has. Both contribute to the expressive power of the ensemble and CNNE was able to balance the two to obtain suitable ensembles. Unsurprisingly, it was found that more complex problems needed larger ensembles.

### Multiobjective Ensembles

Memetic pareto artificial NN (MPANN) (Abbass 2003) was the first ensemble of NNs to use multiobjective evolution. It also uses gradient-based local search to optimize network complexity and error. Diverse and accurate ensembles (DIVACE) (Chandra and Yao 2006a) uses multi-objective evolution to maximize accuracy and diversity. Evolutionary selection is based on non-dominated sorting (Srinivas and Deb 1994), a cluster-and-select approach is used to form the ensemble, and search is provided by simulated annealing and a variant of differential evolution (Storn and Price 1996). DIVACE-II (Chandra and Yao 2006) is a heterogeneous multi-objective Michigan approach using NNs, support vector machines, and radial basis function nets. The role of crossover and mutation is played by bagging (Breiman 1996) and boosting (Freund and Schapire 1996), which produce accurate and diverse candidates. Each generation bagging and boosting make candidate ensemble members and only dominated members are replaced. The accuracy of DIVACE-II was very good compared to 25 other learners on the Australian credit card and diabetes datasets and it outperformed the original DIVACE.

### 3.4.2 Yao's Framework for Evolving NNs

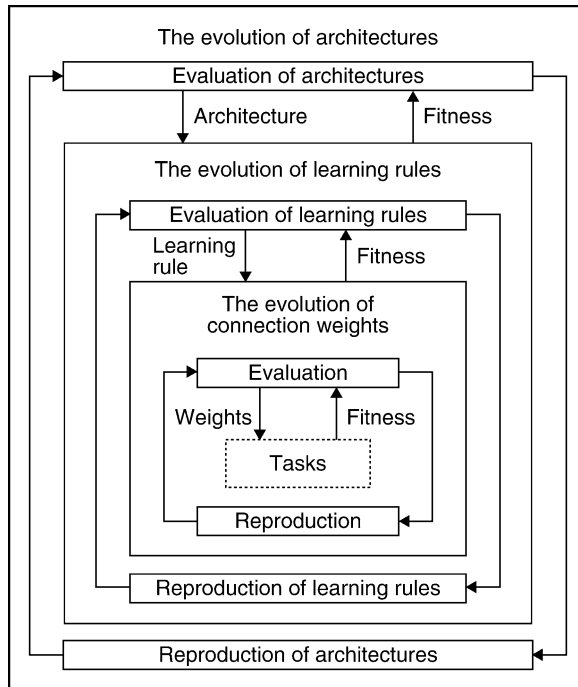
Figure 7 shows Yao's framework for evolving architectures, training rules, and weights as nested processes (Yao 1999). Weight evolution is the innermost as it occurs at the fastest time scale, while either rule or architecture evolution is outermost. If we have prior knowledge, or are interested in a specific class of either rule or architecture, this constrains the search space and Yao suggests the outermost should be the one which constrains it most. The framework can be thought of as a three-dimensional space of evolutionary NNs where 0 on each axis represents one-shot search and infinity represents exhaustive search. If we remove references to EAs and NNs, it becomes a general framework for adaptive systems.

### 3.4.3 Conclusions

Evolution is widely used with NNs, indeed according to Floreano et al. (2008) most studies of neural robots in real environments use some form of evolution. Floreano et al. go on to claim that evolving NNs can be used to study "brain development and dynamics because it can encompass multiple temporal and spatial scales along which an organism evolves, such as genetic, developmental, learning, and behavioral phenomena. The possibility to coevolve both the neural system and the morphological properties of agents . . . adds an additional valuable perspective to the evolutionary approach that cannot be matched by any other approach." (Floreano et al. 2008, p. 59).

■ Fig. 7

Yao's framework for evolving architectures, training rules, and weights.



### Reading

Key reading on evolving NNs includes Yao's classic 1999 survey (Yao 1999), Kasabov 2007 book (Kasabov 2007), Floreano, Dürr, and Mattiussi's 2008 survey (Floreano et al. 2008), which includes reviews of evolving dynamic and neuromodulatory NNs, and Yao and Islam's 2008 survey of evolving NN ensembles (Yao and Islam 2008).

## 3.5 Learning Classifier Systems (LCS)

Learning classifier systems (LCS) originated in the GA community as a way of applying GAs to learning problems. The LCS field is one of the oldest, largest, and most active areas of GBML. The majority of LCS research is currently carried out on XCS (Wilson 1995; Butz and Wilson 2001) and its derivatives XCSF (Wilson 2001b, 2002a) for regression/function approximation, and UCS (Bernadó-Mansilla and Garrell-Guiu 2003; Orriols-Puig and Bernadó-Mansilla 2008) for supervised learning.

### The Game of the Name

Terminology has been contentious in this area (Heitkötter and Beasley 2001). LCS are also widely simply called classifier systems (abbreviated CS or CFS) and sometimes evolutionary (learning) classifier systems. At one time GBML referred exclusively to LCS. None of these names is very satisfactory but the field appears to have settled on LCS.

The difficulty in naming the field relates in part to the difficulty in defining what an LCS is (Smith 1992; Holland et al. 2000). In practice, what is accepted as an LCS has become more inclusive over the years. A reasonable definition of an LCS would be an evolutionary rule-based system – except that a significant minority of LCS are not evolutionary! On the other hand, most non-evolutionary rule-based systems are not considered LCS, so the boundaries of the field are defined more by convention than principle. Even EA practitioners are far from unanimous; work continues to be published which some would definitely consider forms of LCS, but which make no reference to the term and which contain few or no LCS references.

(L)CS has at times been taken to refer to Michigan systems only (see, e.g., Greene and Smith (1993)) but it now generally includes Pitt systems as well, as implied by the name and content of the international workshop on learning classifier systems (IWLCS) which includes both Pitt and Michigan, evolutionary and non-evolutionary systems. As a final terminological note, rules in LCS are often referred to as “classifiers”.

### 3.5.1 Production Systems and Rule(Set) Parameters

LCS evolve condition-action (IF–THEN) rules. Recall from ● Sect. 2.2 and ● Fig. 2 that in Michigan rule-based systems a chromosome is a single rule, while in Pittsburgh systems a chromosome is a variable-length *set* of rules. Pittsburgh, Michigan, IRL, and GCCL are all used. Michigan systems are rare elsewhere but are the most common form of LCS. Within LCS, IRL is most common with fuzzy systems, but see Aguilar-Ruiz et al. (2003) for a non-fuzzy version. In LCS, we typically evolve rule conditions and actions although non-evolutionary operators may act upon them. In addition, each phenotype has parameters associated with it and these parameters are typically learned rather than evolved using the Widrow–Hoff update or similar (see Lanzi et al. (2006b) for examples). In Michigan LCS parameters are associated with each rule but in Pittsburgh systems they are associated with each ruleset. For example, in UCS the parameters are: fitness, mean action set size (to bias a deletion process, which seeks to balance action set sizes), and experience (a count of the number of times a rule has been applied, in order to estimate confidence in its fitness). In GAssist (a supervised Pittsburgh system) the only parameter is fitness. Variations of the above exist; in some cases, rules predict the next input or read and write to memory.

### 3.5.2 Representing Conditions and Actions

The most common representation in LCS uses fixed-length strings with binary inputs and outputs and ternary conditions. In a simple Michigan version (see, e.g., Wilson (1995)), each rule has one action and one condition from  $\{0, 1, \#\}$  where  $\#$  is a wildcard, matching both 0 and 1 in inputs. For example, the condition 01 $\#$  matches two inputs: 010 and 011. Similar representations were used almost exclusively prior to approximately 2000 and are inherited from GAs and their preference for minimal alphabets. (Indeed, ternary conditions have an interesting parallel with ternary schemata (Reeves and Rowe 2002) for binary GAs.) Such rules individually have limited expressive power (Schuurmans and Schaeffer 1989) (but see also Booker (1991)), which necessitates that solutions are sets of rules. More insidiously, the lack of individual expressiveness can be a factor in pathological credit assignment (strong/fit overgenerals

(Kovacs 2004)). Various extensions to the simple scheme described above have been studied (see (Kovacs 2004, Sect. 2.2.2)).

Real-Valued Intervals

Following Bacardit (2004, p. 84) we distinguish two approaches to real-valued interval representation in conditions. The first is representations based on discretization: HIDER\* uses *natural coding* (Giraldez et al. 2003), ECL clusters attribute values and evolves constraints on them (Divina et al. 2003) while GAssist uses *adaptive discretization intervals* (Bacardit 2004). The second approach is to handle real values directly. In HIDER genes specify a lower and upper bound (where lower is always less than upper) (Aguilar-Ruiz et al. 2003). In Corcoran and Sen (1994) a variation of HIDER’s scheme is used where the attribute is ignored when the upper bound is less than the lower. Interval representations are also used in Wilson (2001a) and Stone and Bull (2003). Finally, Wilson (2000) specifies bounds using centre and spread genes.

Default/Exception Structures

Various forms of default/exception rule structures have been used with LCS. It has been argued that they should increase the number of solutions possible without increasing the search space and should allow gradual refinement of knowledge by adding exceptions (Holland 1986). However, the space of *combinations* of rules is much larger than the set of rules and the evolutionary dynamics of default/exception rule combinations have proved difficult to manage in Michigan systems. Nonetheless, default rules can significantly reduce the number of rules needed for a solution (Valenzuela-Rendón 1989) and there have been some successes.

Figure 8 illustrates three representations for a Boolean function. The leftmost is a truth table, which lists all possible inputs and their outputs. The middle representation is the ternary language commonly used by LCS, which requires only four rules to represent the eight input/output pairs, thanks to the generalization provided by the # symbol. Finally, on the right, a default rule (### → 1) has been added to the ternary representation. This rule matches all inputs and states that the output is always 1. This rule is incorrect by itself, but the two rules above it provide exceptions and, taken together, the three accurately represent the function using one less rule than the middle representation. One difficulty in evolving such

Fig. 8  
Three representations for the 3 multiplexer function.

Truth table							
A	B	C	Output	Ternary rules		Default rule	
0	0	0	0	0	0	#	→ 0
0	0	1	0	0	1	#	→ 1
0	1	0	1	1	#	0	→ 0
0	1	1	1	1	#	1	→ 1
1	0	0	0				
1	0	1	1				
1	1	0	0				
1	1	1	1				



default/exception structures lies in identifying which rules are the defaults and which the exceptions; a simple solution is to maintain the population in some order and make earlier rules exceptions to later ones (as in a decision list (Rivest 1987)). This is straightforward in Pitt systems in which individual rulesets are static but is more complex in Michigan populations in which individual rules are created and deleted dynamically. The other issue is how to assign credit to the overall multi-rule structure. In Pittsburgh systems this is again straightforward since fitness is assigned only at the level of rulesets, but in Michigan systems each rule has a fitness, and it is not obvious how to credit the three rules in the default/exception structure in a way which recognizes their cooperation.

The Pittsburgh GABIL (De Jong and Spears 1991) and GAssist (Bacardit 2004) use decision lists and often evolve default rules spontaneously (e.g., a fully general last rule). Bacardit found that enforcing a fully general last rule in each ruleset in GAssist (and allowing evolution to select the most useful class for such rules) was effective (Bacardit 2004).

In Michigan systems, default/exception structures are called default hierarchies. Rule specificity has been used as the criterion for determining which rules are exceptions and accordingly conflict resolution methods have been biased according to specificity. There are, however, many problems with this approach (Smith and Goldberg 1991). It is difficult for evolution to produce these structures since they depend on cooperation between otherwise competing rules. The structures are unstable since they are interdependent; unfortunate deletion of one member alters the utility of the entire structure. As noted, they complicate credit assignment and conflict resolution since exception rules must override defaults (Wilson 1989; Smith and Goldberg 1991). There are also problems with the use of specificity to prioritize rules. For one, having fewer #s does not mean a rule actually matches fewer inputs; counting #s is a purely syntactic measure of generality. For another, there is no reason why exception rules should be more specific. The consequence of these difficulties is that there has not been much interest in Michigan default hierarchies since the early 1990s (but see Vallim et al. (2003)) and indeed not all Michigan LCS support them (e.g., ZCS (Wilson 1995), XCS/XCSF and UCS do not). Nonetheless, the idea should perhaps be revisited and an ensembles perspective might prove useful.

### Other Representations for Conditions and Actions

A great range of other representations has been used, particularly in recent years. These include  $VL_1$  logic (Michalski et al. 1986) as used in GIL (Janikow 1993), first-order logic (Mellor 2005a, b, 2008), decision lists as used in GABIL (De Jong and Spears 1991) and GAssist (Bacardit 2004), messy encoding (Lanzi 1999a), ellipses (Butz 2005) and hyperellipses (Butz et al. 2006), hyperspheres (Marshall and Kovacs 2006), convex hulls (Lanzi and Wilson 2006a), tile coding (Lanzi et al. 2006) and a closely related hyperplane coding (Booker 2005a, b), GP trees (Ahluwalia and Bull 1999; Lanzi 1999b, 2001), Boolean networks defined by GP (Bull 2009), support vectors (Loiacono et al. 2007), edges of an augmented transition network (Landau et al. 2005), gene expression programming (Wilson 2008), fuzzy rules (see ▶ Sect. 3.6), and neural networks (Smith and Cribbs 1994; Cribbs and Smith 1996; Smith and Cribbs 1997; Bull and O'Hara 2002; O'Hara and Bull 2005; Dam et al. 2008; Howard et al. 2008; Howard and Bull 2008). GALE (Llorá and Garrell 2001; Llorá 2002; Llorá and Wilson 2004) has used particularly complex representations, including the use of GP to evolve trees defining axis-parallel and oblique hyper-rectangles (Llorá and Wilson 2004), and evolved prototypes, which are used with a k-nearest-neighbor classifier. The prototypes need not be fully specified; some attributes can be left undefined. This representation has also been used in

GAssist (Bacardit 2004). There has been limited work with alternative action representations including computed actions (Tran et al. 2007; Lanzi and Loiacono 2007) and continuous actions (Wilson 2007).

### 3.5.3 Evolutionary Selection of Representations

As we have seen, there are many representations to choose from. Unfortunately, it is generally not clear which might be best for a given problem or part of a problem. One approach is to let evolution make these choices. This can be seen as a form of meta-learning in which evolution adapts the inductive bias of the learner. In Bacardit (2004) and Bacardit et al. (2007), evolution was used to select default actions in decision lists in GAssist. GAssist's initial population was seeded with last rules, which together advocated all possible classes and over time evolution selected the most suitable of these rules. To obtain good results, it was necessary to encourage diversity in default actions. In GALE (Llorá 2002), evolution selects both classification algorithms and representations. GALE has elements of cellular automata and artificial life: individuals are distributed on a two-dimensional grid. Only neighbors within  $r$  cells interact: two neighbors may perform crossover, an individual may be cloned and copied to a neighboring cell, and an individual may die if its neighbors are fitter. A number of representations have been used: rule sets, prototypes, and decision trees (orthogonal, oblique, and multivariate based on nearest neighbor). Decision trees are evolved using GP while prototypes are used by a  $k$ -nearest-neighbor algorithm to select outputs. An individual uses a particular representation and classification algorithm and hence evolutionary selection operates on both. Populations may be homogeneous or heterogeneous and in Llorá and Wilson (2004) GALE was modified to interbreed orthogonal and oblique trees.


In the representational ecology approach (Marshall and Kovacs 2006) condition shapes were selected by evolution. Two Boolean classification tasks were used: a plane function, which is easy to describe with hyperplanes, but hard with hyperspheres, and a sphere function, which has the opposite characteristics. Three versions of XCS were used: with hyperplane conditions (XCS-planes), with hyperspheres (XCS-spheres), and both (XCS-both). XCS was otherwise unchanged, but in XCS-both the representations compete due to XCS's pressure against overlapping rules (Kovacs 2004). In XCS-both planes and spheres do not interbreed; they constitute genetically independent populations, that is, species. In terms of classification accuracy XCS-planes did well on the plane function and poorly on the sphere function while XCS-sphere showed the opposite results. XCS-both performed well on both; there was no significant difference in accuracy compared to the better single-representation version on each problem. Furthermore, XCS-both selected the more appropriate representation for each function. In terms of the amount of training needed, XCS-both was similar to XCS-sphere on the sphere function but was significantly slower than XCS-plane on the plane function.

#### Selecting Discretization Methods and Cut Points


GAssist's Adaptive Discretization Intervals (ADI) approach has two parts (Bacardit 2004). The first consists of adapting interval sizes. To begin, a discretization algorithm proposes cut points for each attribute and this defines the finest discretization possible, said to be composed of micro-intervals. Evolution can merge and split macro-intervals, which are composed of micro-intervals, and each individual can have different macro-intervals. The second part consists of selecting discretization algorithms. Evolution is allowed to select discretization

algorithms for each attribute or rule from a pool including uniform width, uniform frequency, ID3, Fayyad and Irani, Mântaras, USD, ChiMerge, and random. Unfortunately, evolving the best discretizers was found to be difficult and the use of ADI resulted in only small improvements in accuracy. However, further work was suggested.

### 3.5.4 Optimization of Population Representation: Macroclassifiers

Wilson (1995) introduced an optimization for Michigan populations called macroclassifiers. He noted that as an XCS population converges on the solution set, many identical copies of this set accumulate. A macroclassifier is simply a rule with an additional *numerosity* parameter, which indicates how many identical virtual rules it represents. Using macroclassifiers saves a great deal of run-time compared to processing a large number of identical rules. Furthermore, macroclassifiers provide interesting statistics on evolutionary dynamics. Empirically, macroclassifiers perform essentially as the equivalent “micro” classifiers (Kovacs 1996).  Figure 9 illustrates how the rules *m* and *m'* in the top can be represented by *m* alone in the bottom by adding a numerosity parameter.

### 3.5.5 Rule Discovery

LCS are interesting from an evolutionary perspective, particularly Michigan systems in which evolutionary dynamics are more complex than in Pittsburgh systems. Where Pittsburgh systems face two objectives (evolving accurate and parsimonious rulesets) Michigan systems face a third: coverage of the input (or input/output) space. Furthermore, Michigan systems have coevolutionary dynamics as rules both cooperate and compete. Since the level of selection (rules) is lower than the level of solutions (rulesets) researchers have attempted to coax better results by modifying rule fitness calculations with various methods. Fitness sharing and crowding have been used to encourage diversity and, hence, coverage. Fitness sharing is naturally based on inputs (see ZCS) while crowding has been implemented by making deletion probability proportional to the degree of overlap with other rules (as in XCS). Finally, restricted mating as implemented by a niche GA plays an important role in XCS and UCS (see  Sect. 3.5.5).

#### Fig. 9

A population of microclassifiers (*top*) and the equivalent macroclassifiers (*bottom*).

Rule	Cond.	Action	Strength
<i>m</i>	# # 0 0 1 1	1	200.0
<i>m'</i>	# # 0 0 1 1	1	220.0
<i>n</i>	# # 0 0 1 1	0	100.0
<i>o</i>	0 0 1 1 1 0	1	100.0

Rule	Cond.	Action	Strength	Numerosity
<i>m</i>	# # 0 0 1 1	1	200.0	2
<i>n</i>	# # 0 0 1 1	0	100.0	1
<i>o</i>	0 0 1 1 1 0	1	100.0	1

### Windowing in Pittsburgh LSS

As noted in ► Sect. 2.2 naive implementations of the Pittsburgh approach are slower than Michigan systems, which are themselves slow compared to non-evolutionary methods. The naive Pitt approach is to evaluate each individual on the entire data set, but much can be done to improve this. Instead, windowing methods (Förnkrantz 1998) learn on subsets of the data to improve runtime. Windowing has been used in Pitt LCS since, at least, ADAM (Greene and Smith 1987). More recently GAssist used incremental learning by alternating strata (ILAS) (Bacardit 2004) which partitions the data into  $n$  strata, each with the same class distribution as the entire set. A different stratum is used for fitness evaluation each generation. On larger data sets speed-up can be an order of magnitude. Windowing has become a standard part of recent Pittsburgh systems applied to real-world problems (e.g., Bacardit and Krasnogor (2008) and Bacardit et al. (2008, 2009)).

Many ensemble methods improve classification accuracy by sampling data in similar ways to windowing techniques, which suggests the potential for both improved accuracy and runtime, but this has not been investigated in LCS.

### Michigan Rule Discovery

Most rule discovery work focuses on Michigan LCS as they are more common and their evolutionary dynamics are more complex. The rest of this section deals with Michigan systems although many ideas, such as self-adaptive mutation, could be applied to Pitt systems. Michigan LCS use the steady state GAs introduced in ► Sect. 2.4 as they minimize disruption to the rule population during on-line learning. An unusual feature of Michigan LCS is the emphasis placed on minimizing population size, for which various techniques are used: niche GAs, the addition of a generalization term in fitness, subsumption deletion, condensation and various compaction methods.

### Niche GAs

Whereas in a standard panmictic GA all rules are eligible for reproduction, in a niche GA mating is restricted to rules in the same action set (which is considered a niche). (See ► Fig. 3 on action sets.) The input spaces of rules in an action set overlap and their actions agree, which suggests their predictions will tend to be related. Consequently, mating these related rules is more effective, on average, than mating rules drawn from the entire population. This is a form of speciation since it creates non-interbreeding sub-populations. However, the niche GA has many other effects (Wilson 2001). First, a strong bias toward general rules, since they match more inputs and hence appear in more action sets. Second, pressure against overlapping rules, since they compete for reproduction (Kovacs 2004). Third, complete coverage of the input space, since competition occurs for each input. The niche GA was introduced in Booker (1989) and originally operated in the match set but was later further restricted to the action set (Wilson 1998). It is used in XCS and UCS and is related to *universal suffrage* (Giordana and Saitta 1994).

### EDAs Instead of GAs

Recently Butz et al. (2005, 2006) and Butz and Pelikan (2006) replaced XCS's usual crossover with an estimation of distribution algorithm (EDA)-based method to improve solving of difficult hierarchical problems, while (Llorà et al. 2005a, b) introduced CCS: a Pitt LCS based on compact GAs (a simple form of EDA).

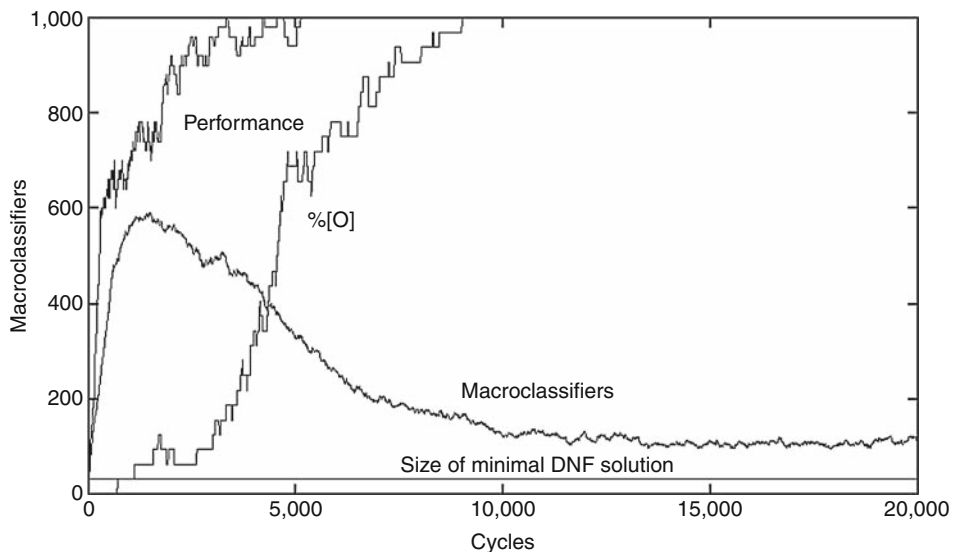
### Subsumption Deletion

A rule  $x$  logically subsumes a rule  $y$  when  $x$  matches a superset of the inputs  $y$  matches and they have the same action. For example,  $00\# \rightarrow 0$  subsumes  $000 \rightarrow 0$  and  $001 \rightarrow 0$ . In XCS  $x$  is allowed to subsume  $y$  if: (i)  $x$  logically subsumes  $y$ , (ii)  $x$  is sufficiently accurate and (iii)  $x$  is sufficiently experienced (has been evaluated sufficiently) so we can have confidence in its accuracy. Subsumption deletion was introduced in XCS (see Butz and Wilson (2001)) and takes two forms. In *GA subsumption*, when a child is created, we check to see if its parents subsume it, which constrains accurate parents to only produce more general children. In *action set subsumption*, the most general of the sufficiently accurate and experienced rules in the action set is given the opportunity to subsume the others. This removes redundant, specific rules from the population but is too aggressive for some problems.

### Michigan Evolutionary Dynamics

Michigan LCS have interesting evolutionary dynamics and plotting macroclassifiers is a useful way to monitor population convergence and parsimony. Figure 10 illustrates by showing XCS learning the 11 multiplexer function. The performance curve is a moving average of the proportion of the last 50 inputs which were classified correctly, % $[O]$  shows the proportion of the minimal set of 16 ternary rules XCS needs to represent this function (indicated by the straight line labeled “Size of minimal DNF solution” in the figure) and macroclassifiers were explained in Sect. 3.5.4. In this experiment, the population was initially empty and was seeded by covering Sect. 3.5.5. “Cycles” refers to the number of inputs presented, inputs were drawn uniform randomly from the input space, the population size limit was 800, all input/output pairs were in both the train and test sets, GA subsumption was used but action set subsumption was not and curves are the average of 10 runs. Other settings are as in Wilson (1995).

Fig. 10  
Evolutionary dynamics of XCS on the 11 multiplexer.



Note that XCS continues to refine its solution (population) after 100% performance is reached and that it finds the minimal representation (at the point where % $[O]$  reaches the top of the figure) but that continued crossover and mutation generate extra transient rules, which make the population much larger.

### Condensation

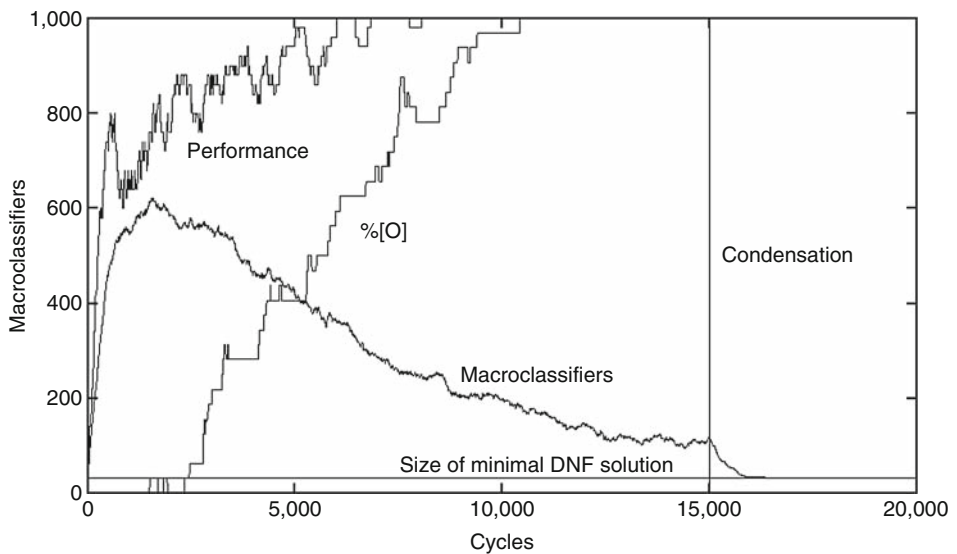
As illustrated by [Fig. 10](#), an evolved population normally contains many redundant and low-fitness rules. These rules are typically transient, but more are generated while the GA continues to run. Condensation (Wilson 1998; Kovacs 1996) is a very simple technique to remove such rules, which consists of running the system with crossover and mutation turned off; we only clone and delete existing rules. [Figure 11](#) repeats the experiment from [Fig. 10](#) but switches after 15,000 cycles to condensation after which the population quickly converges to the minimal solution. Other methods of compacting the population have been investigated (Kovacs 1997; Wilson 2002b; Dixon et al. 2002).

### Tuning Evolutionary Search

XCS is robust to class imbalances (Orriols-Puig and Bernadó-Mansilla 2006) but for very high imbalances tuning the GA based on a facetwise model improved performance (Orriols-Puig and Bernadó-Mansilla 2006; Orriols-Puig et al. 2007b). Self-tuning evolutionary search has also been studied. The mutation rate can be adapted during evolution for example Hurst and Bull (2003, 2004), Howard et al. (2008), and Butz et al. (2008), while De Jong et al. (1993) dynamically controls use of two generalization operators: each has a control bit specifying whether it can be used and control bits evolve with the rest of the genotypes.

■ Fig. 11

XCS with condensation on the 11 multiplexer.



### Non-evolutionary Rule Discovery

Evolution has been supplemented by heuristics in various ways. Covering, first suggested in Holland (1976), creates a rule to match an unmatched input. It can be used to create (“seed”) the initial population (Venturini 1993; Wilson 1995; Hekanaho 1995) or to supplement the GA throughout evolution (Wilson 1995). Kovacs (2004, p. 42) found covering each action set was preferable to covering the match set when applying XCS to sequential tasks. Most covering/seeding is done as needed but instead (Juan Liu and Tin-Yau Kwok 2000) selects inputs at the center of same-class clusters. For other non-evolutionary operators see (Booker 1989; Riolo 1987), the work on corporations of rules (Wilson and Goldberg 1989; Smith 1994; Tomlinson and Bull 1998, 2002; Tomlinson 1999) and the work on non-evolutionary LCS.

### Non-evolutionary LCS

Although LCS were originally conceived as a way of applying GAs to learning problems (Holland and Reitman 1978), not all LCS include a GA. Various heuristics have been used to create and refine rules in for example YACS (Gerard et al. 2002) and MACS (Gérard and Sigaud 2003). A number of systems have been inspired by psychological models of learning. ACS (Stolzmann 1996; Butz 2002) and ACS2 (Butz 2002a) are examples, although ACS was also later supplemented by a GA (Butz et al. 2000a, b). Another is AgentP, a specialized LCS for maze tasks (Zatuchna 2004, 2005).

## 3.5.6 LCS Credit Assignment

While credit assignment in Pittsburgh LCS is a straightforward matter of multiobjective fitness evaluation, as noted in ► Sect. 3.5.5 it is far more complex in Michigan systems with their more complex evolutionary dynamics. Credit assignment is also more complex in some learning paradigms, particularly reinforcement learning, which we will not cover here. Within supervised learning, credit assignment is more complex in regression tasks than in classification. These difficulties have been the major issue for Michigan LCS and have occupied a considerable part of the literature, particularly prior to the development of XCS, which provided a reasonable solution for both supervised and reinforcement learning.

### Strength and Accuracy in Michigan LCS

Although we are not covering reinforcement learning, Michigan LCS have traditionally been designed for such problems. XCS/XCSF are reinforcement learning systems, but since supervised learning can be formulated as simplified reinforcement learning, they have been applied to SL tasks. Consequently, we now very briefly outline the difference between the two major forms of Michigan reinforcement learning LCS.

In older (pre-1995) reinforcement learning, LCS fitness is proportional to the magnitude of reward and is called *strength*. Strength is used both for conflict resolution and as fitness in the GA (see e.g., ZCS (Wilson 1994)). Such LCS are referred to as strength-based and they suffer from many difficulties with credit assignment (Kovacs 2004), the analysis of which is quite complex. Although some strength-based systems incorporate accuracy as a component of fitness, their fitness is still proportional to reward. In contrast, the main feature of XCS is that it adds a prediction parameter, which estimates the reward to be obtained if the action advocated by a rule is taken. Rule fitness is proportional to the accuracy of reward prediction and not to its magnitude, which avoids many problems strength-based systems have with

credit assignment. In XCS, accuracy is estimated from the variance in reward and since overgeneral rules have high variance they have low fitness. Although XCS has proved robust in a range of applications, a major limitation is that the accuracy estimate conflates several things: (i) overgenerality in rules, (ii) noise in the training data, and (iii) stochasticity in the transition function in sequential problems. In contrast, strength-based systems may be less affected by noise and stochasticity since they are little affected by reward variance. See Kovacs (2004) for analysis of the two approaches.

### Prediction Updates

To update rule predictions while training, the basic XCS system (Wilson 1995; Butz and Wilson 2001) uses the Widrow–Hoff update for nonsequential problems and the Q-learning update for sequential ones. Various alternatives have been used: average rewards (Tharakannel and Goldberg 2002; Lanzi and Loiacono 2006), gradient descent (Butz et al. 2005b; Lanzi et al. 2007), and eligibility traces (Drugowitsch and Barry 2005). The basic XCSF uses NLMS (linear piecewise) prediction (Wilson 2000b, 2002a) but Lanzi et al. (2006b) has compared various alternative classical parameter estimation (RLS and Kalman filter) and gain adaptation algorithms (K1, K2, IDBD, and IDD). He found that Kalman filter and RLS have significantly better accuracy than the others and that Kalman filter produces more compact solutions than RLS. There has also been recent work on other systems; UCS is essentially a supervised version of XCS and the main difference is its prediction update. Bull has also studied simplified LCS (Bull 2005).

### Evolutionary Selection of Prediction Functions

In Lanzi et al. (2008), Lanzi selects prediction functions in XCSFHP (XCSF with Heterogeneous Predictors) in a way similar to the selection of condition types in the representational ecology approach in ♦ Sect. 3.5.3. Polynomial functions (linear, quadratic, and cubic) and constant, linear and NN predictors were available. XCSFHP selected the most suitable predictor for regression and sequential tasks and performed almost as well as XCSF using the best single predictor.

### Theoretical Results

Among the notable theoretical works on LCS, Lanzi (2002a) demonstrates that XCS without generalization implements tabular Q-learning, Butz et al. (2005a) investigate the computational complexity of XCS in a probably approximately correct (PAC) setting, and Wada et al. (2005a, b, c, 2007) analyze credit assignment and relate LCS to mainstream reinforcement learning methods. Kovacs (2004) identifies pathological rule types: strong overgeneral and fit overgeneral rules, which are overgeneral yet stronger/fitter than not-overgeneral competitors. Fortunately, such rules are only possible under specific circumstances. A number of papers seek to characterize problems which are hard for LCS (Goldberg et al. 1992; Kovacs 2000, 2001, 2004; Bernadó-Mansilla and Ho 2005; Bagnall and Zatuchna 2005) while others model evolutionary dynamics (Butz et al. 2004a, b, 2007; Butz 2006; Orriols-Puig 2007c, d) and yet others attempt to reconstruct LCS from first principles using probabilistic models (Drugowitsch 2007, 2008; Edakunni et al. 2009).

### Hierarchies and Ensembles of LCS

Hierarchical LCS have been studied for some time and Barry (1996) reviews early work. Dorigo and Colombetti (1998), Donnart and Meyer (1996a, b), and Donnart (1998) apply hierarchical LCS to robot control while Barry (2000) uses hierarchical XCSs to learn long



sequences of actions. The ensembles field (► Sect. 3.3) studies how to combine predictions (Kuncheva 2004) and all the above could be reformulated as ensembles of LCS. Some recent work has taken this approach (Dam et al. 2005; Bull et al. 2007).

### 3.5.7 Conclusions

LCS face certain inherent difficulties; Michigan systems face complex credit assignment problems while in Pittsburgh systems, run-time can be a major issue. The same is true for all GBML systems, but the Michigan approach has been explored far more extensively within LCS than elsewhere. Recently, there has been much integration with mainstream machine learning and much research on representations and credit assignment algorithms. Most recent applications have been to data mining and function approximation, although some work continues on reinforcement learning. Future directions are likely to include exposing more of the LCS to evolution and further integration with machine learning, ensembles, memetic algorithms, and multiobjective optimization.

#### Reading

No general up-to-date introduction to LCS exists. For the basics see Goldberg (1989) and the introductory parts of Kovacs (2004) or Butz (2006). For a good introduction to representations and operators see chapter 6 of Freitas (2002a). For a review of early LCS see Barry (2000). For reviews of LCS research see Wilson and Goldberg (1989), Lanzi and Riolo (2000), and Lanzi (2008). For a review of state-of-the-art GBML and empirical comparison to non-evolutionary pattern recognition methods see Orriols-Puig et al. (2008b). For other comparisons with non-evolutionary methods see Bonelli and Alexandre (1991), Greenyer (2000), Saxon and Barry (2000), Wilson (2000), Bernadó et al. (2002), and Bernadó-Mansilla and Garrell-Guiu (2003). Finally, the LCS bibliography (Kovacs 2009) has over 900 references.

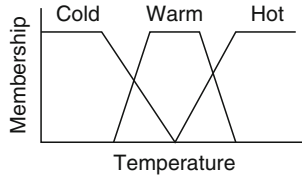
## 3.6 Genetic Fuzzy Systems

Following the section on LCS, this section covers a second actively developing approach to evolving rule-based systems. We will see that the two areas overlap considerably and that the distinction between them is somewhat arbitrary. Nonetheless, the two communities and their literatures are somewhat disjoint.

Fuzzy logic is a major paradigm in soft computing which provides a means of approximate reasoning not found in traditional crisp logic. Genetic fuzzy systems (GFS) apply evolution to fuzzy learning systems in various ways: GAs, GP, and evolution strategies have all been used. We will cover a particular form of GFS called genetic fuzzy rule-based systems (FRBS), which are also known as learning fuzzy classifier systems (LFCS) (Bonarini 2000) or referred to as, for example, “genetic learning of fuzzy rules” and (for reinforcement learning tasks) “fuzzy Q-learning”. Like other LCS, FRBS evolve if-then rules but in FRBS the rules are fuzzy. Most systems are Pittsburgh, but there are many Michigan examples (Valenzuela-Rendón 1991, 1998; Geyer-Schulz 1997; Bonarini 2000; Orriols-Puig et al. 2007a, 2008a; Casillas et al. 2007). In addition to FRBS, we briefly cover genetic fuzzy NNs, but we do not cover genetic fuzzy clustering (see Oscar-Cordón et al. (2001)).

In the terminology of fuzzy logic, ordinary scalar values are called *crisp* values. A *membership function* defines the degree of match between crisp values and a set of fuzzy linguistic

terms. The set of terms is a *fuzzy set*. The following figure shows a membership function for the set {cold, warm, and hot}.



Each crisp value matches *each* term to some degree in the interval [0,1], so, for example, a membership function might define 5° as 0.8 cold, 0.3 warm and 0.0 hot. The process of computing the membership of each term is called fuzzification and can be considered a form of discretization. Conversely, defuzzification refers to computing a crisp value from fuzzy values.

Fuzzy rules are condition/action (IF–THEN) rules composed of a set of linguistic variables (e.g., temperature, humidity), which can each take on linguistic terms (e.g., cold, warm, and hot). For example:

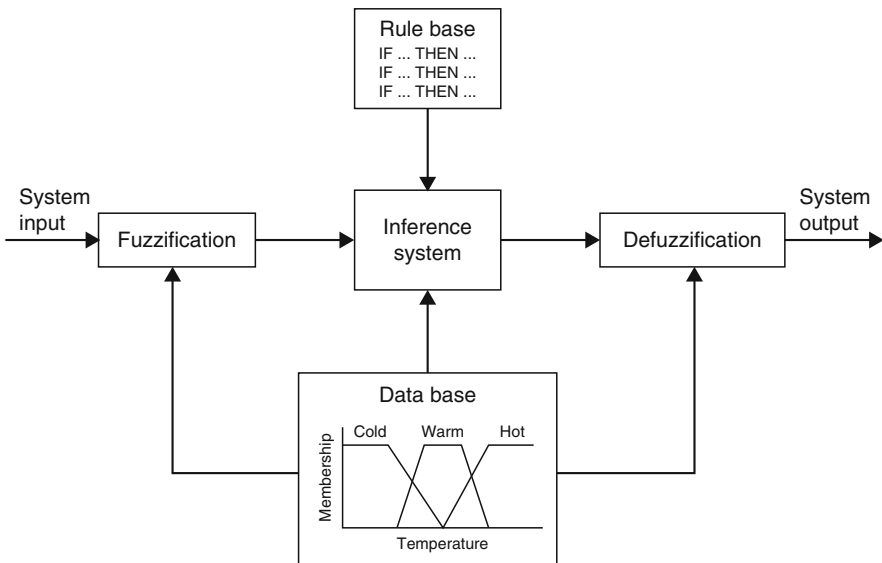
IF temperature IS cold AND humidity IS high THEN heater IS high  
IF temperature IS warm AND humidity IS low THEN heater IS medium

As illustrated in Fig. 12, a fuzzy rule-based system consists of:

- A rule base (RB) of fuzzy rules
- A data base (DB) of linguistic terms and their membership functions
- Together the RB and DB are the *knowledge base* (KB)
- A fuzzy inference system which maps from fuzzy inputs to a fuzzy output
- Fuzzification and defuzzification processes

Fig. 12

Components and information flow in a fuzzy rule-based system. (Adapted from Hekanaho 1995.)



### 3.6.1 Evolution of FRBSs

We distinguish (i) genetic tuning and (ii) genetic learning of DB, RB, or inference engine parameters.

#### Genetic Tuning

The concept behind genetic tuning is to first train a hand-crafted FRBS and then to evolve the DB (linguistic terms and membership functions) to improve performance. In other words, we do not alter the hand-crafted rule base but only tune its parameters. Specifically, we can adjust the shape of the membership functions and the parameterized expressions in the (adaptive) inference system and adapt defuzzification methods.

#### Genetic Learning

The concept of genetic learning is to evolve the DB, RB, or inference engine parameters. There are a number of approaches. In *genetic rule learning*, we usually predefine the DB by hand and evolve the RB. In *genetic rule selection* we use the GA to remove irrelevant, redundant, incorrect, or conflicting rules. This is a similar role to condensation in LCS (see Sect. 3.5.5). In *genetic KB learning* we learn both the DB and RB. We either learn the DB first and then learn the RB or we iteratively learn a series of DBs and evaluate each one by learning an RB using it.

It is also possible to learn components simultaneously, which may produce better results though the larger search space makes it slower and more difficult than adapting components independently. As examples, Morimoto et al. (1997) learn the DB and RB simultaneously while Homaitar and McCormick (1995) learn KB components and inference engine parameters simultaneously.

Recently, Sánchez and Couso (2007) claimed that all existing GFS have been applied to crisp data and that with such data the benefits of GFS compared to other learning methods are limited to linguistic interpretability. However, GFS has the potential to outperform other methods on fuzzy data and they identify three cases (Sánchez and Couso 2007, p. 558):

1. Crisp data with hand-added fuzziness
2. Transformations of data based on semantic interpretations of fuzzy sets
3. Inherently fuzzy data

They argue that GFS should use fuzzy fitness functions in such cases to deal directly with the uncertainty in the data and propose such systems as a new class of GFS to add to the taxonomy of Herrera (2008).

### 3.6.2 Genetic Neuro-fuzzy Systems

A neuro-fuzzy system (NFS) or fuzzy neural network (FNN) is any combination of fuzzy logic and neural networks. Among the many examples of such systems, Liangjie and Yanda (1996) use a GA to minimize the error of the NN, Hanebeck and Schmidt (1996) use both a GA and backpropagation to minimize error, Perneel and Themlin (1995) optimize a fuzzy expert system using a GA and NN, and Morimoto et al. (1997) use a NN to approximate the fitness function for a GA, which adapts membership functions and controls rules. See Cordon et al.

(2001) for an introduction to NFS, Linkens and Nyongesa (1996) for a review of EAs, NNs, and fuzzy logic from the perspective of intelligent control, and He et al. (1999) for a discussion of combining the three. Kolman and Margaliot (2009) introduce fuzzy all-permutations rule-bases (FARBs), which are mathematically equivalent to NNs.

### 3.6.3 Conclusions

Herrera (2008, p. 38) lists the following active areas within GFS:

1. Multi-objective genetic learning of FRBSs: interpretability–precision trade-off
2. GA-based techniques for mining fuzzy association rules and novel data mining approaches
3. Learning genetic models based on low quality data (e.g., noisy data)
4. Genetic learning of fuzzy partitions and context adaptation
5. Genetic adaptation of inference engine components
6. Revisiting the Michigan-style GFSs

Herrera also lists (p. 42) current issues for GFS:

1. Human readability
2. New data mining tasks: frequent and interesting pattern mining, mining data streams
3. Dealing with high dimensional data

#### Reading

There is a substantial GFS literature. Notable works include the four seminal 1991 papers on genetic tuning of the DB (Karr 1991), the Michigan approach (Valenzuela-Rendón 1989), the Pittsburgh approach (Thrift 1991), and relational matrix-based FRBS (Pham and Karaboga 1991). Subsequent work includes Geyer-Schulz's 1997 book on Michigan fuzzy LCS learning RBs with GP (Geyer-Schulz 1997), Bonarini's 2000 introductory chapter from an LCS perspective (Bonarini 2000), Mitra and Hayashi's 2000 survey of neuro-fuzzy rule generation methods (Mitra and Hayashi 2000), Cordon et al.'s 2001 book on genetic fuzzy systems in general (Cordón et al. 2001), Angelov's 2002 book on evolving FRBS (Angelov 2002), chapter 10 of Freitas' 2002 book on evolutionary data mining (Freitas 2002a), Herrera's 2008 survey article on GFS (Herrera 2008) (which lists further key reading), and finally Kolman and Margaliot's 2009 book on the neuro-fuzzy FARB approach (Kolman and Margaliot 2009).

## 4 Conclusions

The reader should need no convincing that GBML is a very diverse and active area. Although much integration with mainstream machine learning has taken place in the last 10 years, more is needed. The use of multiobjective EAs in GBML is spreading. Integration with ensembles is natural given the population-based nature of EAs but it is only just beginning. Other areas which need attention are memetics, meta-learning, hyperheuristics, and estimation of distribution algorithms. In addition to further integration with other areas, the constituent areas of GBML need more interaction with each other.

Two persistent difficulties for GBML are worth highlighting. First, run-time speed remains an issue as EAs are much slower than most other methods. While this sometimes matters little

(e.g., in off-line learning with modest datasets), it is sometimes equally critical (e.g., in stream mining). Various methods to speed up GBML exist (see e.g., Freitas (2002a, Sect. 12.1.3) and more research is warranted, but this may simply remain a weakness. The second difficulty is theory. EA theory is notoriously difficult and when coupled with other processes becomes even less tractable. Nonetheless, substantial progress has been made in the past 10 years, most notably with LCS.

Other active research directions will no doubt include meta-learning such as the evolution of bias (e.g., selection of representation), evolving heuristics and learning rules for specific problem classes, and other forms of self-adaptation. In the area of data preparation, Freitas (2002a, Sect. 12.2.1) argues that attribute construction is a promising area for GBML and that filter methods for feature selection are faster than wrappers and deserve more GBML research. Finally, many specialized learning problems (not to mention specific applications) remain little or unexplored with GBML, including ranking, semi-supervised learning, transductive learning, inductive transfer, learning to learn, stream mining, and no doubt others that have not yet been formulated.

## Acknowledgments

---

Thanks to my editor Thomas Bäck for his patience and encouragement, and to Larry Bull, John R. Woodward, Natalio Krasnogor, Gavin Brown and Arjun Chandra for comments.

## Glossary

---

EA	Evolutionary Algorithm
FRBS	Fuzzy Rule-Based System
GA	Genetic Algorithm
GBML	Genetics-Based Machine Learning
GFS	Genetic Fuzzy System
GP	Genetic Programming
LCS	Learning Classifier System
NN	Neural Network
SL	Supervised Learning

## References

---

- Abbass HA (2003) Speeding up backpropagation using multiobjective evolutionary algorithms. *Neural Comput*, 15(11):2705–2726
- Ackley DH, Littman ML (1992) Interactions between learning and evolution. In: Langton C, Taylor C, Rasmussen S, Farmer J (eds) *Artificial life II: Santa Fe institute studies in the sciences of Complexity*, vol 10. Addison-Wesley, New York, pp 487–509
- Aguilar-Ruiz J, Riquelme J, Toro M (2003) Evolutionary learning of hierarchical decision rules. *IEEE Trans Syst Man Cybern B* 33(2):324–331
- Ahluwalia M, Bull L (1999) A genetic programming-based classifier system. In: Banzhaf et al. (eds) *GECCO-99: Proceedings of the genetic and evolutionary computation conference*. Morgan Kaufmann, San Francisco, pp 11–18

- Andersen HC, Tsoi AC (1993) A constructive algorithm for the training of a multi-layer perceptron based on the genetic algorithm. *Complex Syst*, 7(4):249–268
- Angeline PJ, Saunders GM, Pollack JB (1994) An evolutionary algorithm that constructs recurrent neural networks. *IEEE Trans Neural Netw* 5:54–65
- Angelov P (2002) Evolving rule-based models: a tool for design of flexible adaptive systems. *Studies in fuzziness and soft computing*, vol 92. Springer, Heidelberg
- Asuncion A, Newman DJ (2009) UCI machine learning repository. <http://www.ics.uci.edu/~mllearn/MLRepository.html>
- Bacardit J, Stout M, Hirst JD and Krasnogor N (2008) Data mining in proteomics with learning classifier systems. In: Bull L, Bernadó Mansilla E, Holmes J (eds) *Learning classifier systems in data mining*. Springer, Berlin, pp 17–46
- Bacardit J, Burke EK, Krasnogor N (2009a) Improving the scalability of rule-based evolutionary learning. *Memetic Comput* 1(1):55–57
- Bacardit J, Stout M, Hirst JD, Valencia A, Smith RE, Krasnogor N (2009b) Automated alphabet reduction for protein datasets. *BMC Bioinformatics*, vol 10, 6
- Bacardit J (2004) Pittsburgh genetic-based machine learning in the data mining era: representations, generalization, and run-time. PhD thesis, Universitat Ramon Llull, Barcelona, Spain
- Bacardit J, Goldberg DE, Butz MV (2007) Improving the performance of a Pittsburgh learning classifier system using a default rule. In: Kovacs T, Llorà X, Takadama K, Lanzi PL, Stolzmann W, Wilson SW (eds) *Learning Classifier Systems. International workshops, IWLCS 2003–2005, Revised selected papers, Lecture notes in computer science*, vol 4399. Springer, Berlin, pp 291–307
- Bacardit J, Krasnogor N (2008) Empirical evaluation of ensemble techniques for a Pittsburgh learning classifier system. In: Bacardit J, Bernadó-Mansilla E, Butz M, Kovacs T, Llorà X, Takadama K (eds) *Learning Classifier Systems. 10th and 11th International Workshops (2006–2007), Lecture notes in computer science*, vol 4998. Springer, Berlin, pp 255–268
- Bagnall AJ, Zatuchna ZV (2005) On the classification of maze problems. In: Bull L, Kovacs T (eds) *Applications of learning classifier systems*. Springer, Berlin, pp 307–316
- Banzhaf W, Daida J, Eiben AE, Garzon MH, Honavar V, Jakiela M, Smith RE (eds) (1999) *GECCO-99: Proceedings of the genetic and evolutionary computation conference*. Morgan Kaufmann, San Francisco, CA
- Barry A (1996) Hierarchy formulation within classifiers system: a review. In: Goodman EG, Uskov VL, Punch WF (eds) *Proceedings of the first international conference on evolutionary algorithms and their application EVCA'96. The Presidium of the Russian Academy of Sciences, Moscow*, pp 195–211
- Barry A (2000) XCS performance and population structure within multiple-step environments. PhD thesis, Queen's University Belfast, Belfast
- Beielstein T, Markon S (2002) Threshold selection, hypothesis tests and DOE methods. In: 2002 congress on evolutionary computation. IEEE Press, Washington, DC, pp 777–782
- Belew RK, McInerney J, Schraudolph NN (1992) Evolving networks: using the genetic algorithm with connectionistic learning. In: Langton CG, Taylor C, Farmer JD, Rasmussen S (eds) *Proceedings of the 2nd conference on artificial life*. Addison-Wesley, New York, pp 51–548
- Bernadó E, Llorà X, Garrell JM (2002) XCS and GALE: a comparative study of two learning classifier systems on data mining. In: Lanzi PL, Stolzmann W, Wilson SW (eds) *Advances in learning classifier systems, Lecture notes in artificial intelligence*, vol 2321. Springer, Berlin, pp 115–132
- Bernadó-Mansilla E, Garrell-Guiu JM (2003) Accuracy-based learning classifier systems: models, analysis and applications to classification tasks. *Evolut Comput* 11(3):209–238
- Bernadó-Mansilla E, Ho TK (2005) Domain of competence of XCS classifier system in complexity measurement space. *IEEE Trans Evolut Comput* 9(1):82–104
- Bonarini A (2000) An introduction to learning fuzzy classifier systems. In: Lanzi PL, Stolzmann W, Wilson SW (eds) *Learning classifier systems: from foundations to applications, Lecture note in artificial intelligence*, vol 1813. Springer, Berlin, pp 83–104
- Bonelli P, Alexandre P (1991) An efficient classifier system and its experimental comparison with two representative learning methods on three medical domains. In: Booker LB, Belew RK (eds) *Proceedings of the 14th international conference on genetic algorithms (ICGA'91)*. Morgan Kaufmann, San Francisco, CA, pp 288–295
- Booker LB (1989) Triggered rule discovery in classifier systems. In: Schaffer JD (ed) *Proceedings of the 3rd international conference on genetic algorithms (ICGA-89)*. Morgan Kaufmann, San Francisco, CA, pp 265–274
- Booker LB (1991) Representing attribute-based concepts in a classifier system. In: Rawlins GJE (ed) *Proceedings of the first workshop on foundations of genetic algorithms (FOGA91)*. Morgan Kaufmann, San Mateo, CA, pp 115–127
- Booker LB (2005a) Adaptive value function approximations in classifier systems. In: *GECCO '05: proceedings of the 2005 workshops on genetic and evolutionary computation*. ACM, New York, pp 90–91

- Booker LB (2005b) Approximating value functions in classifier systems. In: Bull L, Kovacs T (eds) *Foundations of learning classifier systems* (Studies in fuzziness and soft computing), Lecture notes in artificial intelligence, vol 183, Springer, Berlin, pp 45–61
- Booker LB, Belew RK (eds) (1991) *Proceedings of the 4th international conference on genetic algorithms (ICGA91)*. Morgan Kaufmann, San Francisco, CA
- Bot MCJ, Langdon WB (2000) Application of genetic programming to induction of linear classification trees. In: *Genetic programming: proceedings of the 3rd European conference (EuroGP 2000)*, Lecture notes in computer science, vol 1802, Springer, Berlin, pp 247–258
- Breiman L (1996) Bagging predictors. *Mach Learn* 24 (2):123–140
- Breiman L (1998) Arcing classifiers. *Ann Stat* 26(3): 801–845
- Brown G (2010) Ensemble learning. In: Sammut C, Webb G (eds) *Encyclopedia of machine learning*. Springer, Berlin
- Brown G, Wyatt J, Harris R, Yao X (2005) Diversity creation methods: A survey and categorisation. *J Inform Fusion* 6(1):5–20
- Bull L (2009) On dynamical genetic programming: simple Boolean networks in learning classifier systems. *IJPEDS* 24(5):421–442
- Bull L, Studley M, Bagnall T, Whitley I (2007) On the use of rule-sharing in learning classifier system ensembles. *IEEE Trans Evolut Comput* 11:496–502
- Bull L (2005) Two simple learning classifier systems. In: Bull L, Kovacs T (eds) *Foundations of learning classifier systems* (Studies in fuzziness and soft computing), Lecture notes in artificial intelligence, vol 183, Springer, Berlin, pp 63–90
- Bull L, O'Hara T (2002) Accuracy-based neuro and neuro-fuzzy classifier systems. In: Langdon WB, Cantú-Paz E, Mathias K, Roy R, Davis R, Poli R, Balakrishnan K, Honavar V, Rudolph G, Wegener J, Bull L, Potter MA, Schultz AC, Miller JF, Burke E, Jonoska N (eds) *GECCO 2002: Proceedings of the genetic and evolutionary computation conference*. Morgan Kaufmann, San Francisco, CA, pp 905–911
- Burke EK, Hyde MR, Kendall G, Ochoa G, Ozcan E, Woodward JR (2009) Exploring hyper-heuristic methodologies with genetic programming. In: Mumford C, Jain L (eds) *Collaborative computational intelligence*. Springer, Berlin
- Burke EK, Kendall G (2005) Introduction. In: Burke EK, Kendall G (eds) *Search methodologies: introductory tutorials in optimization and decision support techniques*. Springer, Berlin, pp 5–18
- Burke EK, Kendall G, Newall J, Hart E, Russ P, Schulenburg S (2003) Hyper-heuristics: an emerging direction in modern search technology. In: Glover F, Kochenberger G (eds) *Handbook of meta-heuristics*. Kluwer, Norwell, MA, pp 457–474
- Butz MV, Kovacs T, Lanzi PL, Wilson SW (2004b) Toward a theory of generalization and learning in XCS. *IEEE Trans Evolut Comput* 8(1):8–46
- Butz MV (2002a) An algorithmic description of ACS2. In: Lanzi PL, Stolzmann W, Wilson SW (eds) *Learning classifier systems: from foundations to applications*, Lecture notes in artificial intelligence, vol 2321. Springer, Berlin, pp 211–229
- Butz MV (2002b) Anticipatory learning classifier systems. Kluwer, Norwell, MA
- Butz MV, Goldberg DV, Stolzmann W (2000a) Introducing a genetic generalization pressure to the anticipatory classifier system – part 1: theoretical approach. In: Whitley D, Goldberg D, Cantú-Paz E, Spector L, Parmee I, Beyer H-G (eds) *Proceedings of genetic and evolutionary computation conference (GECCO 2000)*. Morgan Kaufmann, San Francisco, CA, pp 34–41
- Butz MV, Goldberg DE, Stolzmann W (2000b) Introducing a genetic generalization pressure to the anticipatory classifier system – part 2: performance analysis. In: Whitley D, Goldberg D, Cantú-Paz E, Spector L, Parmee I, Beyer H-G (eds) *Proceedings of genetic and evolutionary computation conference (GECCO 2000)*. Morgan Kaufmann, San Francisco, CA, pp 42–49
- Butz MV, Wilson SW (2001) An algorithmic description of XCS. In: Lanzi PL, Stolzmann W, Wilson SW (eds) *Advances in learning classifier systems*, Lecture notes in artificial intelligence, vol 1996. Springer, Berlin, pp 253–272
- Butz MV (2005) Kernel-based, ellipsoidal conditions in the real-valued XCS classifier system. In: Beyer HG et al. (eds) *Proceedings of the genetic and evolutionary computation conference (GECCO 2005)*. ACM, New York, pp 1835–1842
- Butz MV (2006) Rule-based evolutionary online learning systems: a principled approach to LCS analysis and design. *Studies in fuzziness and soft computing*. Springer, Berlin
- Butz MV, Goldberg DE, Lanzi PL (2004a) Bounding learning time in XCS. In: *Genetic and evolutionary computation (GECCO 2004)*, Lecture notes in computer science, vol 3103. Springer, Berlin, pp 739–750
- Butz MV, Goldberg DE, Lanzi PL (2005a) Computational complexity of the XCS classifier system. In: Bull L, Kovacs T (eds) *Foundations of learning classifier systems*, Studies in fuzziness and soft computing, Lecture notes in artificial intelligence, vol 183. Springer, Berlin, pp 91–126
- Butz MV, Goldberg DE, Lanzi PL (2005b) Gradient descent methods in learning classifier systems: improving XCS performance in multistep problems. *IEEE Trans Evolut Comput* 9(5):452–473

- Butz MV, Goldberg DE, Lanzi PL, Sastry K (2007) Problem solution sustenance in XCS: Markov chain analysis of niche support distributions and the impact on computational complexity. *GP and Evol Machines* 8(1):5–37
- Butz MV, Lanzi PL, Wilson SW (2006) Hyper-ellipsoidal conditions in XCS: rotation, linear approximation, and solution structure. In: Cattolico M (ed) *Proceedings of the genetic and evolutionary computation conference (GECCO 2006)*. ACM, New York, pp 1457–1464
- Butz MV, Pelikan M (2006) Studying XCS/BOA learning in Boolean functions: structure encoding and random Boolean functions. In: Cattolico M et al. (eds) *Genetic and evolutionary computation conference, GECCO 2006*. ACM, New York, pp 1449–1456
- Butz MV, Pelikan M, Llorà X, Goldberg DE (2005) Extracted global structure makes local building block processing effective in XCS. In: Beyer HG, O'Reilly UM (eds) *Proceedings of the genetic and evolutionary computation conference, GECCO 2005*. ACM, New York, pp 655–662
- Butz MV, Pelikan M, Llorà X, Goldberg DE (2006) Automated global structure extraction for effective local building block processing in XCS. *Evolut Comput* 14(3):345–380
- Butz MV, Stalsh P, Lanzi PL (2008) Self-adaptive mutation in XCSE. In *GECCO '08: Proceedings of the 10th annual conference on genetic and evolutionary computation*. ACM, New York, pp 1365–1372
- Cantú-Paz E, Kamath C (2003) Inducing oblique decision trees with evolutionary algorithms. *IEEE Trans Evolut Comput* 7(1):54–68
- Cantú-Paz E (2002) Feature subset selection by estimation of distribution algorithms. In: *GECCO '02: Proceedings of the genetic and evolutionary computation conference*. Morgan Kaufmann, San Francisco, CA, pp 303–310
- Caruana R, Niculescu-Mizil A (2006) An empirical comparison of supervised learning algorithms. In: *ICML '06: Proceedings of the 23rd international conference on machine learning*. ACM, New York, pp 161–168
- Casillas J, Carse B, Bull L (2007) Fuzzy-XCS: a Michigan genetic fuzzy system. *IEEE Trans Fuzzy Syst* 15: 536–550
- Castilloa PA, Merelo JJ, Arenas MG, Romero G (2007) Comparing evolutionary hybrid systems for design and optimization of multilayer perceptron structure along training parameters. *Inform Sciences*, 177 (14):2884–2905
- Chalmers D (1990) The evolution of learning: An experiment in genetic connectionism. In: Touretsky E (ed) *Proceedings 1990 connectionist models summer school*. Morgan Kaufmann, San Francisco, CA pp 81–90
- Chandra A, Yao X (2006a) Ensemble learning using multi-objective evolutionary algorithms. *J Math Model Algorithm* 5(4):417–445
- Chandra A, Yao X (2006b) Evolving hybrid ensembles of learning machines for better generalisation. *Neuro-computing* 69(7–9):686–700
- Cho S, Cha K (1996) Evolution of neural net training set through addition of virtual samples. In: *Proceedings of the 1996 IEEE international conference on evolutionary computation*. IEEE Press, Washington DC, pp 685–688
- Cho S-B (1999) Pattern recognition with neural networks combined by genetic algorithm. *Fuzzy Set Syst* 103:339–347
- Cho S-B, Park C (2004) Speciated GA for optimal ensemble classifiers in DNA microarray classification. In: *Congress on evolutionary computation (CEC 2004)*, vol 1. pp 590–597
- Corcoran AL, Sen S (1994) Using real-valued genetic algorithms to evolve rule sets for classification. In: *Proceedings of the IEEE conference on evolutionary computation*. IEEE Press, Washington DC, pp 120–124
- Cordón O, Herrera F, Hoffmann F, Magdalena L (2001) *Genetic fuzzy systems*. World Scientific, Singapore
- Cribbs III HB, Smith RE (1996) Classifier system renaissance: new analogies, new directions. In: Koza JR, Goldberg DE, Fogel DB, Riolo RL (eds) *Genetic programming 1996: proceedings of the first annual conference*. MIT Press, Cambridge, MA, Stanford University, CA, USA, 28–31 July 1996. pp 547–552
- Dam HH, Abbass HA, Lokan C, Yao X (2008) Neural-based learning classifier systems. *IEEE Trans Knowl Data Eng* 20(1):26–39
- Dam HH, Abbass HA, Lokan C (2005) DXCS: an XCS system for distributed data mining. In: Beyer HG, O'Reilly UM (eds) *Genetic and evolutionary computation conference, GECCO 2005*. pp 1883–1890
- Dasdan A, Oflazer K (1993) Genetic synthesis of unsupervised learning algorithms. Technical Report BU-CEIS-9306, Department of Computer Engineering and Information Science, Bilkent University, Ankara
- De Jong KA, Spears WM (1991) Learning concept classification rules using genetic algorithms. In: *Proceedings of the twelfth international conference on artificial intelligence IJCAI-91*, vol 2. Morgan Kaufmann, pp 651–656
- De Jong KA, Spears WM, Gordon DF (1993) Using genetic algorithms for concept learning. *Mach Learn* 3:161–188
- Dietterich TG (1998) Machine-learning research: four current directions. *AI Mag* 18(4):97–136
- Divina F, Keijzer M, Marchiori E (2002) Non-universal suffrage selection operators favor population diversity in genetic algorithms. In: *Benelearn 2002: proceedings of the 12th Belgian-Dutch conference*



- on machine learning (Technical report UU-CS-2002-046). pp 23–30
- Divina F, Keijzer M, Marchiori E (2003) A method for handling numerical attributes in GA-based inductive concept learners. In: *Proceedings of the genetic and evolutionary computation conference (GECCO-2003)*. Springer, Berlin, pp 898–908
- Divina F, Marchiori E (2002) Evolutionary concept learning. In: Langdon WB, Cantú-Paz E, Mathias K, Roy R, Davis D, Poli R, Balakrishnan K, Honavar V, Rudolph G, Wegener J, Bull L, Potter MA, Schultz AC, Miller JF, Burke E, Jonoska N (eds) *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*. Morgan Kaufmann, San Francisco, CA, New York, 9–13 July 2002. pp 343–350
- Dixon PW, Corne D, Oates MJ (2002) A ruleset reduction algorithm for the XCS learning classifier system. In: Lanzi PL, Stolzmann W, Wilson SW (eds) *Learning classifier systems, 5th international workshop (IW LCS 2002)*, Lecture notes in computer science, vol 2661. Springer, Berlin, pp 20–29
- Donnart J-Y (1998) Cognitive architecture and adaptive properties of an motivationally autonomous animat. PhD thesis, Université Pierre et Marie Curie. Paris, France
- Donnart J-Y, Meyer J-A (1996a) Hierarchical-map Building and Self-positioning with MonaLysa. *Adapt Behav* 5(1):29–74
- Donnart J-Y, Meyer J-A (1996b) Learning reactive and planning rules in a motivationally autonomous animat. *IEEE Trans Syst Man Cybern B Cybern* 26(3):381–395
- Dorigo M, Colombetti M (1998) Robot shaping: an experiment in behavior engineering. MIT Press/Bradford Books, Cambridge, MA
- Drugowitsch J, Barry A (2005) XCS with eligibility traces. In: Beyer H-G, O'Reilly U-M (eds) *Genetic and evolutionary computation conference, GECCO 2005*. ACM, New York, pp 1851–1858
- Drugowitsch J (2008) Design and analysis of learning classifier systems: a probabilistic approach. Springer, Berlin
- Drugowitsch J, Barry A (2007) A formal framework and extensions for function approximation in learning classifier systems. *Mach Learn* 70(1):45–88
- Edakunni NE, Kovacs T, Brown G, Marshall JAR, Chandra A (2009) Modelling UCS as a mixture of experts. In: *Proceedings of the 2009 Genetic and Evolutionary Computation Conference (GECCO'09)*. ACM, pp 1187–1194
- Floreano D, Dürr P, Mattiussi C (2008) Neuroevolution: from architectures to learning. *Evol Intell* 1(1):47–62
- Folino G, Pizzuti C, Spezzano G (2003) Ensemble techniques for parallel genetic programming based classifiers. In: *Proceedings of the sixth European conference on genetic programming (EuroGP'03)*, Lecture notes in computer science, vol 2610. Springer, Berlin, pp 59–69
- Freitas AA (2002a) Data mining and knowledge discovery with evolutionary algorithms. Springer, Berlin
- Freitas AA (2002b) A survey of evolutionary algorithms for data mining and knowledge discovery. In: Ghosh A, Tsutsui S (eds) *Advances in evolutionary computation*. Springer, Berlin, pp 819–845
- Freund Y, Schapire R (1996) Experiments with a new boosting algorithm. In: *Proceedings of the international conference on machine learning (ICML'96)*, Bari, Italy, pp 148–156
- Freund Y, Schapire R (1999) A short introduction to boosting. *J Jpn Soc Artif Intell* 14(5):771–780
- Fürnkranz J (1998) Integrative windowing. *J Artif Intell Res* 8:129–164
- Gagné C, Sebag M, Schoenauer M, Tomassini M (2007) Ensemble learning for free with evolutionary algorithms? In: *GECCO '07: Proceedings of the 9th annual conference on genetic and evolutionary computation*. ACM, New York, pp 1782–1789
- Gathercole C, Ross P (1997) Tackling the Boolean even n parity problem with genetic programming and limited-error fitness. In: Koza JR, Deb K, Dorigo M, Fogel DB, Garzon M, Iba H, Riolo RL (eds) *Genetic programming 1997: proceedings second annual conference*. Morgan Kaufmann, San Francisco, CA, pp 119–127
- Gérard P, Sigaud O (2003) Designing efficient exploration with MACS: Modules and function approximation. In: Cantú-Paz E, Foster JA, Deb K, Davis D, Roy R, O'Reilly U-M, Beyer H-G, Standish R, Kendall G, Wilson S, Harman M, Wegener J, Dasgupta D, Potter MA, Schultz AC, Dowsland K, Jonoska N, Miller J (eds) *Genetic and evolutionary computation – GECCO-2003*, Lecture notes in computer science, vol 2724. Springer, Berlin, pp 1882–1893
- Gerard P, Stolzmann W, Sigaud O (2002) YACS, a new learning classifier system using anticipation. *J Soft Comput* 6(3–4):216–228
- Geyer-Schulz A (1997) Fuzzy rule-based expert systems and genetic machine learning. *Physica, Heidelberg*
- Giordana A, Neri F (1995) Search-intensive concept induction. *Evolut Comput* 3:375–416
- Giordana A, Saitta L (1994) Learning disjunctive concepts by means of genetic algorithms. In: *Proceedings of the international conference on machine learning*, Brunswick, NJ, pp 96–104
- Giraldez R, Aguilar-Ruiz J, Riquelme J (2003) Natural coding: a more efficient representation for evolutionary learning. In: *Proceedings of the genetic and evolutionary computation conference (GECCO-2003)*. Springer, Berlin, pp 979–990

- Giraud-Carrier C, Keller J (2002) Meta-learning. In: Meij J (ed) *Dealing with the data flood*. STT/Beweton, Hague, The Netherlands
- Goldberg DE (1989) *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, Reading, MA
- Goldberg DE, Horn J, Deb K (1992) What makes a problem hard for a classifier system? In: *Collected abstracts for the first international workshop on learning classifier system (IWLCS-92)*. (Also technical report 92007 Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign). Available from ENCORE ([ftp://ftp.krl.caltech.edu/pub/EC/Welcome.html](http://ftp.krl.caltech.edu/pub/EC/Welcome.html)) in the section on Classifier Systems
- Greene DP, Smith SF (1993) Competition-based induction of decision models from examples. *Mach Learn* 13:229–257
- Greene DP, Smith SF (1994) Using coverage as a model building constraint in learning classifier systems. *Evolut Comput* 2(1):67–91
- Greene DP, Smith SF (1987) A genetic system for learning models of consumer choice. In: *Proceedings of the second international conference on genetic algorithms and their applications*. Morgan Kaufmann, San Francisco, CA, pp 217–223
- Greenyer A (2000) The use of a learning classifier system JXCS. In: van der Putten P, van Someren M (eds) *CoIL challenge 2000: the insurance company case*. Leiden Institute of Advanced Computer Science, June 2000. Technical report 2000–09
- Gruau F (1995) Automatic definition of modular neural networks. *Adapt Behav* 3(2):151–183
- Hanebeck D, Schmidt K (1996) Genetic optimization of fuzzy networks. *Fuzzy set syst* 79:59–68
- Hansen LK, Salamon P (1990) Neural network ensembles. *IEEE Trans Pattern Anal Mach Intell* 12(10):993–1001
- Hart WE, Krasnogor N, Smith JE (eds) (2004) *Special issue on memetic algorithms, evolutionary computation*, vol 12, 3
- Hart WE, Krasnogor N, Smith JE (eds) (2005) *Recent advances in memetic algorithms*, *Studies in fuzziness and soft computing*, vol 166. Springer, Berlin
- He L, Wang KJ, Jin HZ, Li GB, Gao XZ (1999) The combination and prospects of neural networks, fuzzy logic and genetic algorithms. In: *IEEE mid-night-sun workshop on soft computing methods in industrial applications*. IEEE, Washington, DC, pp 52–57
- Heitkötter J, Beasley D (2001) The hitch-hiker's guide to evolutionary computation (FAQ for comp.ai.genetic). Accessed 28/2/09. <http://www.aip.de/~ast/Evol-CompFAQ/>
- Hekanaho J (1995) Symbiosis in multimodal concept learning. In: *Proceedings of the 1995 international conference on machine learning (ICML'95)*. Morgan Kaufmann, San Francisco, pp 278–285
- Herrera F (2008) Genetic fuzzy systems: taxonomy, current research trends and prospects. *Evolut Intell* 1(1):27–46
- Holland JH (1976) *Adaptation*. In: Rosen R, Snell FM (eds) *Progress in theoretical biology*. Plenum, New York
- Holland JH (1986) Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In: Mitchell T, Michalski R, Carbonell J (eds) *Machine learning, an artificial intelligence approach*. vol. II, chap. 20. Morgan Kaufmann, San Francisco, CA, pp 593–623
- Holland JH, Booker LB, Colombetti M, Dorigo M, Goldberg DE, Forrest S, Riolo RL, Smith RE, Lanzi PL, Stolzmann W, Wilson SW (2000) What is a learning classifier system? In: Lanzi PL, Stolzmann W, Wilson SW (eds) *Learning classifier systems: from foundations to application*, *Lecture notes in artificial intelligence*, vol 1813. Springer, Berlin, pp 3–32
- Holland JH, Holyoak KJ, Nisbett RE, Thagard PR (1986) *Induction: processes of inference, learning, and discovery*. MIT Press, Cambridge, MA
- Holland JH, Reitman JS (1978) Cognitive systems based on adaptive algorithms. In: Waterman DA, Hayes-Roth F (eds) *Pattern-directed Inference Systems*. Academic Press, New York. Reprinted in: *Evolutionary Computation*. The Fossil Record. Fogel DB (ed) (1998) IEEE Press, Washington DC. ISBN: 0-7803-3481-7
- Homaifar A, McCormick E (1995) Simultaneous design of membership functions and rule sets for fuzzy controllers using genetic algorithms. *IEEE Trans Fuzzy Syst*, 3(2):129–139
- Howard D, Bull L (2008) On the effects of node duplication and connection-orientated constructivism in neural XCSF. In: Keijzer M et al. (eds) *GECCO-2008: proceedings of the genetic and evolutionary computation conference*. ACM, New York, pp 1977–1984
- Howard D, Bull L, Lanzi PL (2008) Self-adaptive constructivism in neural XCS and XCSF. In: Keijzer M et al. (eds) *GECCO-2008: proceedings of the genetic and evolutionary computation conference*. ACM, New York, pp 1389–1396
- Hu Y-J (1998) A genetic programming approach to constructive induction. In: *Genetic programming 1998: proceedings of the 3rd annual conference*. Morgan Kaufmann, San Francisco, CA, pp 146–151
- Hurst J, Bull L (2003) Self-adaptation in classifier system controllers. *Artifi Life Robot* 5(2):109–119
- Hurst J, Bull L (2004) A self-adaptive neural learning classifier system with constructivism for mobile robot control. In: Yao X et al. (eds) *Parallel problem solving from nature (PPSN VIII)*, *Lecture notes*

- in computer science, vol 3242. Springer, Berlin, pp 942–951
- Husbands P, Harvey I, Cliff D, Miller G (1994) The use of genetic algorithms for the development of sensorimotor control systems. In: Gaussier P, Nicoud J-D (eds) *From perception to action*. IEEE Press, Washington DC, pp 110–121
- Iba H (1999) Bagging, boosting and bloating in genetic programming. In: *Proceedings of the genetic and evolutionary computation conference (GECCO'99)*. Morgan Kaufmann, San Francisco, CA, pp 1053–1060
- IEEE (2000) *Proceedings of the 2000 congress on evolutionary computation (CEC'00)*. IEEE Press, Washington DC
- Ishibuchi H, Nakashima T (2000) Multi-objective pattern and feature selection by a genetic algorithm. In: *Proceedings of the 2000 genetic and evolutionary computation conference (GECCO'2000)*. Morgan Kaufmann, San Francisco, CA, pp 1069–1076
- Islam MM, Yao X, Murase K (2003) A constructive algorithm for training cooperative neural network ensembles. *IEEE Trans Neural Netw* 14:820–834
- Jain A, Zongker D (1997) Feature selection: evaluation, application and small sample performance. *IEEE Trans. Pattern Anal Mach Intell* 19(2):153–158
- Janikow CZ (1991) Inductive learning of decision rules in attribute-based examples: a knowledge-intensive genetic algorithm approach. PhD thesis, University of North Carolina
- Janikow CZ (1993) A knowledge-intensive genetic algorithm for supervised learning. *Mach Learn* 13:189–228
- Jin Y, Sendhoff B (2004) Reducing fitness evaluations using clustering techniques and neural network ensembles. In: *Genetic and evolutionary computation conference (GECCO–2004)*, Lecture notes in computer science, vol 3102. Springer, Berlin, pp 688–699
- John G, Kohavi R, Phleger K (1994) Irrelevant features and the feature subset problem. In: *Proceedings of the 11th international conference on machine learning*. Morgan Kaufmann, San Francisco, CA, pp 121–129
- Juan Liu J, Tin-Yau Kwok J (2000) An extended genetic rule induction algorithm. In: *Proceedings of the 2000 congress on evolutionary computation (CEC'00)*. IEEE Press, Washington DC, pp 458–463
- Kelly JD Jr, Davis L (1991) Hybridizing the genetic algorithm and the k nearest neighbors classification algorithm. In: Booker LB, Belew RK (eds) *Proceedings of the 4th international conference on genetic algorithms (ICGA91)*. Morgan Kaufmann, San Francisco, CA, pp 377–383
- Karr C (1991) Genetic algorithms for fuzzy controllers. *AI Expert* 6(2):26–33
- Kasabov N (2007) *Evolving connectionist systems: the knowledge engineering approach*. Springer, Berlin
- Keijzer M, Babovic V (2000) Genetic programming, ensemble methods, and the bias/variance/tradeoff – introductory investigation. In: *Proceedings of the European conference on genetic programming (EuroGP'00)*, Lecture notes in computer science, vol 1802. Springer, Berlin, pp 76–90
- Kitano H (1990) Designing neural networks by genetic algorithms using graph generation system. *J Complex Syst* 4:461–476
- Kolman E, Margaliot M (2009) Knowledge-based neurocomputing: a fuzzy logic approach, *Studies in fuzziness and soft computing*, vol 234. Springer, Berlin
- Kovacs T (1996) *Evolving optimal populations with XCS classifier systems*. Master's thesis, University of Birmingham, Birmingham, UK
- Kovacs T (1997) XCS classifier system reliably evolves accurate, complete, and minimal representations for Boolean functions. In: Chawdhry PK, Roy R, Pant RK (eds) *Soft computing in engineering design and manufacturing*. Springer, London, pp 59–68 <http://ftp.cs.bham.ac.uk/pub/authors/T.Kovacs/index.html>
- Kovacs T (2000) Strength or accuracy? Fitness calculation in learning classifier systems. In: Lanzi PL, Stolzmann W, Wilson SW (eds) *Learning classifier systems: from foundations to applications*, Lecture notes in artificial intelligence, vol 1813. Springer, Berlin, pp 143–160
- Kovacs T (2004) Strength or accuracy: credit assignment in learning classifier systems. Springer, Berlin
- Kovacs T (2009) *A learning classifier systems bibliography*. Department of Computer Science, University of Bristol. <http://www.cs.bris.ac.uk/~kovacs/lcs/search.html>
- Kovacs T, Kerber M (2001) What makes a problem hard for XCS? In: Lanzi PL, Stolzmann W, Wilson SW (eds) *Advances in learning classifier systems*, Lecture notes in artificial intelligence, vol 1996. Springer, Berlin, pp 80–99
- Kovacs T, Kerber M (2004) High classification accuracy does not imply effective genetic search. In: Deb K et al. (eds) *Proceedings of the 2004 genetic and evolutionary computation conference (GECCO)*, Lecture notes in computer science, vol 3102. Springer, Berlin, pp 785–796
- Koza JR (1992) *Genetic programming: on the programming of computers by means of natural selection*. MIT Press, Cambridge, MA
- Koza JR (1994) *Genetic Programming II*. MIT Press
- Krasnogor N (2002) *Studies on the theory and design space of memetic algorithms*. PhD thesis, University of the West of England
- Krasnogor N, Smith JE (2005) A tutorial for competent memetic algorithms: model, taxonomy and

- design issues. *IEEE Trans Evolut Comput* 9(5): 474–488
- Krasnogor N (2004) Self-generating metaheuristics in bioinformatics: the protein structure comparison case. *GP and Evol Machines* 5(2):181–201
- Krasnogor N, Gustafson S (2004) A study on the use of self-generation in memetic algorithms. *Natural Comput* 3(1):53–76
- Krawiec K (2002) Genetic programming-based construction of features for machine learning and knowledge discovery tasks. *GP and Evol Machines* 3(4):329–343
- Krogh A, Vedelsby J (1995) Neural network ensembles, cross validation and active learning. *NIPS* 7:231–238
- Kudo M, Sklansky J (2000) Comparison of algorithms that select features for pattern classifiers. *Pattern Recogn* 33:25–41
- Kuncheva LI (2004) Combining pattern classifiers: methods and algorithms. Wiley, Hoboken
- Kushchu I (2002) An evaluation of evolutionary generalization in genetic programming. *Artif Intell Rev* 18(1):3–14
- Lam L, Suen CY (1995) Optimal combination of pattern classifiers. *Pattern Recogn Lett* 16:945–954 See Kuncheva (2004a) p.167
- Landau S, Sigaud O, Schoenauer M (2005) ATNoSFERES revisited. In: *Proceedings of the genetic and evolutionary computation conference GECCO-2005*. ACM, New York, pp 1867–1874
- Langdon W, Gustafson S, Koza J (2009) The genetic programming bibliography. <http://www.cs.bham.ac.uk/~wbl/biblio/>
- Lanzi PL (1999a) Extending the representation of classifier conditions, part I: from binary to messy coding. In: Banzhaf W et al. (eds) *GECCO-99: Proceedings of the genetic and evolutionary computation conference*. Morgan Kaufmann, San Francisco, CA, pp 337–344
- Lanzi PL (1999b) Extending the representation of classifier conditions part II: from messy coding to S-expressions. In: Banzhaf W et al. (eds) *GECCO-99: Proceedings of the genetic and evolutionary computation conference*. Morgan Kaufmann, San Francisco, CA, pp 345–352
- Lanzi PL (2002a) Learning classifier systems from a reinforcement learning perspective. *J Soft Comput* 6(3–4):162–170
- Lanzi PL (2001) Mining interesting knowledge from data with the XCS classifier system. In: Spector L, Goodman ED, Wu A, Langdon WB, Voigt H-M, Gen M, Sen S, Dorigo M, Pezeshk S, Garzon MH, Burke E (eds) *Proceedings of the genetic and evolutionary computation conference (GECCO-2001)*. Morgan Kaufmann, San Francisco, CA, pp 958–965
- Lanzi PL (2008) Learning classifier systems: then and now. *Evolut Intell* 1(1):63–82
- Lanzi PL, Loiacono D, Wilson SW, Goldberg DE (2006a) Classifier prediction based on tile coding. In: *Genetic and evolutionary computation – GECCO-2006*. ACM, New York, pp 1497–1504
- Lanzi PL, Loiacono D, Wilson SW, Goldberg DE (2006b) Prediction update algorithms for XCSF: RLS, Kalman filter and gain adaptation. In: *Genetic and Evolutionary Computation – GECCO-2006*. ACM, New York, pp 1505–1512
- Lanzi PL, Loiacono D, Zanini M (2008) Evolving classifiers ensembles with heterogeneous predictors. In: Bacardit J, Bernadó-Mansilla E, Butz M, Kovacs T, Llorà X, Takadama K (eds) *Learning classifier systems. 10th and 11th international workshops (2006–2007)*, Lecture notes in computer science, vol 4998. Springer, Berlin, pp 218–234
- Lanzi PL, Riolo RL (2000) A roadmap to the last decade of learning classifier system research (from 1989 to 1999). In: Lanzi PL, Stolzmann W, Wilson SW (eds) *Learning classifier systems: from foundations to applications*, Lecture notes in artificial intelligence, vol 1813. Springer, Berlin, pp 33–62
- Lanzi PL, Stolzmann W, Wilson SW (eds) (2000) *Learning classifier systems: from foundations to applications*, Lecture notes in artificial intelligence, vol 1813. Springer, Berlin
- Lanzi PL, Stolzmann W, Wilson SW (eds) (2001) *Advances in learning classifier systems*, Lecture notes in artificial intelligence, vol 1996. Springer, Berlin
- Lanzi PL, Stolzmann W, Wilson SW (eds) (2002) *Advances in learning classifier systems*, Lecture notes in artificial intelligence, vol 2321. Springer, Berlin
- Lanzi PL, Butz MV, Goldberg DE (2007) Empirical analysis of generalization and learning in XCS with gradient descent. In: Lipson H (ed) *Proceedings of the Genetic and evolutionary computation conference, GECCO 2007*, vol 2. ACM, New York, pp 1814–1821
- Lanzi PL, Loiacono D (2006) Standard and averaging reinforcement learning in XCS. In: Cattoico M (ed) *Proceedings of the 8th annual conference on genetic and evolutionary computation, GECCO 2006*. ACM, New York, pp 1480–1496
- Lanzi PL, Loiacono D (2007) Classifier systems that compute action mappings. In: Lipson H (ed) *Proceedings of the Genetic and evolutionary computation conference, GECCO 2007*. ACM, New York, pp 1822–1829
- Lanzi PL, Wilson SW (2006) Using convex hulls to represent classifier conditions. In: Cattoico M (ed) *Proceedings of the genetic and evolutionary computation conference (GECCO 2006)*. ACM, New York, pp 1481–1488
- Liangjie Z, Yanda L (1996) A new global optimizing algorithm for fuzzy neural networks. *Int J Elect* 80(3):393–403

- Linkens DA, Nyongesa HO (1996) Learning systems in intelligent control: an appraisal of fuzzy, neural and genetic algorithm control applications. *IEE Proc Contr Theo Appl* 143(4):367–386
- Liu Y, Yao X (1999) Ensemble learning via negative correlation. *Neural Networ* 12:1399–1404
- Liu Y, Yao X, Higuchi T (2000) Evolutionary ensembles with negative correlation learning. *IEEE Trans Evolut Comput* 4(4):380–387
- Llorà X (2002) Genetic based machine learning using fine-grained parallelism for data mining. PhD thesis, Enginyeria i Arquitectura La Salle. Ramon Llull University
- Llorà X, Garrell JM (2001) Knowledge-independent data mining with fine-grained parallel evolutionary algorithms. In: Spector L, Goodman ED, Wu A, Langdon WB, Voigt H-M, Gen M, Sen S, Dorigo M, Pezeshk S, Garzon MH, Burke E (eds) *Proceedings of the genetic and evolutionary computation conference (GECCO'2001)*. Morgan Kaufmann, San Francisco, CA, pp 461–468
- Llorà X, Sastry K, Goldberg DE (2005a) Binary rule encoding schemes: a study using the compact classifier system. In: Rothlauf F (ed) *Proceedings of the 2005 conference on genetic and evolutionary computation GECCO '05*. ACM Press, New York, pp 88–89
- Llorà X, Sastry K, Goldberg DE (2005b) The compact classifier system: scalability analysis and first results. In: Rothlauf F (ed) *Proceedings of the IEEE congress on evolutionary computation, CEC 2005*. IEEE, Press, Washington, DC, pp 596–603
- Llorà X, Wilson SW (2004) Mixed decision trees: minimizing knowledge representation bias in LCS. In: Kalyanmoy Deb et al. (eds) *Proceedings of the genetic and evolutionary computation conference (GECCO-2004)*, Lecture notes in computer science, Springer, Berlin, pp 797–809
- Loiacono D, Marelli A, Lanzi PL (2007) Support vector regression for classifier prediction. In: *GECCO '07: Proceedings of the 9th annual conference on genetic and evolutionary computation*. ACM, Berlin, pp 1806–1813
- Marmelstein RE, Lamont GB (1998) Pattern classification using a hybrid genetic algorithm – decision tree approach. In: *Genetic programming 1998: proceedings of the 3rd annual conference (GP'98)*. Morgan Kaufmann, San Francisco, CA, pp 223–231
- Marshall JAR, Kovacs T (2006) A representational ecology for learning classifier systems. In: Keijzer M et al. (ed) *Proceedings of the 2006 genetic and evolutionary computation conference (GECCO 2006)*. ACM, New York, pp 1529–1536
- Martin-Bautista MJ, Vila M-A (1999) A survey of genetic feature selection in mining issues. In: *Proceedings of the congress on evolutionary computation (CEC'99)*. IEEE Press, Washington, DC, pp 1314–1321
- Meir R, Rätsch G (2003) An introduction to boosting and leveraging. In: *Advanced lectures on machine learning*. Springer, Berlin, pp 118–183
- Mellor D (2005a) A first order logic classifier system. In: Rothlauf F (ed), *GECCO '05: Proceedings of the 2005 conference on genetic and evolutionary computation*. ACM, New York, pp 1819–1826
- Mellor D (2005b) Policy transfer with a relational learning classifier system. In: *GECCO Workshops 2005*. ACM, New York, pp 82–84
- Mellor D (2008) A learning classifier system approach to relational reinforcement learning. In: Bacardit J, Bernadó-Mansilla E, Butz M, Kovacs T, Llorà X, Takadama K (eds) *Learning classifier systems. 10th and 11th international workshops (2006–2007)*, Lecture notes in computer science, vol 4998. Springer, New York, pp 169–188
- Michalski RS, Moztetic I, Hong J, Lavrac N (1986) The AQ15 inductive learning system: an overview and experiments. Technical Report UIUCDCS-R-86-1260, University of Illinois
- Miller GF, Todd PM, Hegde SU (1989) Designing neural networks using genetic algorithms. In: Schaffer JD (ed) *Proceedings of the 3rd international conference genetic algorithms and their applications*, Morgan Kaufmann, San Francisco, CA, pp 379–384
- Mitra S, Hayashi Y (2000) Neurofuzzy rule generation: survey in soft computing framework. *IEEE Trans Neural Networ* 11(3):748–768
- Morimoto T, Suzuki J, Hashimoto Y (1997) Optimization of a fuzzy controller for fruit storage using neural networks and genetic algorithms. *Eng Appl Artif Intell* 10(5):453–461
- Nolfi S, Miglino O, Parisi D (1994) Phenotypic plasticity in evolving neural networks. In: Gaussier P, Nicoud J-D (eds) *From perception to action*. IEEE Press, Washington, DC, pp 146–157
- O'Hara T, Bull L (2005) A memetic accuracy-based neural learning classifier system. In: *Proceedings of the IEEE congress on evolutionary computation (CEC 2005)*. IEEE Press, Washington, DC, pp 2040–2045
- Ong Y-S, Krasnogor N, Ishibuchi H (eds) (2007) Special issue on memetic algorithms, *IEEE Transactions on Systems, Man and Cybernetics - Part B*
- Ong Y-S, Lim M-H, Neri F, Ishibuchi H (2009) Emerging trends in soft computing - memetic algorithms, *Special Issue of Soft Computing*. vol 13, 8–9
- Ong YS, Lim MH, Zhu N, Wong KW (2006) Classification of adaptive memetic algorithms: A comparative study. *IEEE Trans Syst Man Cybern B* 36(1):141–152

- Opitz D, Maclin R (1999) Popular ensemble methods: an empirical study. *J Artif Intell Res* 11:169–198
- Opitz DW, Shavlik JW (1996) Generating accurate and diverse members of a neural-network ensemble. *Advances in neural information processing systems*, vol 8. Morgan Kaufmann, pp 535–541
- Orriols-Puig A, Bernadó-Mansilla E (2006) Bounding XCS's parameters for unbalanced datasets. In: Keijzer M et al. (eds) *Proceedings of the 2006 genetic and evolutionary computation conference (GECCO 2006)*. ACM, New York, pp 1561–1568
- Orriols-Puig A, Casillas J, Bernadó-Mansilla E (2007a) Fuzzy-UCS: preliminary results. In: Lipson H (ed) *Proceedings of the genetic and evolutionary computation conference, GECCO 2007*. ACM, New York, pp 2871–2874
- Orriols-Puig A, Goldberg DE, Sastry K, Bernadó-Mansilla E (2007b) Modeling XCS in class imbalances: population size and parameter settings. In: Lipson H et al. (eds) *Genetic and evolutionary computation conference, GECCO 2007*. ACM, New York, pp 1838–1845
- Orriols-Puig A, Goldberg DE, Sastry K, Bernadó-Mansilla E (2007c) Modeling XCS in class imbalances: population size and parameter settings. In: Lipson H (eds) *Proceedings of the genetic and evolutionary computation conference, GECCO 2007*. ACM, New York, pp 1838–1845
- Orriols-Puig A, Sastry K, Lanzi PL, Goldberg DE, Bernadó-Mansilla E (2007d) Modeling selection pressure in XCS for proportionate and tournament selection. In: Lipson H (ed) *Proceedings of the genetic and evolutionary computation conference, GECCO 2007*. ACM, New York, pp 1846–1853
- Orriols-Puig A, Bernadó-Mansilla E (2008) Revisiting UCS: description, fitness sharing, and comparison with XCS. In: Bacardit J, Bernadó-Mansilla E, Butz M, Kovacs T, Llorà X, Takadama K (eds) *Learning classifier systems. 10th and 11th international workshops (2006–2007)*, *Lecture notes in computer science*, vol 4998. Springer, Berlin, pp 96–111
- Orriols-Puig A, Casillas J, Bernadó-Mansilla E (2008a) Evolving fuzzy rules with UCS: preliminary results. In: Bacardit J, Bernadó-Mansilla E, Butz M, Kovacs T, Llorà X, Takadama K (eds) *Learning classifier systems. 10th and 11th international workshops (2006–2007)*, *Lecture notes in computer science*, vol 4998. Springer, Berlin, pp 57–76
- Orriols-Puig A, Casillas J, Bernadó-Mansilla E (2008b) Genetic-based machine learning systems are competitive for pattern recognition. *Evolut Intell* 1(3):209–232
- Pal S, Bhandari D (1994) Genetic algorithms with fuzzy fitness function for object extraction using cellular networks. *Fuzzy Set Syst* 65(2–3):129–139
- Pappz GL, Freitas AA (2010) Automating the design of data mining algorithms. *An evolutionary computation approach*. Natural computing series. Springer
- Paris G, Robilliard D, Fonlupt C (2001) Applying boosting techniques to genetic programming. In: *Artificial evolution 2001*, *Lecture notes in computer science*, vol 2310. Springer, Berlin, pp 267–278
- Pereira FB, Costa E (2001) Understanding the role of learning in the evolution of busy beaver: a comparison between the Baldwin effect and Lamarckian strategy. In: *Proceedings of the genetic and evolutionary computation conference (GECCO-2001)*. Morgan Kaufmann, San Francisco, pp 884–891
- Perneel C, Themlin J-M (1995) Optimization of fuzzy expert systems using genetic algorithms and neural networks. *IEEE Trans Fuzzy Syst* 3(3):301–312
- Pham DT, Karaboga D (1991) Optimum design of fuzzy logic controllers using genetic algorithms. *J Syst Eng* 1:114–118
- Poli R, Langdon WB, McPhee NF (2008) A field guide to genetic programming, freely available at <http://www.gp-field-guide.org.uk.lulu.com>
- Punch WF, Goodman ED, Pei M, Chia-Shun L, Hovland P, Enbody R (1993) Further research on feature selection and classification using genetic algorithms. In: Forrest S (ed) *Proceedings of the 5th international conference on genetic algorithms (ICGA93)*. Morgan Kaufmann, San Francisco, CA, pp 557–564
- Radi A, Poli R (2003) Discovering efficient learning rules for feedforward neural networks using genetic programming. In: Abraham A, Jain L, Kacprzyk J (eds) *Recent advances in intelligent paradigms and applications*. Springer, Berlin, pp 133–159
- Raymer ML, Punch WF, Goodman ED, Kuhn LA, Jain AK (2000) Dimensionality reduction using genetic algorithms. *IEEE Trans Evolut Comput* 4(2): 164–171
- Reeves CR, Rowe JE (2002) Genetic algorithms – principles and perspectives. A guide to GA theory. Kluwer, Norwell
- Riolo RL (1987) Bucket brigade performance: I. long sequences of classifiers. In: Grefenstette JJ (eds) *Proceedings of the 2nd international conference on genetic algorithms (ICGA'87)*, Lawrence Erlbaum Associates, Cambridge, MA, pp 184–195
- Rivest RL (1987) Learning decision lists. *Mach Learn* 2(3):229–246
- Romaniuk S (1994) Towards minimal network architectures with evolutionary growth networks. In: *Proceedings of the 1993 international joint conference on neural networks*, vol 3. IEEE Press, Washington, DC, pp 1710–1713



- Rouwhorst SE, Engelbrecht AP (2000) Searching the forest: using decision trees as building blocks for evolutionary search in classification databases. In: Proceedings of the 2000 congress on evolutionary computation (CEC00). IEEE Press, Washington, DC, pp 633–638
- Rozenberg G, Bäck T, Kok J (eds) (2012) Handbook of natural computing. Springer, Berlin
- Ruta D, Gabrys B (2001) Application of the evolutionary algorithms for classifier selection in multiple classifier systems with majority voting. In: Kittler J, Roli F (eds) Proceedings of the 2nd international workshop on multiple classifier systems, Lecture notes in computer science, vol 2096. Springer, Berlin, pp 399–408. See Kuncheva (2004a) p.321
- Sánchez L, Couso I (2007) Advocating the use of imprecisely observed data in genetic fuzzy systems. IEEE Trans Fuzzy Syst 15(4):551–562
- Sasaki T, Tokoro M (1997) Adaptation toward changing environments: why Darwinian in nature? In: Husbands P, Harvey I (eds) Proceedings of the 4th European conference on artificial life. MIT Press, Cambridge, MA, pp 145–153
- Saxon S, Barry A (2000) XCS and the Monk's problems. In: Lanzi PL, Stolzmann W, Wilson SW (eds) Learning classifier systems: from foundations to applications, Lecture notes in artificial intelligence, vol 1813. Springer, Berlin, pp 223–242
- Schaffer C (1994) A conservation law for generalization performance. In: Hirsh H, Cohen WW (eds) Machine learning: proceedings of the eleventh international conference. Morgan Kaufmann, San Francisco, CA, pp 259–265
- Schaffer JD (ed) (1989) Proceedings of the 3rd international conference on genetic algorithms (ICGA-89), George Mason University, June 1989. Morgan Kaufmann, San Francisco, CA
- Schmidhuber J (1987) Evolutionary principles in self-referential learning. (On learning how to learn: The meta-meta-... hook.). PhD thesis, Technische Universität München, Germany
- Schuermans D, Schaeffer J (1989) Representational difficulties with classifier systems. In: Schaffer JD (ed) Proceedings of the 3rd international conference on genetic algorithms (ICGA-89). Morgan Kaufmann, San Francisco, CA, pp 328–333
- Sharkey AJC (1996) On combining artificial neural nets. Connection Sci 8(3–4):299–313
- Sharpe PK, Glover RP (1999) Efficient GA based techniques for classification. Appl Intell 11:277–284
- Sirlantzis K, Fairhurst MC, Hoque MS (2001) Genetic algorithms for multi-classifier system configuration: a case study in character recognition. In: Kittler J, Roli F (eds) Proceedings of the 2nd international workshop on multiple classifier systems, Lecture notes in computer science, vol 2096. Springer, Berlin, pp 99–108. See Kuncheva (2004a) p.321
- Smith JE (2007) Coevolving memetic algorithms: a review and progress report. IEEE Trans Syst Man Cybern B Cybern 37(1):6–17
- Smith MG, Bull L (2005) Genetic programming with a genetic algorithm for feature construction and selection. GP and Evol Machines 6(3):265–281
- Smith RE (1992) A report on the first international workshop on learning classifier systems (IWLC-92). NASA Johnson Space Center, Houston, Texas, Oct. 6–9. <http://lumpi.informatik.uni-dortmund.de/pub/LCS/papers/lcs92.ps.gz> or from ENCORE, The Electronic Appendix to the Hitch-Hiker's Guide to Evolutionary Computation (<http://ftp.krl.caltech.edu/pub/EC/Welcome.html>) in the section on Classifier Systems
- Smith RE (1994) Memory exploitation in learning classifier systems. Evolut Comput 2(3):199–220
- Smith RE, Cribbs HB (1994) Is a learning classifier system a type of neural network? Evolut Comput 2(1):19–36
- Smith RE, Goldberg DE (1991) Variable default hierarchy separation in a classifier system. In: Rawlins GJE (ed) Proceedings of the first workshop on foundations of genetic algorithms. Morgan Kaufmann, San Mateo, pp 148–170
- Smith RE, Cribbs III HB (1997) Combined biological paradigms. Robot Auton Syst 22(1):65–74
- Song D, Heywood MI, Zincir-Heywood AN (2005) Training genetic programming on half a million patterns: an example from anomaly detection. IEEE Trans Evolut Comput 9(3):225–239
- Srinivas N, Deb K (1994) Multi-objective function optimization using non-dominated sorting genetic algorithm. Evolut Comput 2(3):221–248
- Stagge P (1998) Averaging efficiently in the presence of noise. In: Parallel problem solving from nature, vol 5. pp 188–197
- Stolzmann W (1996) Learning classifier systems using the cognitive mechanism of anticipatory behavioral control, detailed version. In: Proceedings of the first European workshop on cognitive modelling. Berlin, TU, pp 82–89
- Stone C, Bull L (2003) For real! XCS with continuous-valued inputs. Evolut Comput 11(3):298–336
- Storn R, Price K (1996) Minimizing the real functions of the ICEC'96 contest by differential evolution. In: Proceedings of the IEEE international conference Evolutionary Computation. IEEE Press, Washington, DC, pp 842–844
- Stout M, Bacardit J, Hirst JD, Krasnogor N (2008) Prediction of recursive convex hull class assignment for protein residues. Bioinformatics 24(7):916–923

- Sutton RS (1986) Two problems with backpropagation and other steepest-descent learning procedures for networks. In: *Proceedings of the 8th annual conference cognitive science society*. Erlbaum, pp 823–831
- Sziran yi T (1996) Robustness of cellular neural networks in image deblurring and texture segmentation. *Int J Circuit Theory App* 24(3):381–396
- Tamaddoni-Nezhad A, Muggleton SH (2000) Searching the subsumption lattice by a genetic algorithm. In: Cussens J, Frisch A (eds) *Proceedings of the 10th international conference on inductive logic programming*. Springer, Berlin, pp 243–252
- Tamaddoni-Nezhad A, Muggleton S (2003) A genetic algorithms approach to ILP. In: *Inductive logic programming, Lecture notes in computer science*, vol 2583. Springer, Berlin, pp 285–300
- Tharakannel K, Goldberg D (2002) XCS with average reward criterion in multi-step environment. Technical report, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign
- Thompson S (1998) Pruning boosted classifiers with a real valued genetic algorithm. In: *Research and development in expert systems XV – proceedings of ES'98*. Springer, Berlin, pp 133–146
- Thompson S (1999) Genetic algorithms as postprocessors for data mining. In: *Data mining with evolutionary algorithms: research directions – papers from the AAAI workshop, Tech report WS-99-06*. AAAI Press, Menlo Park, CA, pp 18–22
- Thrft P (1991) Fuzzy logic synthesis with genetic algorithms. In: Booker LB, Belew RK (eds) *Proceedings of 4th international conference on genetic algorithms (ICGA'91)*. Morgan Kaufmann, San Francisco, CA, pp 509–513
- Tomlinson A (1999) *Corporate classifier systems*. PhD thesis, University of the West of England
- Tomlinson A, Bull L (1998) A corporate classifier system. In: Eiben AE, Bäck T, Shoenauer M, Schwefel H-P (eds) *Proceedings of the fifth international conference on parallel problem solving from nature – PPSN V, Lecture notes in computer science*, vol 1498. Springer, Berlin, pp 550–559
- Tomlinson A, Bull L (2002) An accuracy-based corporate classifier system. *J Soft Comput* 6(3–4):200–215
- Tran TH, Sanza C, Duthen Y, Nguyen TD (2007) XCSF with computed continuous action. In: *Genetic and evolutionary computation conference (GECCO 2007)*. ACM, New York, pp 1861–1869
- Tumer K, Ghosh J (1996) Analysis of decision boundaries in linearly combined neural classifiers. *Pattern Recogn* 29(2):341–348
- Turney P (1996) How to shift bias: lessons from the Baldwin effect. *Evolut Comput* 4(3):271–295
- Valentini G, Masulli F (2002) Ensembles of learning machines. In: *WIRN VIETRI 2002: Proceedings of the 13th Italian workshop on neural nets-revised papers*. Springer, Berlin, pp 3–22
- Valenzuela-Rendón M (1989) Two analysis tools to describe the operation of classifier systems. PhD thesis, University of Alabama. Also TCGA technical report 89005
- Valenzuela-Rendón M (1991) The fuzzy classifier system: a classifier system for continuously varying variables. In: Booker LB, Belew RK (eds) *Proceedings of the 4th international conference on genetic algorithms (ICGA'91)*. Morgan Kaufmann, San Francisco, CA, pp 346–353
- Valenzuela-Rendón M (1998) Reinforcement learning in the fuzzy classifier system. *Expert Syst Appl* 14:237–247
- Vallim R, Goldberg D, Llorà X, Duque T, Carvalho A (2003) A new approach for multi-label classification based on default hierarchies and organizational learning. In: *Proceedings of the genetic and evolutionary computation conference, workshop sessions: learning classifier systems*. ACM, New York, pp 2017–2022
- Vanneschi L, Poli R (2012) Genetic programming: introduction, applications, theory and open issues. In: Rozenberg G, Bäck T, Kok J (eds) *Handbook of natural computing*. Springer, Berlin
- Venturini G (1993) SIA: a supervised inductive algorithm with genetic search for learning attributes based concepts. In: Brazdil PB (ed) *ECML-93 - Proceedings of the European conference on machine learning*. Springer, Berlin, pp 280–296
- Vilalta R, Drissi Y (2002) A perspective view and survey of meta-learning. *Artif Intell Rev* 18(2):77–95
- Wada A, Takadama K, Shimohara K, Katai O (2005c) Learning classifier systems with convergence and generalization. In: Bull L, Kovacs T (eds) *Foundations of learning classifier systems*. Springer, Berlin, pp 285–304
- Wada A, Takadama K, Shimohara K (2005a) Counter example for Q-bucket-brigade under prediction problem. In: *GECCO Workshops 2005*. ACM, New York, pp 94–99
- Wada A, Takadama K, Shimohara K (2005b) Learning classifier system equivalent with reinforcement learning with function approximation. In: *GECCO Workshops 2005*. ACM, New York, pp 92–93
- Wada A, Takadama K, Shimohara K (2007) Counter example for Q-bucket-brigade under prediction problem. In: Kovacs T, LLòra X, Takadama K, Lanzi PL, Stolzmann W, Wilson SW (eds) *Learning classifier systems. International workshops, IWLCS 2003-2005, revised selected papers, Lecture notes in computer science*, vol 4399. Springer, Berlin, pp 128–143
- Whitley D, Goldberg D, Cantú-Paz E, Spector L, Parmee I, Beyer HG (eds) (2000) *Proceedings of the genetic and*



- evolutionary computation conference (GECCO-2000). Morgan Kaufmann, San Francisco, CA
- Whiteson S, Stone P (2006) Evolutionary function approximation for reinforcement learning. *J Mach Learn Res* 7:877–917
- Whitley D, Starkweather T, Bogart C (1990) Genetic algorithms and neural networks: optimizing connections and connectivity. *Parallel Comput* 14(3):347–361
- Whitley D, Gordon VS, Mathias K (1994) Lamarckian evolution, the Baldwin effect and function optimization. In: *Parallel problem solving from nature (PPSN-III)*. Springer, Berlin, pp 6–15
- Wilcox JR (1995) Organizational learning within a learning classifier system. Master's thesis, University of Illinois. Also Technical Report No. 95003 IlliGAL
- Wilson SW (2001a) Mining oblique data with XCS. In: Lanzi PL, Stolzmann W, Wilson SW (eds) *Advances in learning classifier systems, third international workshop, IW LCS 2000*, Lecture notes in computer science, vol 1996. Springer, Berlin, pp 158–176
- Wilson SW (1989) Bid competition and specificity reconsidered. *Complex Syst* 2:705–723
- Wilson SW (1994) ZCS: a zeroth level classifier system. *Evolut Comput* 2(1):1–18. <http://prediction-dynamics.com/>
- Wilson SW (1995) Classifier fitness based on accuracy. *Evolut Comput* 3(2):149–175. <http://prediction-dynamics.com/>
- Wilson SW (1998) Generalization in the XCS classifier system. In: Koza JR, Banzhaf W, Chellapilla K, Deb K, Dorigo M, Fogel DB, Garzon MH, Goldberg DE, Iba H, Riolo R (eds) *Genetic programming 1998: proceedings of the third annual conference*, Morgan Kaufmann, San Francisco, CA, pp 665–674. <http://prediction-dynamics.com/>
- Wilson SW (1999) Get real! XCS with continuous-valued inputs. In: Booker L, Forrest S, Mitchell M, Riolo RL (eds) *Festschrift in honor of John H. Holland. Center for the Study of Complex Systems*. pp 111–121. <http://prediction-dynamics.com/>
- Wilson SW (2000) Mining oblique data with XCS. In: *Proceedings of the international workshop on learning classifier systems (IW LCS-2000)*, in the joint workshops of SAB 2000 and PPSN 2000. Extended abstract
- Wilson SW (2001b) Function approximation with a classifier system. In: Spector L, Goodman ED, Wu A, Langdon WB, Voigt HM, Gen M, Sen S, Dorigo M, Pezeshek S, Garzon MH, Burke E (eds) *Proceedings of the genetic and evolutionary computation conference (GECCO-2001)*. Morgan Kaufmann, San Francisco, CA, pp 974–981
- Wilson SW (2002a) Classifiers that approximate functions. *Natural Comput* 1(2–3):211–234
- Wilson SW (2002b) Compact rulesets from XCSI. In: Lanzi PL, Stolzmann W, Wilson SW (eds) *Advances in learning classifier systems, Lecture notes in artificial intelligence*, vol 2321. Springer, Berlin, pp 196–208
- Wilson SW (2007) Three architectures for continuous action. In: Kovacs T, LLòra X, Takadama K, Lanzi PL, Stolzmann W, Wilson SW (eds) *Learning classifier systems. International workshops, IW LCS 2003–2005*, revised selected papers, Lecture notes in computer science, vol 4399. Springer, Berlin, pp 239–257
- Wilson SW (2008) Classifier conditions using gene expression programming. In: Bacardit J, Bernadó-Mansilla E, Butz M, Kovacs T, LLòra X, Takadama K (eds) *Learning classifier systems. 10th and 11th international workshops (2006–2007)*, Lecture notes in computer science, vol 4998. Springer, Berlin, pp 206–217
- Wilson SW, Goldberg DE (1989) A critical review of classifier systems. In: Schaffer JD (ed) *Proceedings of the 3rd international conference on genetic algorithms*. Morgan Kaufmann, San Francisco, CA, pp 244–255. <http://prediction-dynamics.com/>
- Wolpert DH (1996) The lack of a priori distinctions between learning algorithms. *Neural Comput* 8(7):1341–1390
- Wong ML, Leung KS (2000) Data mining using grammar based genetic programming and applications. Kluwer, Norwell
- Woods K, Kegelmeyer W, Bowyer K (1997) Combination of multiple classifiers using local accuracy estimates. *IEEE Trans Pattern Anal Mach Intell* 19:405–410
- Woodward JR (2003) GA or GP? That is not the question. In: *Proceedings of the 2003 congress on evolutionary computation, CEC2003*. IEEE Press, Washington DC, pp 1056–1063
- Yamasaki K, Sekiguchi M (2000) Clear explanation of different adaptive behaviors between Darwinian population and Lamarckian population in changing environment. In: *Proceedings of the fifth international symposium on artificial life and robotics*. pp 120–123
- Yao X (1999) Evolving artificial neural networks. *Proc IEEE* 87(9):1423–1447
- Yao X, Islam MM (2008) Evolving artificial neural network ensembles. *IEEE Comput Intell Mag* 3(1):31–42
- Yao X, Liu Y (1997) A new evolutionary system for evolving artificial neural networks. *IEEE Trans Neural Network* 8:694–713
- Yao X, Liu Y (1998) Making use of population information in evolutionary artificial neural networks. *IEEE Trans Syst Man Cybern B* 28(3):417–425
- Zatuchna ZV (2005) AgentP: a learning classifier system with associative perception in maze environments. PhD thesis, University of East Anglia

Zatuchna ZV (2004) AgentP model: Learning Classifier System with Associative Perception. In 8th parallel problem solving from nature international conference (PPSN VIII). pp 1172–1182

Zhang B-T, Veenker G (1991) Neural networks that teach themselves through genetic discovery of

novel examples. In: Proceedings 1991 IEEE international joint conference on neural networks (IJCNN'91) vol 1. IEEE Press, Washington DC, pp 690–695