Genetics-based Machine Learning

Tim Kovacs
Department of Computer Science
University of Bristol

This is the author's final version (dated April 2009) of a chapter to appear in Springer Verlag's *Handbook of Natural Computing* in 2010. The published version will be available at www.springerlink.com. In this version 3.5 *Learning Classifier Systems* had a new subsubsection 3.5.2 *Representation* and several other subsubsections were moved into the new one. This version also has a more detailed table of contents and its bibliography lists the pages each reference is cited on. Any other differences are due to the proofreader. By default please cite the published version:

- Tim Kovacs. Genetics-based Machine Learning. To appear in Grzegorz Rozenberg, Thomas Bäck, and Joost Kok, editors, *Handbook of Natural Computing: Theory, Experiments, and Applications.* Springer Verlag, 2010.

Abstract

This is a survey of the field of Genetics-based Machine Learning (GBML): the application of evolutionary algorithms to machine learning. We assume readers are familiar with evolutionary algorithms and their application to optimisation 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 optimisation 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 sub-problems of learning, genetic programming, evolving ensembles, evolving neural networks, learning classifier systems, and genetic fuzzy systems.

Contents

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 optimisation problems. In this introductory section we outline the scope of machine learning, introduce the more specific area of supervised learning, and contrast it with optimisation. 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. Section §2 describes a framework for GBML which includes ways of classifying GBML algorithms and a discussion of the interaction between learning and evolution. Section §3 reviews the work of a number of GBML communities with emphasis on their evolutionary aspects. Finally, section §4 concludes.

What's 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 [311] and the *conservation law of generalisation* [245]). 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), co-evolution, and comparisons between GBML and non-evolutionary alternatives. However, [94] contains good treatments of GBML for, among others, clustering and data preparation.

1.1 Machine Learning

Machine learning is concerned with machines which improve with experience and reason inductively or abductively in order to: optimise, approximate, summarise, generalise 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 optimisation problems.

Inductive Generalisation Inductive generalisation 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 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 maximising 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 optimisation and learning. We will not deal with abduction.

1-Max: a Typical Optimisation Problem The 1-max problem is to maximise 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 (colour, 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 maximises 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 labelled input/output pairs. Inputs are typically called instances, examples 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 Optimisation 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 generalisation of the induced hypothesis from the train set to the previously unused test set. In contrast, in optimisation we can typically can 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 non-stationary 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 unlabelled 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 [94] §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 §2.3.

Epistasis There is some evidence the accuracy of GBML methods may not suffer from epistasis as much as typical non-evolutionary greedy search [94] §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 §3.5.

Adapting Bias GBML methods are well-suited to adapting inductive bias. We can adapt representational bias by e.g. selecting rule condition shapes §3.5.2, and algorithmic bias by e.g. evolving learning rules §3.4.

Exploiting Diversity We can exploit the diversity of a population of solutions to combine and improve predictions (the ensemble approach §3.3) and to generate Pareto sets for multiobjective problems.

Dynamic Adaptation All the above can be done dynamically, to improve accuracy, to deal with non-stationarity, and to minimise population size. This last is of interest in order to reduce overfitting, improve run-time and improve human-readability.

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

Parallelisation Population-based search is easily parallelised.

Suitable Problem Characteristics From an optimisation perspective, learning problems are typically large, non-differentiable, noisy, epistatic, deceptive, and multimodal [207]. To this list we could add high-dimensional and highly constrained. EAs are a good choice for such problems.

See [61] and §3.4 for more arguments in favour of GBML. At the same time there are arguments against using GBML.

Algorithmic Complexity GBML algorithms are typically more complex than their non-evolutionary alternatives. This makes them harder to implement, harder to analyse, and means there is less theory to guide parameterisation 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 SWOT (Strengths, Weaknesses, Opportunities, Threats) analysis of GBML in [224] 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 categorisations. In the first (§2.1), GBML systems are classified by their role in learning; specifically their application to i) sub-problems of machine learning, ii) learning itself, or iii) meta-learning. In the second categorisation (§2.2), GBML systems are classified by their high-level algorithmic approach as either Pittsburgh or Michigan systems. Following this, in §2.3 we briefly review ways in which learning and evolution interact and in §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 backpropagation, 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 §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 §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 overemphasised; 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 meta-learning role. Furthermore, both contribute to the ultimate goal of adaptation, and in §2.3 we will see ways in which they interact.

Evolution as learning is illustrated in the left of figure 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*

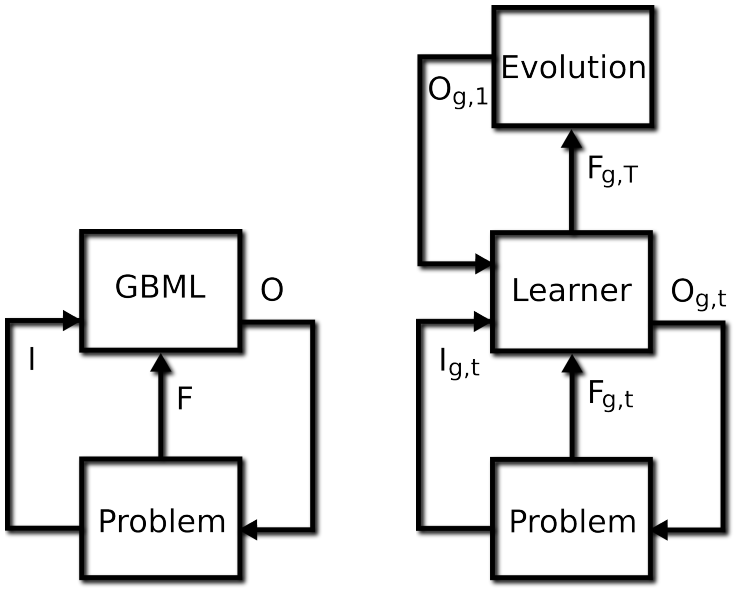


Figure 1: Left: GBML as learner. **I**nput, **O**utput and **F**itness shown. Right: GBML as meta-learner. Subscripts denote **g**eneration and **t**ime step (1 . . . T)

as any learner can be augmented by GBML, and is related to the wrapper approach to feature selection in 3.1.

Meta-learning is a broad term with different interpretations but the essential idea is *learning about learning*. A meta-learner may optimise parameters of a learner, learn which learner to apply to a given input or a given problem, learn which representation(s) to use, optimise 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 [288, 107] on non-evolutionary meta-learning and [43, 162, 163, 42] on the hyperheuristics (*heuristics to learn heuristics*) approach, of which a subset is evolutionary.

A third role for evolution is application to various sub-problems of learning including feature selection, feature construction, and other optimisation 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 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, complemen-

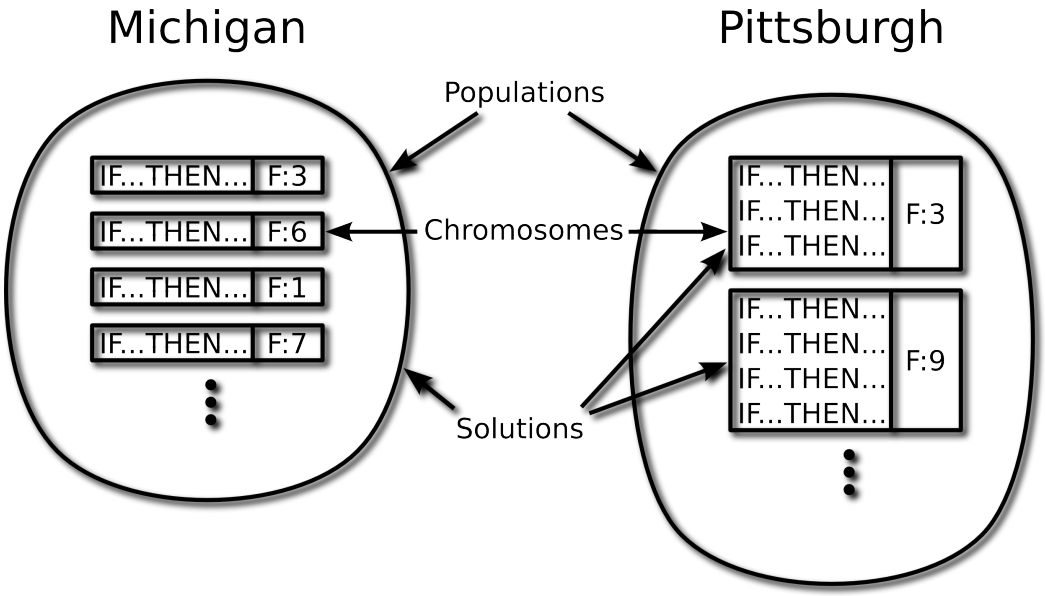


Figure 2: Michigan and Pittsburgh rule-based systems compared. The F:x associated with each chromosome indicates its fitness

tarity, and coverage of 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 [94]. To illustrate, figure 2 depicts the representation used by Pitt and Michigan versions of the rule-based systems called Learning Classifier Systems (see §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. [297].

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.[1] Additionally, since their chromosomes are more complex so are their genetic operators. On the other hand they face less complex credit assignment problems and are hence 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

[1] See, however, §3.5.3 on the windowing approach to improving run-time and [9] for an approach which avoids performing matching operations between rule conditions and irrelevant features.

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

Figure 3: A basic Michigan algorithm

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 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 figure 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 on-line. The Michigan approach has mainly been used with LCS. See [110, 141, 297, 95, 154] 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 alters the co-evolutionary dynamics of the system. The output of multiple runs is combined to produce the solution. The approach originated with SIA (Supervised Inductive Algorithm) [287, 190], a supervised genetic rule learner.

Genetic Cooperative-Competitive Learning GCCL is another Michigan approach in which on 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 remaining rules' collective accuracy 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 [110, 111], REGAL [104], and LOGENPRO [312].

2.3 The Interaction of Learning and Evolution

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

Memetic Learning We can characterise 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 generally outperforms either alone [316]. 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 [92]. Methods which combine global and local search are called *memetic* algorithms [117, 118, 214, 212, 252, 213, 240]. See [161] for a self-contained tutorial.

Darwinian and Lamarckian Evolution In Lamarckian evolution/inheritance, learning during an individual's lifetime directly alters 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 [2] showed Lamarckian evolution was much faster on stationary learning tasks but Saski and Tokoro [243] showed Darwinian evolution is generally better on non-stationary tasks. See also [296, 315, 228, 293].

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 allows 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 [293] §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 behaviour can become instinctive. This allows learned behaviours to become inherited without Lamarckian inheritance.

Turney [280] 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 behaviour.

Evaluating Evolutionary Search by Evaluating Accuracy Kovacs and Kerber [157] point out that high classification accuracy does not imply effective genetic search. To illustrate, they initialised XCS [301] 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 [293] allocated trials to chromosomes in proportion to their fitness with the following procedure. At each new generation evaluate each chromosome once only. Allocate subsequent evaluations using a softmax distribution based on the initial fitnesses and recalculate the average fitness of a chromosome after each evaluation. In non-stationary problems a recency-weighted average of fitness samples is used. They call this approach *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 optimising noisy fitness functions [262, 19], 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 minimise disruption to the population, which is useful for on-line 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 co-evolve 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 NNs and fuzzy rules, and some are powered by GP. Nonetheless, the differences between the communities and their approaches is such that it seemed most useful to structure this section by community and not e.g. 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 Sub-problems of Learning

This section briefly reviews ways in which evolution has been used not for the primary task of learning – generating hypotheses – but for sub-problems including data preparation and optimisation 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 [143] in which the base learner (the one which generates hypotheses) is treated as a black box to be optimised by a search algorithm. In this EAs usually give good results compared to non-evolutionary methods [139, 250, 166] but there are exceptions [139]. In [62], 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 e.g. weighted k-nearest neighbours [234]. The main drawback of EAs for feature selection is their slowness compared to non-evolutionary methods. See [201, 94, 95] for overviews and [266, 11] 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 itself or we can preprocess data to construct informative new features by combining existing ones e.g. new feature $f_{\text{new}} = f_1$ AND f_3 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 e.g. [131, 164, 253]. The original features have also been linearly transformed by evolving a vector of coefficients [145, 232]. Simultaneous feature transformation and selection has had good results [234].

Other Sub-problems of Learning EAs have been used in a variety of other ways. One is training set optimisation in which we can partition the data into training sets [238], select the most useful training inputs [137], and even generate synthetic inputs [322, 69]. EAs have also been used for optimisation within a learner e.g. [145] optimised weighted k-nearest neighbours with a GA, [61] optimised decision tree tests using a GA and an Evolution Strategy and [272, 273] optimised voting weights in an ensemble. [141] replaced beam

A	B	C	Class
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

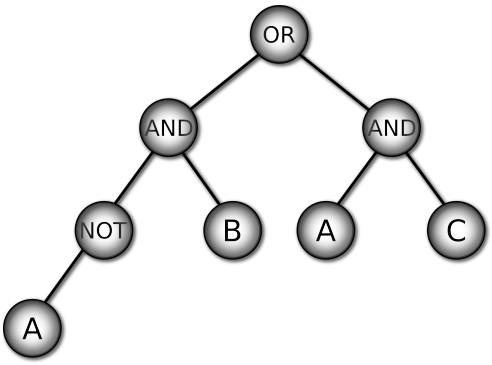


Figure 4: Two representations of the 3 multiplexer function: truth table (left) and GP tree (right)

search in AQ with a genetic algorithm and similarly [269, 82, 80, 81, 270] have investigated Inductive Logic Programming driven by a GA.

3.2 Genetic Programming

Genetic Programming (GP) is a major evolutionary paradigm which evolves programs [286]. 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 [314] for discussion. [94] 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 first-order logic, in GP, ii) it is easier to represent complex concepts compactly, and iii) GP is good at finding novel, 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 generalise well, and iii) variable-length representations suffer from bloat (see e.g. [231]).

While GAs are typically applied to function optimisation, GP is widely applied to learning. To illustrate, of the set of "typical GP problems" defined by Koza [158], which have become more-or-less agreed benchmarks for the GP community [286], 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 classification of 2-dimensional points as belonging to one of 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 upwards from leaves though the functions in the non-leaf nodes and iii) output the value of the root (top) node as the classification.

$x^2 + 2y^2 - 13$

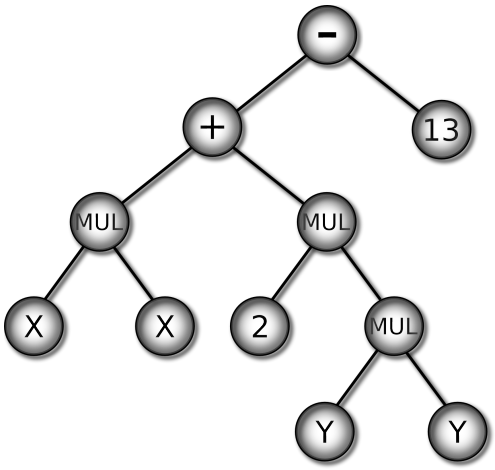


Figure 5: Two representations of a real-valued function

A	B	C	Class
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

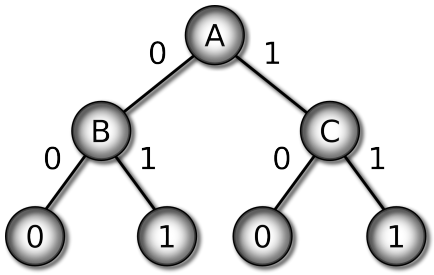


Figure 6: Two representations of the 3 multiplexer function: truth table (left) and decision tree (right)

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 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. [239] 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 and more expressive but have a larger search space. See [31].

Evolving Individual Nodes in Decision Trees In most GP-based tree evolvers an individual is a complete tree but in [199] 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 §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 [93, 135] and the boosting approach in [260]. Although not an ensemble technique, the Limited Error Fitness method introduced in [100] 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 has improved accuracy by building an ensemble of GP trees. In [148, 227] each run adds one tree to the ensemble and weights are computed with standard boosting.

GP Hyperheuristics Schmidhuber [247] 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 [160, 162, 163]. Freitas [94] §12.2.3 sketches a similar approach which he calls *algorithm induction* while in [226] Pappa and Freitas go into the subject at length. [41] also deals 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 [124] in which *brittleness* refers to overfitting or poor generalisation to unseen cases, and *robustness* refers to good generalisation. A feature of the GP literature is that GP is usually evaluated only on the training set [168, 286]. Kushchu has also criticised the way in which test sets have been used [168]. Nonetheless GP has the same need for test sets to evaluate generalisation as other methods [168] and as a result the ability of GP to perform inductive generalisation is one of the open issues for GP identified in [286]. See [168, 286] for methods which have been used to encourage generalisation in GP, many of which can be applied to other methods.

Reading See Koza's 1994 book [159] for the basics of evolving decision trees with GP, Wong and Leung's 2000 book on data mining with grammar-based GP [312], Freitas' 2002 book [94] for a good introduction to GP, decision trees and both evolutionary and non-evolutionary learning, Poli, Langdon and McPhee's free 2008 GP book [231], and Vanneschi and Poli's 2010 survey of GP [286]. The GP bibliography has over 5000 entries [171].

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 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 [36] but see e.g. [313, 71, 68] for examples. Ensembles some enjoy good theoretical foundations [36, 279], perform very well in practice [63] and were identified by Dietterich as one of four current directions for machine learning in 1998 [79]. While the key advantage of using ensembles is better test set generalisation 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 [249].

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 [32, 33] and boosting [96, 97, 202].

Creating a good ensemble is an inherently multiobjective problem [281]. In addition to maximising accuracy we also want to maximise 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 [165]. Hence we want to create accurate predictors with diverse errors [79, 116, 165, 216, 215] In addition we may want to minimise ensemble size in order to reduce run time and to make the ensemble easier to understand. Finally, evolving variable-length chromosomes without pressure towards parsimony results in bloat [231], in which case we have a reason to minimise 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:* [169, 272, 273, 70] 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 §3.1 and [167] §8.1.4). *v) Data selection:* generating diverse classifiers by train-

ing on different data (see §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, [191] combines accuracy and diversity into a single objective. In comparison, multi-objective evolutionary ensembles are rare [68] but they are starting to appear e.g. [1, 68, 67]. In addition to upgrading GBML to multi-objective GBML other measures can be taken to evolve diversity, for example fitness sharing e.g. [192] and the co-evolutionary fitness method we describe next. Gagné et al. [99] compare boosting and co-evolution of learners and problems – both gradually focus on cases which are harder to learn – and argue that co-evolution is less likely to overfit noise. Their co-evolution 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 EEL (Evolving Ensemble Learning) approach dominates their on-line version as the latter lacks a way to remove bad classifiers. Good results were obtained compared to Adaboost on 6 UCI [8] 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 populations, 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 [99] §3.2. See e.g. [251, 241] for evolutionary approaches.

3.3.2 Conclusions

Research directions for evolutionary ensembles include multi-objective evolution [68], hybrid ensembles [68] and minimising ensemble complexity [192].

Reading Key works include Opitz and Shavlik's classic 1996 paper on evolving NN ensembles [216], Kuncheva's 2004 book on ensembles [167], Chandra and Yao's 2006 discussion of multi-objective evolution of ensembles [67], Yao and Islam's 2008 review of evolving NN ensembles [317] and Brown's 2005 and 2010 surveys of ensembles [36, 34]. We cover evolving NN ensembles in §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 EANNs (Evolving Artificial Neural Networks) [316] or ECoSs (Evolving Connectionist Systems) [147].

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 backpropagation. The *architecture* of a NN refers to the set of nodes, connections, activation functions and the plasticity of nodes; 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 representation have been used: i) direct encoding [316, 92] in which all details (connections and nodes) are specified, ii) indirect encoding [316, 92] 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 [92] in which a developmental process is genetically encoded [149, 114, 210, 134, 225, 268]. 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: [5, 256, 259]. 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 specialise? Two options are to encourage diversity during evolution, e.g. with fitness sharing, or after evolution, by pruning redundant nodes [5].

Adapting Weights Most NN learning rules are based on gradient descent, including the best known: backpropagation (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 [267, 294].

An alternative is to evolve the weights which has the advantages that EAs don't 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, higher order), which is a great convenience for the engineer [316]. Consequently, much research has been done on evolution of weights. Unsurprisingly fitness functions penalise NN error but they also typically penalise 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 depends on the problem [316]. However, Yao [316] states that evolving weights AND architecture is better than evolving weights alone and that evolution seems better for reinforcement learning and recurrent networks. Floreano [92] suggests evolution is better for dynamic networks. Happily we don't 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 [316] we should evolve initial weights and then train them with gradient descent. [92] claims 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 [6]. Another alternative is to evolve architectures. Miller et al. [207] make the following suggestions (quoted from [316]) as to why EAs should be suitable for searching the space of architectures.

1. The surface is infinitely large since the number of possible nodes and connections is unbounded;
2. 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;
3. 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;
4. the surface is deceptive since similar architectures may have quite different performance;
5. 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 [318]. 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-optimise other parameters of the network [92] at the same time. For example, [20] 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-optimising architecture and weights evolution would not have been able to take all factors into account at the same time.

Evolving Learning Rules 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 don't a priori know 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 specialised 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 [316] 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 [65], working with multi-layer perceptrons, found evolving the architecture, initial weights and rule parameters together as good or better than evolving only first two or the third.

We can also evolve new learning rules [316, 233]. Open-ended evolution of rules was initially considered impractical and instead Chalmers [66] 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 [77]. 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 [233]. 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 [317] but a population of evolving NNs is naturally treated as an ensemble and recent work has begun to do so. Evolving NNs is inherently multi-objective: we want accurate yet simple and diverse networks. Some work combines these objectives into one fitness function while others is explicitly multi-objective.

Single-objective Ensembles Yao [319] used EPNet's [318] 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 [191] pursued accuracy and diversity in two ways. The first was to modify backpropagation to minimise error and maximise 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. EENCL (Evolutionary Ensembles for NCL) [192] automatically determines the size of an ensemble. It encourages diversity with fitness sharing and NCL and it deals with the ensemble member selection problem §3.3.1 with a cluster-and-select method (see [142]). 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 small than the population. CNNE (Cooperative Neural Net Ensembles) [138] 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.

Multi-objective Ensembles MPANN (Memetic Pareto Artificial NN) [1] was the first ensemble of NNs to use multi-objective evolution. It also uses gradient-based local search to optimise network complexity and error. DIVACE (diverse and accurate ensembles) [67] uses multiobjective evolution to maximise accuracy and diversity. Evolutionary selection is based on non-dominated sorting [261], a cluster-and-select approach is used to form the ensemble, and search is provided by simulated annealing and a variant of differential evolution [265]. DIVACE-II [68] is a heterogeneous multiobjective Michigan approach using NNs, Support Vector Machines and Radial Basis Function Nets. The role of crossover and mutation is played by bagging [32] and boosting [96] which produce accurate and diverse candidates. Each generation bagging and boosting makes 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 [316]. Weight evolution is 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 3-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. [92] most studies of neural robots in real environments use some form of evolution. Floreano et al. go on to claim 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 co-evolve both the neural system and the morphological properties of agents ... adds an additional valuable per-

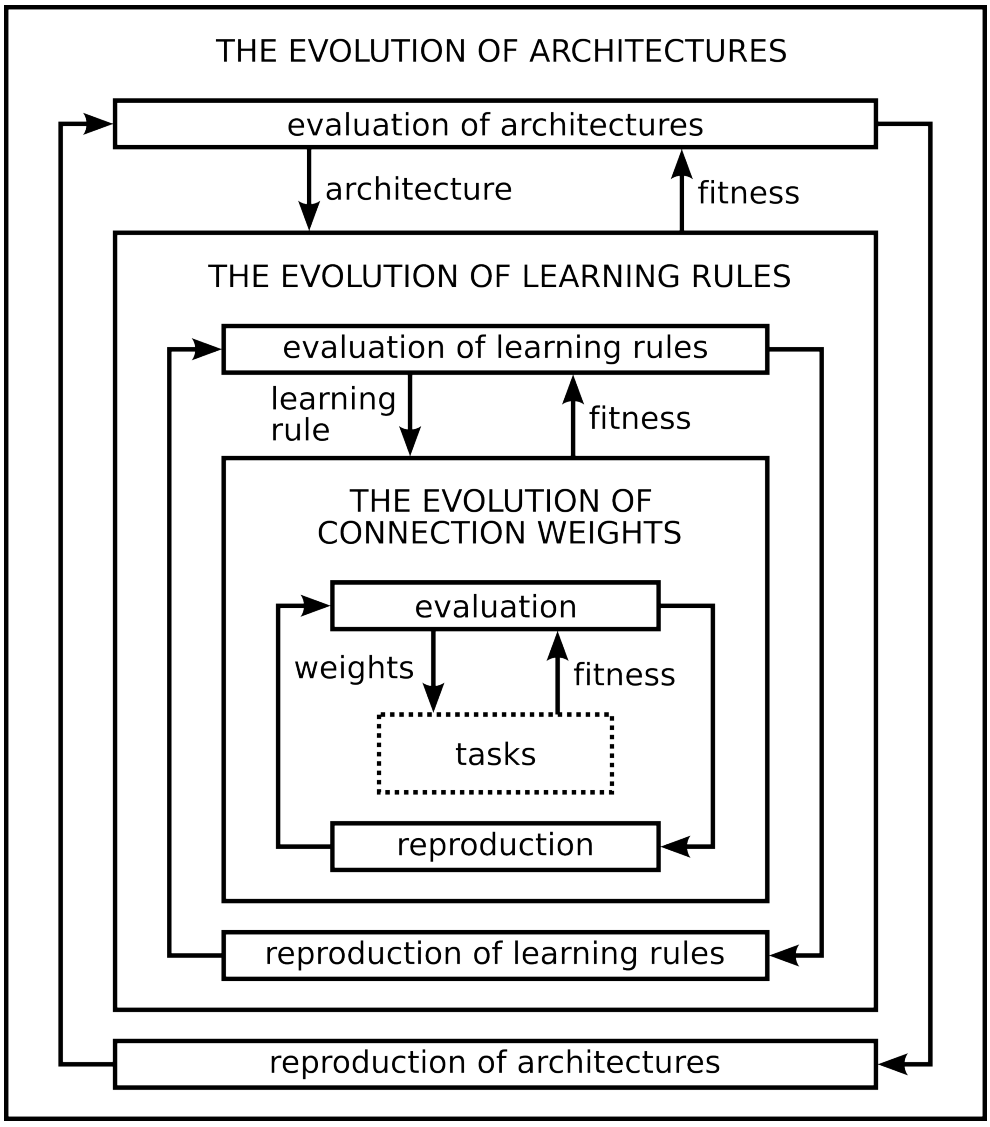


Figure 7: Yao's framework for evolving architectures, training rules and weights

spective to the evolutionary approach that cannot be matched by any other approach." [92] (p. 59).

Reading Key reading on evolving NNs includes Yao's classic 1999 survey [316], Kasabov's 2007 book, Floreano, Dürr and Mattiussi's 2008 survey [92] which includes reviews of evolving dynamic and neuromodulatory NNs, and Yao and Islam's 2008 survey of evolving NN ensembles [317].

3.5 Learning Classifier Systems

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 [301, 49] and its derivatives XCSF [305, 306] for regression/function approximation and UCS [22, 222] for supervised learning.

The Game of the Name Terminology has been contentious in this area [120]. 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 [254, 125]. In practise, 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. [110]) but it now generally includes Pitt systems as well, as implied by the name and content of IWLCS – the International Workshop on Learning Classifier Systems – 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 §2.2 and figure 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 [3] 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 [178] 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 Representation

Representing Conditions and Actions The most common representation in LCS uses fixed-length strings with binary inputs and outputs and ternary condition. In a simple Michigan version (see e.g. [301]) 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 [235] for binary GAs.) Such rules individually have limited expressive power [248] (but see also [27]) 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 [154]). Various extensions to the simple scheme described above have been studied (see [154] §2.2.2).

Real-valued Intervals Following [12] (p. 87) we distinguish two approaches to real-valued interval representation in conditions. The first is representations based on discretisation: HIDER* uses *natural coding* [106], ECL clusters attribute values and evolves constraints on them [81] while GAssist uses *adaptive discretisation intervals* [12]. The second approach is to handle real values directly. In HIDER (unlike HIDER*) genes specify a lower and upper bound (where lower is always less than upper) [3]. In [72] 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 [298, 264]. Finally [303] 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 [126]. However, the the space of *combinations* of rules is much larger than the set of rules and the evolutionary dynamics of default/exception rule combinations has proved difficult to manage in Michigan systems. Nonetheless, default rules can significantly reduce the number of rules needed for a solution [282] 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 generalisation provided by the # symbol. Finally, on the right a default rule (### $\rightarrow$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

Truth table			
A	**B**	**C**	**Output**
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Ternary Rules
0 0 # →0
0 1 # →1
1 # 0 →0
1 # 1 →1

Default Rule
0 0 # →0
1 # 0 →0
→1

Figure 8: Three representations for the 3 multiplexer function

represent the function using one less rule than the middle representation. One difficulty in evolving such 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 [237]). 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 recognises their cooperation.

The Pittsburgh GABIL [144] and GAssist [12] 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 [12].

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 [258]. It is difficult for evolution to produce these structure 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 [299, 258]. There are also problems with the use of specificity to prioritise 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 [285]) and indeed not all Michigan LCS support them (e.g. ZCS [300], 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 have been used, particularly in recent years. These include VL_1 logic [206] as used in GIL [140], first-order logic [203, 204, 205], decision lists as used in GABIL [144] and GAssist [12], messy encoding [172], ellipses [50] and hyperellipses [56], hyperspheres [200], convex hulls [187], tile coding [177] and a closely related hyperplane coding [29, 28], GP trees [4, 173, 174], Boolean networks defined by GP [37], support vectors [198], edges of an Augmented Transition Network [170], Gene Expression Programming [309], fuzzy rules (see §3.6) and neural networks [257, 74, 259, 40, 211, 75, 130, 129]. GALE [194, 193, 197] has used particularly complex representations, including the use of GP to evolve trees defining axis-parallel and oblique hyper-rectangles [197], and evolved prototypes which are used with a k-nearest-neighbour classifier. The prototypes need not be fully specified; some attributes can be left undefined. This representation has also been used in GAssist [12]. There has been limited work with alternative action representations including computed actions [278, 186] and continuous actions [308].

Evolutionary Selection of Representations As we have seen there are many representations to chose 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 [12, 13] 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 [193] evolution selects both classification algorithms and representations. GALE has elements Cellular Automata and Artificial Life: individuals are distributed on a 2-dimensional grid. Only neighbours within r cells interact: two neighbours may perform crossover, an individual may be cloned and copied to a neighbouring cell, and an individual may die if its neighbours 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-neighbour 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 [197] GALE was modified to interbreed orthogonal and oblique trees.

In the representational ecology approach [200] 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 [154]. 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

Rule	Cond.	Action	Strength
m	# # 0 0 1 1	1	200.0
m′	# # 0 0 1 1	1	220.0
n	# # 0 0 1 1	0	100.0
o	0 0 1 1 1 0	1	100.0

Rule	Cond.	Action	Strength	Numerosity
m	# # 0 0 1 1	1	200.0	2
n	# # 0 0 1 1	0	100.0	1
o	0 0 1 1 1 0	1	100.0	1

Figure 9: A population of microclassifiers (top) and the equivalent macroclassifiers (bottom)

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 Discretisation Methods and Cut Points GAssist's Adaptive Discretization Intervals (ADI) approach has two parts [12]. The first consists of adapting interval sizes. To begin, a discretisation algorithm proposes cut points for each attribute and this defines the finest discretisation 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 discretisation algorithms. Evolution is allowed to select discretisation algorithms for each attribute or rule from a pool including uniform-width, uniform-frequency, ID3, Fayyad and Irani, Màntaras, USD, ChiMerge and random. Unfortunately, evolving the best discretisers was found to be difficult and the use of ADI resulted in only small improvements in accuracy. However, further work was suggested.

Optimisation of Population Representation: Macroclassifiers In [301] Wilson introduced an optimisation 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 [151]. 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.

3.5.3 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 §3.5.3).

Windowing in Pittsburgh LCS As noted in §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 [98] learn on subsets of the data to improve runtime. Windowing has been used in Pitt LCS since at least ADAM [112]. More recently GAssist used ILAS (Incremental Learning by Alternating Strata) [12] 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. [14, 10, 9]).

Many ensemble methods improve classification accuracy by sampling data in similar ways to windowing techniques which suggests the potential for both improved accuracy and run-time 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 §2.4 as they minimise disruption to the rule population during on-line learning. An unusual feature of Michigan LCS is the emphasis placed on minimising population size, for which various techniques are use: niche GAs, the addition of a generalisation 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 figure 3 on action sets.) The inputs 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 [301]. First, a strong bias towards general rules, since they match more inputs and hence appear in more action sets. Second, pressure against overlapping rules, since they compete for reproduction [154]. Third, complete coverage of the input space, since competition occurs for each input. The niche GA was introduced in [26] and originally operated in the match set but was later further restricted to the action set [302]. It is used in in XCS and UCS and is related to *universal suffrage* [105].

EDAs instead of GAs Recently Butz et al. [58, 57, 59] replaced XCS's usual crossover with an an Estimation of Distribution Algorithm (EDA)-based method to improve solving of difficult hierarchical problems while [195, 196] 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#→0 subsumes 000 → 0 and 001 → 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 [49]) 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 labelled "Size of minimal DNF solution" in the figure) and macroclassifiers were explained in §3.5.2. In this experiment the population was initially empty and was seeded by covering §3.5.3. "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 [301].

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 figure 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 [301, 151] 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 figure 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 [152, 307, 83].

Tuning Evolutionary Search XCS is robust to class imbalances [217] but for very high imbalances tuning the GA based on a facetwise model improved performance [217, 219]. Self-tuning evolutionary search has also been studied. The mutation rate can be adapted

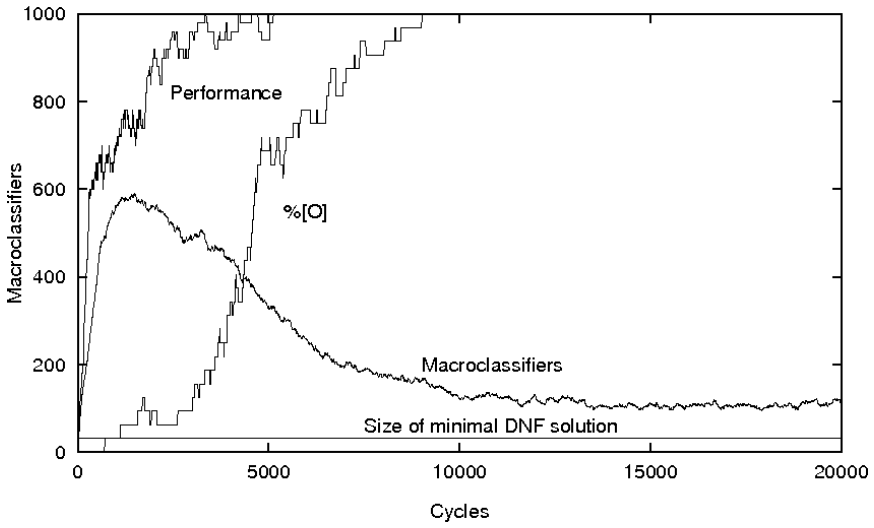


Figure 10: Evolutionary dynamics of XCS on the 11 multiplexer

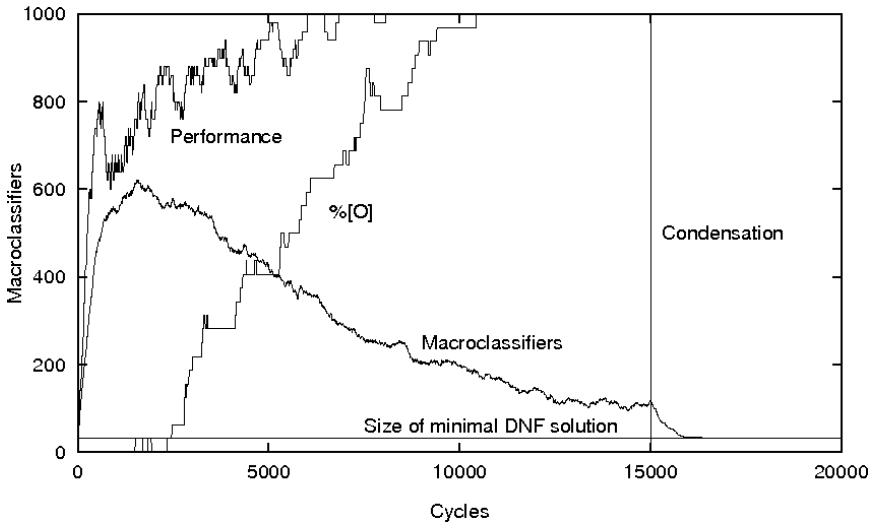


Figure 11: XCS with condensation on the 11 multiplexer

during evolution e.g. [132, 133, 130, 60], while [78] dynamically controls use of two generalisation operators: each has a control bit specifying whether it can be used and control bits evolve with the rest of the genotype.

Non-evolutionary Rule Discovery Evolution has been supplemented by heuristics in various ways. Covering, first suggested in [123], creates a rule to match an unmatched input. It can be used to an create ("seed") the initial population [287, 301, 121] or to supplement the GA throughout evolution [301]. Kovacs [154] (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 [190] selects inputs at the center of same-class clusters. For other non-evolutionary operators see [26, 236], the work on corporations of rules [310, 255, 276, 275, 277], and the work on non-evolutionary LCS.

Non-evolutionary LCS Although LCS were originally conceived as a way of applying GAs to learning problems [127], not all LCS include a GA. Various heuristics have been used to create and refine rules in e.g. YACS [102] and MACS [101]. A number of systems have been inspired by psychological models of learning. ACS [263, 46] and ACS2 [45] are examples, although ACS was also later supplemented by a GA [47, 48]. Another is AgentP, a specialised LCS for maze tasks [321, 320].

3.5.4 LCS Credit Assignment

While credit assignment in Pittsburgh LCS is a straightforward matter of multi-objective fitness evaluation, as noted in §3.5.3 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 work, 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 [300]). Such LCS are referred to as strength-based and they suffer from many difficulties with credit assignment [154], 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 [154] for analysis of the two approaches.

Prediction Updates To update rule predictions while training, the basic XCS system [301, 49] uses the Widrow-Hoff update for non-sequential problems and the Q-learning update for sequential ones. Various alternatives have been used: average rewards [271, 185], gradient descent [54, 184] and eligibility traces [88]. The basic XCSF uses NLMS (linear piecewise) prediction [305, 306] but Lanzi [178] 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 [39].

Evolutionary Selection of Prediction Functions In [179] 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 §3.5.2. 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, [175] demonstrates that XCS without generalisation implements tabular Q-learning, [53] investigates the computational complexity of XCS in a Probably Approximately Correct (PAC) setting, and [291, 290, 289, 292] analyse credit assignment and relate LCS to mainstream reinforcement learning methods. [154] 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 characterise problems which are hard for LCS [109, 156, 153, 154, 23, 15] while others model evolutionary dynamics [44, 52, 51, 55, 220, 221] and others attempt to reconstruct LCS from first principles using probabilistic models [90, 89, 91].

Hierarchies and Ensembles of LCS Hierarchical LCS have been studied for some time and [17] reviews early work. [87] and [85, 86, 84] apply hierarchical LCS to robot control while [18] uses hierarchical XCSs to learn long sequences of actions. The ensembles field §3.3 studies how to combine predictions [167] and all the above could be reformulated as ensembles of LCS. There has been some recent work on ensembles of LCS [76, 38] and also treating a single LCS as an ensemble [35, 89, 91].

3.5.5 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 multi-objective optimisation.

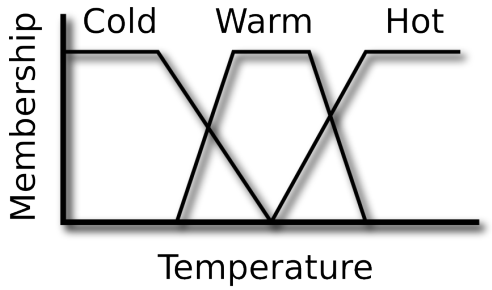
Reading No general up-to-date introduction to LCS exists. For the basics see [108] and the introductory parts of [154] or [51]. For a good introduction to representations and operators see chapter 6 of [94]. For a review of early LCS see [18]. For reviews of LCS research see [310, 180, 176]. For a review of state-of-the-art GBML and empirical comparison to non-evolutionary pattern recognition methods see [224]. For other comparisons with non-evolutionary methods see [25, 113, 244, 304, 21, 22]. Finally, the LCS bibliography [155] 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) [24] or referred to as e.g. "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 [283, 284, 103, 24, 218, 64, 223]. In addition to FRBS we briefly cover genetic fuzzy NNs but we do not cover genetic fuzzy clustering (see [73]).

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, 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 discretisation. 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, 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 figure 12 (adapted from [122]), 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

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, adjust parameterised 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

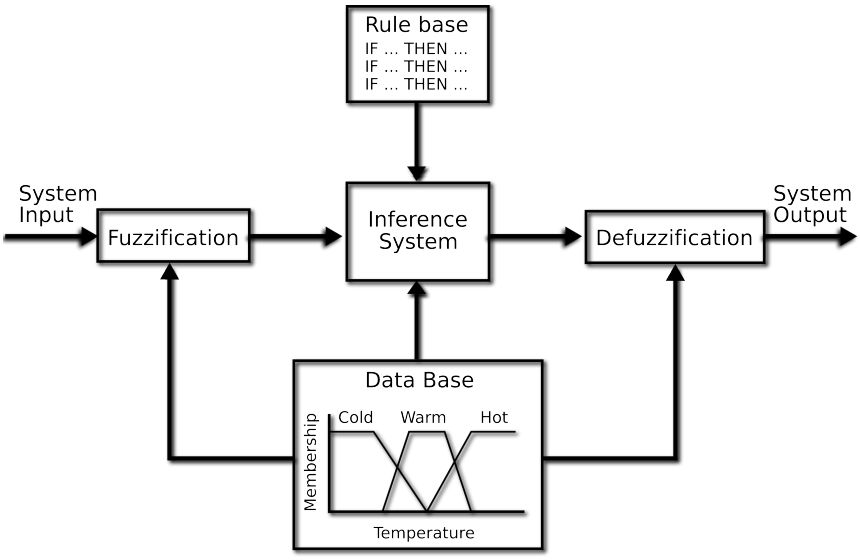


Figure 12: Components and information flow in a fuzzy rule-based system

remove irrelevant, redundant, incorrect or conflicting rules. This is a similar role to condensation in LCS (see §3.5.3). 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, [209] learns the DB and RB simultaneously while [128] simultaneously learns KB components and inference engine parameters.

Recently [242] 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 ([242] 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 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 [122].

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, [188] uses a GA to minimise the error of the NN, [115] uses both a GA and backpropagation to minimise error,

[229] optimises a fuzzy expert system using a GA and NN, and [209] uses a NN to approximate the fitness function for a GA which adapts membership functions and control rules. See [73] for an introduction to NFS, [189] for a review of EAs, NNs and fuzzy logic from the perspective of intelligent control, and [119] for a discussion of combining the three. [150] introduces Fuzzy All-permutations Rule-Bases (FARBs) which are mathematically equivalent to NNs.

3.6.3 Conclusions

Herrera [122] p. 38 lists the following active areas within GFS:

1. Multiobjective 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 [146], the Michigan approach [283], the Pittsburgh approach [274] and relational matrix-based FRBS [230]. Subsequent work includes Geyer-Schulz's 1997 book on Michigan fuzzy LCS learning RBs with GP [103], Bonarini's 2000 introductory chapter from an LCS perspective [24], Mitra and Hayashi's 2000 survey of neuro-fuzzy rule generation methods [208], Cordon et al.'s 2001 book on Genetic Fuzzy Systems in general [73], Angelov's 2002 book on evolving FRBS [7], chapter 10 of Freitas' 2002 book on evolutionary data mining [94], Herrera's 2008 survey article on GFS [122] (which lists further key reading), and finally Kolman and Margaliot's 2009 book on the neuro-fuzzy FARB approach [150].

4 Conclusions

The reader should need no convincing that GBML is a very diverse and active area. Although much integration with mainstream machine learning and has taken place in the last ten years more is needed. The use of multi-objective EAs in GBML is spreading. Integration with ensembles is natural given the population-based nature of EAs but 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), equally it is sometimes critical (e.g. in stream mining). Various methods to speed up GBML exist (see e.g. [94] §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 ten 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 [94] §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 specialised learning problems (not to mention specific applications) remain little- or un-explored with GBML, including ranking, semi-supervised learning, transductive learning, inductive transfer, learning to learn, stream mining and no doubt others which have not yet been formulated.

Acknowledgements

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

[1] H.A. Abbass. Speeding up backpropagation using multiobjective evolutionary algorithms. *Neural Computation*, 15(11):2705–2726, 2003. 18, 22

[2] D.H. Ackley and M.L. Littman. Interactions between learning and evolution. In C. Langton, C. Taylor, S. Rasmussen, and J. Farmer, editors, *Artificial Life II: Santa Fe Institute Studies in the Sciences of Complexity*, volume 10, pages 487–509. Addison Wesley, 1992. 11

[3] J. Aguilar-Ruiz, J. Riquelme, and M. Toro. Evolutionary learning of hierarchical decision rules. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 33(2):324–331, 2003. 24, 25

[4] Manu Ahluwalia and Larry Bull. A Genetic Programming-based Classifier System. In Banzhaf et al. [16], pages 11–18. 27

[5] H.C. Andersen and A.C. Tsoi. A constructive algorithm for the training of a multilayer perceptron based on the genetic algorithm. *Complex Systems*, 7(4):249–268, 1993. 19

[6] P.J. Angeline, G.M. Sauders, and J.B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Trans. Neural Networks*, 5:54–65, 1994. 20

[7] Plamen Angelov. *Evolving Rule-based Models. A tool for design of flexible adaptive systems*, volume 92 of *Studies in fuzziness and soft computing*. Springer-Verlag, 2002. 37

[8] A. Asuncion and D.J. Newman. UCI machine learning repository http://www.ics.uci.edu/∼mlearn/MLRepository.html, 2009. 18

[9] J. Bacardit, E.K. Burke, and N. Krasnogor. Improving the scalability of rule-based evolutionary learning. *Memetic Computing*, 1(1):55–67, 2009. 9, 29

[10] J. Bacardit, M. Stout, J.D. Hirst, and N. Krasnogor. Data mining in proteomics with learning classifier systems. In L. Bull, E. Bernadó Mansilla, and J. Holmes, editors, *Learning Classifier Systems in Data Mining*, pages 17–46. Springer, 2008. 29

[11] J. Bacardit, M. Stout, J.D. Hirst, A. Valencia, R.E. Smith, and N. Krasnogor. Automated alphabet reduction for protein datasets. *BMC Bioinformatics*, 10(6), 2009. 13

[12] Jaume Bacardit. *Pittsburgh Genetic-Based Machine Learning in the Data Mining era: Representations, generalization, and run-time*. PhD thesis, Universitat Ramon Llull, 2004. 25, 26, 27, 28, 29

[13] Jaume Bacardit, David E. Goldberg, and Martin V. Butz. Improving the performance of a pittsburgh learning classifier system using a default rule. In Tim Kovacs, Xavier LLòra, Keiki Takadama, Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors, *Learning Classifier Systems. International Workshops, IWLCS 2003-2005, Revised Selected Papers*, volume 4399 of *LNCS*, pages 291–307. Springer, 2007. 27

[14] Jaume Bacardit and Natalio Krasnogor. Empirical evaluation of ensemble techniques for a pittsburgh learning classifier system. In Jaume Bacardit, Ester Bernadó-Mansilla, Martin Butz, Tim Kovacs, Xavier Llorà, and Keiki Takadama, editors, *Learning Classifier Systems. 10th and 11th International Workshops (2006-2007)*, volume 4998/2008 of *Lecture Notes in Computer Science*, pages 255–268. Springer, 2008. 29

[15] A.J. Bagnall and Z.V. Zatuchna. On the classification of maze problems. In L. Bull and T. Kovacs, editors, *Applications of Learning Classifier Systems*, pages 307–316. Springer, 2005. 33

[16] W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors. *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*. Morgan Kaufmann, 1999. 39, 51

[17] Alwyn Barry. Hierarchy Formulation Within Classifiers System – A Review. In E. G. Goodman, V. L. Uskov, and W. F. Punch, editors, *Proceedings of the First International Conference on Evolutionary Algorithms and their Application EVCA'96*, pages 195–211, Moscow, 1996. The Presidium of the Russian Academy of Sciences. 33

[18] Alwyn Barry. *XCS Performance and Population Structure within Multiple-Step Environments*. PhD thesis, Queens University Belfast, 2000. 33, 34

[19] Thomas Beielstein and Shandor Markon. Threshold selection, hypothesis tests and DOE methods. In *2002 Congress on Evolutionary Computation*, pages 777–782, 2002. 12

[20] R.K. Belew, J. McInerney, and N.N. Schraudolph. Evolving networks: using the genetic algorithm with connectionistic learning. In C.G. Langton, C. Taylor, J.D. Farmer, and S. Rasmussen, editors, *Proceedings of the 2nd Conference on Artificial Life*, pages 51–548. Addison-Wesley, 1992. 20

[21] Ester Bernadó, Xavier Llorà, and Josep M. Garrell. XCS and GALE: A Comparative Study of Two Learning Classifier Systems on Data Mining. In Lanzi et al. [183], pages 115–132. 34

[22] Ester Bernadó-Mansilla and Josep M. Garrell-Guiu. Accuracy-Based Learning Classifier Systems: Models, Analysis and Applications to Classification Tasks. *Evolutionary Computation*, 11(3):209–238, 2003. 24, 34

[23] Ester Bernadó-Mansilla and T.K. Ho. Domain of competence of XCS classifier system in complexity measurement space. *IEEE Trans. Evolutionary Computation*, 9(1):82–104, 2005. 33

[24] Andrea Bonarini. An Introduction to Learning Fuzzy Classifier Systems. In Lanzi et al. [181], pages 83–104. 34, 37

[25] Pierre Bonelli and Alexandre Parodi. An Efficient Classifier System and its Experimental Comparison with two Representative learning methods on three medical domains. In Booker and Belew [30], pages 288–295. 34

[26] Lashon B. Booker. Triggered rule discovery in classifier systems. In Schaffer [246], pages 265–274. 29, 32

[27] Lashon B. Booker. Representing Attribute-Based Concepts in a Classifier System. In Gregory J. E. Rawlins, editor, *Proceedings of the First Workshop on Foundations of Genetic Algorithms (FOGA91)*, pages 115–127. Morgan Kaufmann: San Mateo, 1991. 25

[28] Lashon B. Booker. Adaptive value function approximations in classifier systems. In *GECCO '05: Proceedings of the 2005 workshops on Genetic and evolutionary computation*, pages 90–91. ACM, 2005. 27

[29] Lashon B. Booker. Approximating value functions in classifier systems. In L. Bull and T. Kovacs, editors, *Foundations of Learning Classifier Systems*, volume 183/2005 of *Studies in Fuzziness and Soft Computing*, pages 45–61. Springer, 2005. 27

[30] Lashon B. Booker and Richard K. Belew, editors. *Proceedings of the 4th International Conference on Genetic Algorithms (ICGA91)*. Morgan Kaufmann, July 1991. 41, 59

[31] M.C.J. Bot and W.B. Langdon. Application of genetic programming to induction of linear classification trees. In *Genetic Programming: Proceedings of the 3rd European Conference (EuroGP 2000)*, volume 1802 of *LNCS*, pages 247–258. Springer, 2000. 16

[32] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996. 17, 22

[33] L. Breiman. Arcing classifiers. *Annals of Statistics*, 26(3):801–845, 1998. 17

[34] Gavin Brown. Ensemble learning. In Claude Sammut and Geoffrey Webb, editors, *Encyclopedia of Machine Learning*. Springer-Verlag, 2010. 18

[35] Gavin Brown, Tim Kovacs, and James Marshall. UCSpv: Principled Voting in UCS Rule Populations. In Hod Lipson et al., editor, *GECCO'07: the Genetic and Evolutionary Computation Conference*, pages 1774–1781. ACM, 2007. 33

[36] Gavin Brown, Jeremy Wyatt, Rachel Harris, and Xin Yao. Diversity creation methods: A survey and categorisation. *Journal of Information Fusion (Special issue on Diversity in Multiple Classifier Systems)*, 6(1):5–20, 2005. 17, 18

[37] L. Bull. On dynamical genetic programming: Simple boolean networks in learning classifier systems. *International Journal of Parallel, Emergent and Distributed Systems*, 24(5):421–442, 2009. 27

[38] L. Bull, M. Studley, T. Bagnall, and I. Whittley. On the use of rule-sharing in learning classifier system ensembles. *IEEE Trans. Evolutionary Computation*, 11:496–502, 2007. 33

[39] Larry Bull. Two Simple Learning Classifier Systems. In Larry Bull and Tim Kovacs, editors, *Foundations of Learning Classifier Systems*, number 183 in Studies in Fuzziness and Soft Computing, pages 63–90. Springer-Verlag, 2005. 33

[40] Larry Bull and Toby O'Hara. Accuracy-based neuro and neuro-fuzzy classifier systems. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 905–911. Morgan Kaufmann Publishers, 9-13 July 2002. 27

[41] Edmund K. Burke, Mathew R. Hyde, Graham Kendall, Gabriela Ochoa, Ender Ozcan, and John R. Woodward. Exploring hyper-heuristic methodologies with genetic programming. In C. Mumford and L. Jain, editors, *Collaborative Computational Intelligence*. Springer, 2009. 16

[42] E.K. Burke and G. Kendall. Introduction. In E.K. Burke and G. Kendall, editors, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, pages 5–18. Springer, 2005. 8

[43] E.K. Burke, G. Kendall, J. Newall, E. Hart, P. Russ, and S. Schulenburg. Hyper-heuristics: An emerging direction in modern search technology. In F. Glover and G. Kochenberger, editors, *Handbook of Meta-heuristics*, pages 457–474. Kluwer, 2003. 8

[44] Martin Butz, Tim Kovacs, Pier Luca Lanzi, and Stewart W. Wilson. Toward a theory of generalization and learning in XCS. *IEEE Transactions on Evolutionary Computation*, 8(1):8–46, 2004. 33

[45] Martin V. Butz. An Algorithmic Description of ACS2. In Lanzi et al. [183], pages 211–229. 32

[46] Martin V. Butz. *Anticipatory learning classifier systems*. Kluwer Academic Publishers, 2002. 32

[47] Martin V. Butz, David E. Goldberg, and Wolfgang Stolzmann. Introducing a Genetic Generalization Pressure to the Anticipatory Classifier System – Part 1: Theoretical Approach. In Whitley et al. [295], pages 34–41. Also Technical Report 2000005 of the Illinois Genetic Algorithms Laboratory. 32

[48] Martin V. Butz, David E. Goldberg, and Wolfgang Stolzmann. Introducing a Genetic Generalization Pressure to the Anticipatory Classifier System – Part 2: Performance Analysis. In Whitley et al. [295], pages 42–49. Also Technical Report 2000006 of the Illinois Genetic Algorithms Laboratory. 32

[49] Martin V. Butz and Stewart W. Wilson. An Algorithmic Description of XCS. In Lanzi et al. [182], pages 253–272. 24, 30, 33

[50] M.V. Butz. Kernel-based, ellipsoidal conditions in the real-valued XCS classifier system. In H.G. Beyer et al., editor, *Proc. genetic and evolutionary computation conference (GECCO 2005)*, pages 1835–1842. ACM, 2005. 27

[51] M.V. Butz. *Rule-Based Evolutionary Online Learning Systems: A Principled Approach to LCS Analysis and Design*. Studies in Fuzziness and Soft Computing. Springer–Verlag, 2006. 33, 34

[52] M.V. Butz, D.E. Goldberg, and P.L. Lanzi. Bounding learning time in XCS. In *Genetic and evolutionary computation (GECCO 2004)*, volume 3103/2004 of *LNCS*, pages 739–750. Springer, 2004. 33

[53] M.V. Butz, D.E. Goldberg, and P.L. Lanzi. Computational complexity of the XCS classifier system. In Larry Bull and Tim Kovacs, editors, *Foundations of Learning Classifier Systems*, number 183 in Studies in Fuzziness and Soft Computing, pages 91–126. Springer-Verlag, 2005. 33

[54] M.V. Butz, D.E. Goldberg, and P.L. Lanzi. Gradient descent methods in learning classifier systems: improving XCS performance in multistep problems. *IEEE Trans. Evolutionary Computation*, 9(5):452–473, 2005. 33

[55] M.V. Butz, D.E. Goldberg, P.L. Lanzi, and K. Sastry. Problem solution sustenance in XCS: Markov chain analysis of niche support distributions and the impact on computational complexity. *Genetic Programming and Evolvable Machines*, 8(1):5–37, 2007. 33

[56] M.V. Butz, P.L. Lanzi, and S.W. Wilson. Hyper-ellipsoidal conditions in XCS: rotation, linear approximation, and solution structure. In M. Cattolico, editor, *Proc. genetic and evolutionary computation conference (GECCO 2006)*, pages 1457–1464. ACM, 2006. 27

[57] M.V. Butz and M. Pelikan. Studying XCS/BOA learning in boolean functions: structure encoding and random boolean functions. In M. Cattolico et al., editor, *Genetic and evolutionary computation conference, GECCO 2006*, pages 1449–1456. ACM, 2006. 30

[58] M.V. Butz, M. Pelikan, X. Llorà, and D.E. Goldberg. Extracted global structure makes local building block processing effective in XCS. In H.G. Beyer and U.M. O'Reilly, editors, *Genetic and evolutionary computation conference, GECCO 2005*, pages 655–662. ACM, 2005. 30

[59] M.V. Butz, M. Pelikan, X. Llorà, and D.E. Goldberg. Automated global structure extraction for effective local building block processing in XCS. *Evolutionary Computation*, 14(3):345–380, 2006. 30

[60] M.V. Butz, P. Stalph, and P.L. Lanzi. Self-adaptive mutation in XCSF. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1365–1372. ACM, 2008. 32

[61] E. Cantu-Paz and C. Kamath. Inducing oblique decision trees with evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 7(1):54–68, 2003. 5, 13

[62] Erick Cantú-Paz. Feature subset selection by estimation of distribution algorithms. In *GECCO '02: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 303–310. Morgan Kaufmann, 2002. 13

[63] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 161–168. ACM, 2006. 17

[64] J. Casillas, B. Carse, and L. Bull. Fuzzy-XCS: a michigan genetic fuzzy system. *IEEE Trans. Fuzzy Systems*, 15:536–550, 2007. 34

[65] P.A. Castilloa, J.J. Merelo, M.G. Arenas, and G. Romero. Comparing evolutionary hybrid systems for design and optimization of multilayer perceptron structure along training parameters. *Information Sciences*, 177(14):2884–2905, 2007. 21

[66] D. Chalmers. The evolution of learning: An experiment in genetic connectionism. In E. Touretsky, editor, *Proc. 1990 Connectionist Models Summer School*, pages 81–90. Morgan Kaufmann, 1990. 21

[67] Arjun Chandra and Xin Yao. Ensemble learning using multi-objective evolutionary algorithms. *Journal of Mathematical Modelling and Algorithms*, 5(4):417–445, 2006. Introduces DIVACE. 18, 22

[68] Arjun Chandra and Xin Yao. Evolving hybrid ensembles of learning machines for better generalisation. *Neurocomputing*, 69(7–9):686–700, 2006. Introduces DIVACE-II. 17, 18, 22

[69] S. Cho and K. Cha. Evolution of neural net training set through addition of virtual samples. In *Proc. 1996 IEEE Int. Conf. Evol. Comp., ICEC'96*, pages 685–688. IEEE, 1996. 13

[70] S.-B. Cho. Pattern recognition with neural networks combined by genetic algorithm. *Fuzzy Sets and Systems*, 103:339–347, 1999. See Kuncheva2004a p.167. 17

[71] Sung-Bae Cho and Chanho Park. Speciated GA for optimal ensemble classifiers in DNA microarray classification. In *Congress on Evolutionary Computation (CEC 2004)*, volume 1, pages 590–597, 2004. 17

[72] A.L. Corcoran and S. Sen. Using real-valued genetic algorithms to evolve rule sets for classification. In *Proceedings of the IEEE Conference on Evolutionary Computation*, pages 120–124. IEEE Press, 1994. 25

[73] Oscar Cordón, Francisco Herrera, Frank Hoffmann, and Luis Magdalena. *Genetic Fuzzy Systems*. World Scientific, 2001. 34, 37

[74] Henry Brown Cribbs III and Robert E. Smith. Classifier system renaissance: New analogies, new directions. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 547–552, Stanford University, CA, USA, 28–31 July 1996. MIT Press. 27

[75] Hai Huong Dam, Hussein A. Abbass, Chris Lokan, and Xin Yao. Neural-based learning classifier systems. *IEEE Trans. Knowl. Data Eng.*, 20(1):26–39, 2008. 27

[76] H.H. Dam, H.A. Abbass, and C. Lokan. DXCS: an XCS system for distributed data mining. In H.G. Beyer and U.M. O'Reilly, editors, *Genetic and evolutionary computation conference, GECCO 2005*, pages 1883–1890, 2005. 33

[77] A. Dasdan and K. Oflazer. Genetic synthesis of unsupervised learning algorithms. Technical Report BU-CEIS-9306, Department of Computer Engineering and Information Science, Bilkent University, Ankara, 1993. 21

[78] Kenneth A. De Jong, William M. Spears, and Dianna F. Gordon. Using Genetic Algorithms for Concept Learning. *Machine Learning*, 3:161–188, 13. 32

[79] T.G. Dietterich. Machine-learning research: four current directions. *AI Magazine*, 18(4):97–136, 1998. 17

[80] F. Divina, M. Keijzer, and E. Marchiori. 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)*, pages 23–30, 2002. 14

[81] F. Divina, M. Keijzer, and E. Marchiori. A method for handling numerical attributes in GA-based inductive concept learners. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2003)*, pages 898–908. Springer-Verlag, 2003. 14, 25

[82] Federico Divina and Elena Marchiori. Evolutionary concept learning. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 343–350, New York, 9-13 July 2002. Morgan Kaufmann Publishers. 14

[83] P.W. Dixon, D. Corne, and M.J. Oates. A ruleset reduction algorithm for the XCS learning classifier system. In P.L. Lanzi, W. Stolzmann, and S.W. Wilson, editors, *Learning classifier systems, 5th international workshop (IWLCS 2002)*, volume 2661 of *LNCS*, pages 20–29. Springer, 2002. 30

[84] Jean-Yves Donnart. *Cognitive Architecture and Adaptive Properties of an Motivationally Autonomous Animat*. PhD thesis, Université Pierre et Marie Curie. Paris, France, 1998. 33

[85] Jean-Yves Donnart and Jean-Arcady Meyer. Hierarchical-map Building and Self-positioning with MonaLysa. *Adaptive Behavior*, 5(1):29–74, 1996. 33

[86] Jean-Yves Donnart and Jean-Arcady Meyer. Learning Reactive and Planning Rules in a Motivationally Autonomous Animat. *IEEE Transactions on Systems, Man and Cybernetics - Part B: Cybernetics*, 26(3):381–395, 1996. 33

[87] Marco Dorigo and Marco Colombetti. *Robot Shaping: An Experiment in Behavior Engineering*. MIT Press/Bradford Books, 1998. 33

[88] J. Drugowitsch and A. Barry. XCS with eligibility traces. In H.G. Beyer and U.M. O'Reilly, editors, *Genetic and evolutionary computation conference, GECCO 2005*, pages 1851–1858. ACM, 2005. 33

[89] Jan Drugowitsch. *Design and Analysis of Learning Classifier Systems: A Probabilistic Approach*. Springer, 2008. 33

[90] Jan Drugowitsch and Alwyn Barry. A Formal Framework and Extensions for Function Approximation in Learning Classifier Systems. *Machine Learning*, 70(1):45–88, 2007. 33

[91] Narayanan E. Edakunni, Tim Kovacs, Gavin Brown, and James A.R. Marshall. Modeling UCS as a mixture of experts. In *Proceedings of the 2009 Genetic and Evolutionary Computation Conference (GECCO'09)*, pages 1187–1194. ACM, 2009. 33

[92] Dario Floreano, Peter Dürr, and Claudio Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008. 11, 19, 20, 22, 24

[93] G. Folino, C. Pizzuti, and G. Spezzano. Ensemble techniques for parallel genetic programming based classifiers. In *Proc. European Conf. on Genetic Programming (EuroGP'03)*, pages 59–69, 2003. 16

[94] A.A. Freitas. *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Spinger-Verlag, Berlin, 2002. 3, 5, 9, 13, 14, 16, 17, 34, 37, 38

[95] A.A. Freitas. A survey of evolutionary algorithms for data mining and knowledge discovery. In A. Ghosh and S. Tsutsui, editors, *Advances in Evolutionary Computation*, pages 819–845. Springer-Verlag, 2002. 10, 13

[96] Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *Proc. of the INt. Conf. on Machine Learning (ICML'96)*, pages 148–156, 1996. 17, 22

[97] Y. Freund and R. Schapire. A short introduction to boosting. *Journal of the Japanese Society for Artificial Intelligence*, 14(5):771–780, 1999. 17

[98] J. Fürnkranz. Integrative windowing. *Journal of Artificial Intelligence Research*, 8:129–164, 1998. 29

[99] Christian Gagné, Michèle Sebag, Marc Schoenauer, and Marco Tomassini. Ensemble learning for free with evolutionary algorithms? In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1782–1789. ACM, 2007. 18

[100] C. Gathercole and P. Ross. Tackling the boolean even n parity problem with genetic programming and limited-error fitness. In J.R. Koza, K. Deb, M. Dorigo, D.B. Fogel, M. Garzon, H. Iba, and R.L. Riolo, editors, *Genetic Programming 1997: Proc. Second Annual Conference*, pages 119–127. Morgan Kaufmann, 1997. 16

[101] Pierre Gérard and Olivier Sigaud. Designing efficient exploration with MACS: Modules and function approximation. In E. Cantú-Paz, J. A. Foster, K. Deb, D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. Dowsland, N. Jonoska, and J. Miller, editors, *Genetic and Evolutionary Computation – GECCO-2003*, volume 2724 of *LNCS*, pages 1882–1893. Springer-Verlag, 2003. 32

[102] Pierre Gerard, Wolfgang Stolzmann, and Olivier Sigaud. YACS, a new learning classifier system using anticipation. *Journal of Soft Computing*, 6(3–4):216–228, 2002. 32

[103] Andreas Geyer-Schulz. *Fuzzy Rule-Based Expert Systems and Genetic Machine Learning*. Physica Verlag, 1997. 34, 37

[104] Attilio Giordana and Filippo Neri. Search-Intensive Concept Induction. *Evolutionary Computation*, 3:375–416, 1995. 10

[105] Attilio Giordana and L. Saitta. Learning disjunctive concepts by means of genetic algorithms. In *Proc. Int. Conf. on Machine Learning*, pages 96–104, 1994. 29

[106] R. Giraldez, J. Aguilar-Ruiz, and J. Riquelme. Natural coding: A more efficient representation for evolutionary learning. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2003)*, pages 979–990. Springer-Verlag, 2003. 25

[107] C. Giraud-Carrier and J. Keller. Meta-learning. In J. Meij, editor, *Dealing with the data flood*. STT/Beweton, 2002. 8

[108] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Mass., 1989. 34

[109] David E. Goldberg, Jeffrey Horn, and Kalyanmoy Deb. What Makes a Problem Hard for a Classifier System? In *Collected Abstracts for the First International Workshop on Learning Classifier System (IWLCS-92)*, 1992. (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) in the section on Classifier Systems. 33

[110] David Perry Greene and Stephen F. Smith. Competition-based induction of decision models from examples. *Machine Learning*, 13:229–257, 1993. 10, 24

[111] David Perry Greene and Stephen F. Smith. Using Coverage as a Model Building Constraint in Learning Classifier Systems. *Evolutionary Computation*, 2(1):67–91, 1994. 10

[112] D.P. Greene and S.F. Smith. A genetic system for learning models of consumer choice. In *Proceedings of the Second International Conference on Genetic Algorithms and their Applications*, pages 217–223. Morgan Kaufmann, 1987. 29

[113] A. Greenyer. The use of a learning classifier system JXCS. In P. van der Putten and M. van Someren, editors, *CoIL Challenge 2000: The Insurance Company Case*. Leiden Institute of Advanced Computer Science, June 2000. Technical report 2000-09. 34

[114] F. Gruau. Automatic definition of modular neural networks. *Adaptive Behavior*, 3(2):151–183, 1995. 19

[115] D. Hanebeck and K. Schmidt. Genetic optimization of fuzzy networks. *Fuzzy sets and systems*, 79:59–68, 1996. 36

[116] L.K. Hansen and P. Salamon. Neural network ensembles. *IEEE Trans. Pattern Analysis and Machine Intelligence*, pages 993–1001, 1990. 17

[117] W.E. Hart, N. Krasnogor, and J.E. Smith (editors). Special issue on memetic algorithms. *Evolutionary Computation*, 12(3), 2004. 11

[118] William E. Hart, N. Krasnogor, and J.E. Smith, editors. *Recent Advances in Memetic Algorithms*, volume 166 of *Studies in Fuzziness and Soft Computing*. Springer, 2005. 11

[119] Lin He, Ke jun Wang, Hong zhang Jin, Guo bin Li, and X.Z. Gao. The combination and prospects of neural networks, fuzzy logic and genetic algorithms. In *IEEE Midnight-Sun Workshop on Soft Computing Methods in Industrial Applications*, pages 52–57. IEEE, 1999. 37

[120] Jörg Heitkötter and David Beasley. The Hitch-Hiker's Guide to Evolutionary Computation (FAQ for comp.ai.genetic). Accessed 28/2/09. http://www.aip.de/˜ast/EvolCompFAQ/, 2001. 24

[121] J. Hekanaho. Symbiosis in multimodal concept learning. In *Proc. 1995 Int. Conf. on Machine Learning (ML'95)*, pages 278–285, 1995. 32

[122] Francisco Herrera. Genetic fuzzy systems: taxonomy, current research trends and prospects. *Evolutionary Intelligence*, 1(1):27–46, 2008. 35, 36, 37

[123] John H. Holland. Adaptation. In R. Rosen and F. M. Snell, editors, *Progress in Theoretical Biology*. New York: Plenum, 1976. 32

[124] John H. Holland. Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In T. Mitchell, R. Michalski, and J. Carbonell, editors, *Machine learning, an artificial intelligence approach. Volume II*, chapter 20, pages 593–623. Morgan Kaufmann, 1986. 16

[125] John H. Holland, Lashon B. Booker, Marco Colombetti, Marco Dorigo, David E. Goldberg, Stephanie Forrest, Rick L. Riolo, Robert E. Smith, Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson. What is a Learning Classifier System? In Lanzi et al. [181], pages 3–32. 24

[126] John H. Holland, Keith J. Holyoak, Richard E. Nisbett, and P. R. Thagard. *Induction: Processes of Inference, Learning, and Discovery*. MIT Press, Cambridge, 1986. 25

[127] John H. Holland and J. S. Reitman. Cognitive systems based on adaptive algorithms. In D. A. Waterman and F. Hayes-Roth, editors, *Pattern-directed Inference Systems*. New York: Academic Press, 1978. Reprinted in: Evolutionary Computation. The Fossil Record. David B. Fogel (Ed.) IEEE Press, 1998. ISBN: 0-7803-3481-7. 32

[128] A. Homaifar and E. Mccormick. Simultaneous design of membership functions and rule sets for fuzzy controllers using genetic algorithms. *IEEE Trans. Fuzzy. Syst.*, 3(2):129–139, 1995. 36

[129] D. Howard and L. Bull. On the effects of node duplication and connection-orientated constructivism in neural XCSF. In M. Keijzer et al., editor, *GECCO-2008: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1977–1984. ACM, 2008. 27

[130] D. Howard, L. Bull, and P.L. Lanzi. Self-Adaptive Constructivism in Neural XCS and XCSF. In M. Keijzer et al., editor, *GECCO-2008: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1389–1396. ACM, 2008. 27, 32

[131] Y.-J. Hu. A genetic programming approach to constructive induction. In *Genetic Programming 1998: Proceedings of the 3rd Annual Conference*, pages 146–151. Morgan Kaufmann, 1998. 13

[132] J. Hurst and L. Bull. Self-adaptation in classifier system controllers. *Artificial Life and Robotics*, 5(2):109–119, 2003. 32

[133] J. Hurst and L. Bull. A self-adaptive neural learning classifier system with constructivism for mobile robot control. In X. Yao et al., editor, *Parallel problem solving from nature (PPSN VIII)*, volume 3242 of *LNCS*, pages 942–951. Springer, 2004. 32

[134] P. Husbands, I. Harvey, D. Cliff, and G. Miller. The use of genetic algorithms for the development of sensorimotor control systems. In P. Gaussier and J.-D. Nicoud, editors, *From perception to action*, pages 110–121. IEEE Press, 1994. 19

[135] H. Iba. Bagging, boosting and bloating in genetic programming. In *Proc. of the Genetic and Evolutionary Computation Conference (GECCO'99)*, pages 1053–1060, 1999. 16

[136] IEEE. *Proceedings of the 2000 Congress on Evolutionary Computation (CEC00)*. IEEE Press, 2000. 53, 56

[137] H. Ishibuchi and T. Nakashima. Multi-objective pattern and feature selection by a genetic algorithm. In *Proceedings of the 2000 Genetic and Evolutionary Computation Conference (GECCO'2000)*, pages 1069–1076. Morgan Kaufmann, 2000. 13

[138] M.M. Islam, X. Yao, and K. Murase. A constructive algorithm for training cooperative neural network ensembles. *IEEE Transactions on Neural Networks*, 14:820–834, 2003. 22

[139] A. Jain and D. Zongker. Feature selection: evaluation, application and small sample performance. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19(2):153–158, 1997. 13

[140] C.Z. Janikow. *Indictive learning of decision rules in attribute-based examples: a knowledge-intensive genetic algorithm approach*. PhD thesis, University of North Carolina, 1991. 27

[141] C.Z. Janikow. A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning*, 13:189–228, 1993. 10, 13

[142] Y. Jin and B. Sendhoff. Reducing fitness evaluations using clustering techniques and neural network ensembles. In *Genetic and Evolutionary Computation Conference (GECCO–2004)*, volume 3102 of *Lecture Notes in Computer Science*, pages 688–699. Springer, 2004. 22

[143] G. John, R. Kohavi, and K. Phleger. Irrelevant features and the feature subset problem. In *Proceedings of the 11th International Conference on Machine Learning*, pages 121–129. Morgan Kaufmann, 1994. 13

[144] Kenneth A. De Jong and William M. Spears. Learning Concept Classification Rules using Genetic Algorithms. In *Proceedings of the Twelfth International Conference on Artificial Intelligence IJCAI-91*, volume 2, pages 651–656. Morgan Kaufmann, 1991. 26, 27

[145] J.D. Kelly Jr. and L. Davis. Hybridizing the genetic algorithm and the k nearest neighbors classification algorithm. In Lashon B. Booker and Richard K. Belew, editors, *Proceedings of the 4th International Conference on Genetic Algorithms (ICGA91)*, pages 377–383. Morgan Kaufmann, July 1991. 13

[146] C. Karr. Genetic algorithms for fuzzy controllers. *AI Expert*, 6(2):26–33, 1991. 37

[147] N. Kasabov. *Evolving Connectionist Systems: The Knowledge Engineering Approach*. Springer, 2007. 19

[148] M. Keijzer and V. Babovic. Genetic programming, ensemble methods, and the bias/variance/tradeoff – introductory investigation. In *Proc. of the European Conf. on Genetic Programming (EuroGP'00)*, pages 76–90, 2000. 16

[149] H. Kitano. Designing neural networks by genetic algorithms using graph generation system. *Journal of Complex System*, 4:461–476, 1990. 19

[150] Eyal Kolman and Michael Margaliot. *Knowledge-Based Neurocomputing: A Fuzzy Logic Approach*, volume 234 of *Studies in Fuzziness and Soft Computing*. Springer, 2009. 37

[151] Tim Kovacs. Evolving Optimal Populations with XCS Classifier Systems. Master's thesis, University of Birmingham, Birmingham, UK, 1996. 28, 30

[152] Tim Kovacs. XCS Classifier System Reliably Evolves Accurate, Complete, and Minimal Representations for Boolean Functions. In Roy, Chawdhry, and Pant, editors, *Soft Computing in Engineering Design and Manufacturing*, pages 59–68. Springer-Verlag, London, 1997. ftp://ftp.cs.bham.ac.uk/pub/authors/T.Kovacs/index.html. 30

[153] Tim Kovacs. Strength or Accuracy? Fitness calculation in learning classifier systems. In Lanzi et al. [181], pages 143–160. 33

[154] Tim Kovacs. *Strength or Accuracy: Credit Assignment in Learning Classifier Systems*. Springer, 2004. 10, 25, 27, 29, 32, 33, 34

[155] Tim Kovacs. A Learning Classifier Systems Bibliography. Department of Computer Science, University of Bristol, 2009. http://www.cs.bris.ac.uk/~kovacs/lcs/search.html. 34

[156] Tim Kovacs and Manfred Kerber. What makes a problem hard for XCS? In Lanzi et al. [182], pages 80–99. 33

[157] Tim Kovacs and Manfred Kerber. High classification accuracy does not imply effective genetic search. In K. Deb et al., editor, *Proceedings of the 2004 Genetic and Evolutionary Computation Conference (GECCO)*, volume 3102 of *LNCS*, pages 785–796. Springer, 2004. 11

[158] J.R. Koza. *Genetic Programming: on the programming of computers by means of natural selection*. MIT Press, 1992. 14

[159] J.R. Koza. *Genetic Programming II*. MIT Press, 1994. 17

[160] N. Krasnogor. *Studies on the Theory and Design Space of Memetic Algorithms*. PhD thesis, University of the West of England, 2002. 16

[161] N. Krasnogor and J.E. Smith. A tutorial for competent memetic algorithms: model, taxonomy and design issues. *IEEE Transactions on Evolutionary Computation*, 9(5):474–488, 2005. 11

[162] Natalio Krasnogor. Self-generating metaheuristics in bioinformatics: the protein structure comparison case. *Genetic Programming and Evolvable Machines*, 5(2):181–201, 2004. 8, 16

[163] Natalio Krasnogor and S. Gustafson. A study on the use of self-generation in memetic algorithms. *Natural Computing*, 3(1):53–76, 2004. 8, 16

[164] K. Krawiec. Genetic programming-based construction of features for machine learning and knowledge discovery tasks. *Genetic Programming and Evolvable Machines*, 3(4):329–343, 2002. 13

[165] A. Krogh and J. Vedelsby. Neural network ensembles, cross validation and active learning. *Neural Information Processing Systems*, 7:231–238, 1995. 17

[166] M. Kudo and J. Skalansky. Comparison of algorithms that select features for pattern classifiers. *Pattern Recognition*, 33:25–41, 2000. 13

[167] Ludmila I. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms*. Wiley, 2004. 17, 18, 33

[168] I. Kushchu. An evaluation of evolutionary generalization in genetic programming. *Artificial Intelligence Review*, 18(1):3–14, 2002. 16

[169] L. Lam and C.Y. Suen. Optimal combination of pattern classifiers. *Pattern Recognition Letters*, 16:945–954, 1995. See Kuncheva2004a p.167. 17

[170] Samuel Landau, Olivier Sigaud, and Marc Schoenauer. ATNoSFERES revisited. In *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-2005*, pages 1867–1874. ACM, 2005. 27

[171] William Langdon, Steven Gustafson, and John Koza. The genetic programming bibliography http://www.cs.bham.ac.uk/ wbl/biblio/, 2009. 17

[172] Pier Luca Lanzi. Extending the Representation of Classifier Conditions Part I: From Binary to Messy Coding. In Banzhaf et al. [16], pages 337–344. 27

[173] Pier Luca Lanzi. Extending the Representation of Classifier Conditions Part II: From Messy Coding to S-Expressions. In Banzhaf et al. [16], pages 345–352. 27

[174] Pier Luca Lanzi. Mining interesting knowledge from data with the XCS classifier system. In Lee Spector, Erik D. Goodman, Annie Wu, W.B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 958–965, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann. 27

[175] Pier Luca Lanzi. Learning classifier systems from a reinforcement learning perspective. *Journal of Soft Computing*, 6(3–4):162–170, 2002. 33

[176] Pier Luca Lanzi. Learning classifier systems: then and now. *Evolutionary Intelligence*, 1(1):63–82, 2008. 34

[177] Pier Luca Lanzi, Daniele Loiacono, Stewart W. Wilson, and David E. Goldberg. Classifier prediction based on tile coding. In *Genetic and Evolutionary Computation – GECCO-2006*, pages 1497–1504. ACM, 2006. 27

[178] Pier Luca Lanzi, Daniele Loiacono, Stewart W. Wilson, and David E. Goldberg. Prediction Update Algorithms for XCSF: RLS, Kalman Filter and Gain Adaptation. In *Genetic and Evolutionary Computation – GECCO-2006*, pages 1505–1512. ACM, 2006. 24, 33

[179] Pier Luca Lanzi, Daniele Loiacono, and Matteo Zanini. Evolving classifiers ensembles with heterogeneous predictors. In Jaume Bacardit, Ester Bernadó-Mansilla, Martin Butz, Tim Kovacs, Xavier Llorà, and Keiki Takadama, editors, *Learning Classifier Systems. 10th and 11th International Workshops (2006-2007)*, volume 4998/2008 of *Lecture Notes in Computer Science*, pages 218–234. Springer, 2008. 33

[180] Pier Luca Lanzi and Rick L. Riolo. A Roadmap to the Last Decade of Learning Classifier System Research (from 1989 to 1999). In Lanzi et al. [181], pages 33–62. 34

[181] Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors. *Learning Classifier Systems. From Foundations to Applications*, volume 1813 of *LNAI*. Springer-Verlag, Berlin, 2000. 41, 48, 50, 52, 57

[182] Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors. *Advances in Learning Classifier Systems*, volume 1996 of *LNAI*. Springer-Verlag, Berlin, 2001. 43, 50

[183] Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors. *Advances in Learning Classifier Systems*, volume 2321 of *LNAI*. Springer-Verlag, Berlin, 2002. 40, 42, 61

[184] P.L. Lanzi, M.V. Butz, and D.E. Goldberg. Empirical analysis of generalization and learning in XCS with gradient descent. In H. Lipson, editor, *Genetic and Evolutionary Computation Conference, GECCO 2007, Proceedings*, volume 2, pages 1814–1821. ACM, 2007. 33

[185] P.L. Lanzi and D. Loiacono. Standard and averaging reinforcement learning in XCS. In M. Cattolico, editor, *GECCO 2006: Proceedings of the 8th annual conference on genetic and evolutionary computation*, pages 1480–1496. ACM, 2006. 33

[186] P.L. Lanzi and D. Loiacono. Classifier systems that compute action mappings. In H. Lipson, editor, *Genetic and Evolutionary Computation Conference, GECCO 2007, Proceedings*, pages 1822–1829. ACM, 2007. 27

[187] P.L. Lanzi and S.W. Wilson. Using convex hulls to represent classifier conditions. In M. Cattolico, editor, *Proc. genetic and evolutionary computation conference (GECCO 2006)*, pages 1481–1488. ACM, 2006. 27

[188] Z. Liangjie and L. Yanda. A new global optimizing algorithm for fuzzy neural networks. *Int. J. Electronics*, 80(3):393–403, 1996. 36

[189] D.A. Linkens and H.O. Nyongesa. Learning systems in intelligent control: an appraisal of fuzzy, neural and genetic algorithm control applications. *IEE Proceedings - Control Theory and Applications*, 143(4):367–386, 1996. 37

[190] Juliet Juan Liu and James Tin-Yau Kwok. An extended genetic rule induction algorithm. In *Proceedings of the 2000 Congress on Evolutionary Computation (CEC00)* [136], pages 458–463. 10, 32

[191] Y. Liu and X. Yao. Ensemble learning via negative correlation. *Neural Networks*, 12:1399–1404, 1999. 18, 21

[192] Y. Liu, X. Yao, and T. Higuchi. Evolutionary ensembles with negative correlation learning. *IEEE Trans. on Evolutionary Computation*, 4(4):380–387, 2000. 18, 22

[193] Xavier Llorà. *Genetic Based Machine Learning using Fine-grained Parallelism for Data Mining*. PhD thesis, Enginyeria i Arquitectura La Salle. Ramon Llull University, 2002. 27

[194] Xavier Llorà and Josep M. Garrell. Knowledge-Independent Data Mining with Fine-Grained Parallel Evolutionary Algorithms. In Lee Spector, Erik D. Goodman, Annie Wu, W.B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2001)*, pages 461–468. Morgan Kaufmann Publishers, 2001. 27

[195] Xavier Llorà, K. Sastry, and D.E. Goldberg. Binary rule encoding schemes: a study using the compact classifier system. In F. Rothlauf, editor, *GECCO '05: Proceedings of the 2005 conference on genetic and evolutionary computation, workshop proceedings*, pages 88–89. ACM Press, 2005. 30

[196] Xavier Llorà, K. Sastry, and D.E. Goldberg. The compact classifier system: scalability analysis and first results. In F. Rothlauf, editor, *Proceedings of the IEEE congress on evolutionary computation, CEC 2005*, pages 596–603. IEEE, 2005. 30

[197] Xavier Llorà and Stewart W. Wilson. Mixed Decision Trees: Minimizing Knowledge Representation Bias in LCS. In Kalyanmoy Deb et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004)*, volume 3103 of *Lecture Notes in Computer Science*, pages 797–809. Springer, 2004. 27

[198] D. Loiacono, A. Marelli, and P.L. Lanzi. Support vector regression for classifier prediction. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1806–1813. ACM, 2007. 27

[199] R.E. Marmelstein and G.B. Lamont. Pattern classification using a hybrid genetic algorithm – decision tree approach. In *Genetic Programming 1998: Proceedings of the 3rd Annual Conference (GP'98)*, pages 223–231. Morgan Kaufmann, 1998. 16

[200] James A. R. Marshall and Tim Kovacs. A representational ecology for learning classifier systems. In Maarten Keijzer et al., editor, *Proceedings of the 2006 Genetic and Evolutionary Computation Conference (GECCO 2006)*, pages 1529–1536. ACM, 2006. 27

[201] M.J. Martin-Bautista and M.-A. Vila. A survey of genetic feature selection in mining issues. In *Proceedings of the Congress on Evolutionary Computation (CEC'99*, pages 1314–1321. IEEE, 1999. 13

[202] Ron Meir and Gunnar Rätsch. An introduction to boosting and leveraging. In *Advanced lectures on machine learning*, pages 118–183. Springer-Verlag, 2003. 17

[203] Drew Mellor. A first order logic classifier system. In F. Rothlauf, editor, *GECCO '05: Proceedings of the 2005 conference on genetic and evolutionary computation*, pages 1819–1826. ACM Press, 2005. 27

[204] Drew Mellor. Policy transfer with a relational learning classifier system. In *GECCO Workshops 2005*, pages 82–84. ACM Press, 2005. 27

[205] Drew Mellor. A learning classifier system approach to relational reinforcement learning. In Jaume Bacardit, Ester Bernadó-Mansilla, Martin Butz, Tim Kovacs, Xavier Llorà, and Keiki Takadama, editors, *Learning Classifier Systems. 10th and 11th International Workshops (2006-2007)*, volume 4998/2008 of *Lecture Notes in Computer Science*, pages 169–188. Springer, 2008. 27

[206] R.S. Michalski, I. Mozetic, J. Hong, and N. Lavrac. The AQ15 inductive learning system: an overview and experiments. Technical Report UIUCDCS-R-86-1260, University of Illinois, 1986. 27

[207] G.F. Miller, P.M. Todd, and S.U. Hegde. Designing neural networks using genetic algorithms. In J.D. Schaffer, editor, *Proc. 3rd Int. Conf. Genetic Algorithms and Their Applications*, pages 379–384. Morgan Kaufmann, 1989. 5, 20

[208] Sushmita Mitra and Yoichi Hayashi. Neurofuzzy rule generation: Survey in soft computing framework. *IEEE Transactions on Neural Networks*, 11(3):748–768, 2000. 37

[209] T. Morimoto, J. Suzuki, and Y. Hashimoto. Optimization of a fuzzy controller for fruit storage using neural networks and genetic algorithms. *Engineering Applications of Art. Int.*, 10(5):453–461, 1997. 36, 37

[210] S. Nolfi, O. Miglino, and D. Parisi. Phenotypic plasticity in evolving neural networks. In P. Gaussier and J.-D. Nicoud, editors, *From perception to action*, pages 146–157. IEEE Press, 1994. 19

[211] T. O'Hara and L. Bull. A memetic accuracy-based neural learning classifier system. In *Proceedings of the IEEE congress on evolutionary computation (CEC 2005)*, pages 2040–2045. IEEE, 2005. 27

[212] Y.-S. Ong, N. Krasnogor, and H. Ishibuchi (editors). Special issue on memetic algorithms. *IEEE Transactions on Systems, Man and Cybernetics - Part B*, 37(1), 2007. 11

[213] Yew-Soon Ong, Meng-Hiot Lim, Ferrante Neri, and Hisao Ishibuchi. Special issue on memetic algorithms. *Soft Computing*, 13(8-9), 2009. 11

[214] Y.S. Ong, M.H. Lim, N. Zhu, and K.W. Wong. Classification of adaptive memetic algorithms: A comparative study. *IEEE Transactions on Systems Man and Cybernetics – Part B*, 36(1):141–152, 2006. 11

[215] D. Opitz and R. Maclin. Popular ensemble methods: an empirical study. *J. Artificial Intelligence Research*, 11:169–198, 1999. 17

[216] D.W. Opitz and J.W. Shavlik. Generating Accurate and Diverse Members of a Neural-network Ensemble. *Advances in Neural Information Processing Systems*, pages 535–541, 1996. 17, 18

[217] A. Orriols-Puig and E. Bernadó-Mansilla. Bounding XCS's parameters for unbalanced datasets. In Maarten Keijzer et al., editor, *Proceedings of the 2006 Genetic and Evolutionary Computation Conference (GECCO 2006)*, pages 1561–1568. ACM, 2006. 30

[218] A. Orriols-Puig, J. Casillas, and E. Bernadò-Mansilla. Fuzzy-UCS: preliminary results. In H. Lipson, editor, *Genetic and Evolutionary Computation Conference, GECCO 2007, Proceedings*, pages 2871–2874. ACM, 2007. 34

[219] A. Orriols-Puig, D.E. Goldberg, K. Sastry, and E. Bernadó-Mansilla. Modeling XCS in class imbalances: population size and parameter settings. In H. Lipson et al., editor, *Genetic and evolutionary computation conference, GECCO 2007*, pages 1838–1845. ACM, 2007. 30

[220] A. Orriols-Puig, D.E. Goldberg, K. Sastry, and E. Bernadó-Mansilla. Modeling XCS in class imbalances: population size and parameter settings. In H. Lipson, editor, *Genetic and Evolutionary Computation Conference, GECCO 2007, Proceedings*, pages 1838–1845. ACM, 2007. 33

[221] A. Orriols-Puig, K. Sastry, P.L. Lanzi, D.E. Goldberg, and E. Bernadò-Mansilla. Modeling selection pressure in XCS for proportionate and tournament selection. In H. Lipson, editor, *Genetic and Evolutionary Computation Conference, GECCO 2007, Proceedings*, page 18461853. ACM, 2007. 33

[222] Albert Orriols-Puig and Ester Bernadó-Mansilla. Revisiting UCS: Description, Fitness Sharing, and Comparison with XCS. In Jaume Bacardit, Ester Bernadó-Mansilla, Martin Butz, Tim Kovacs, Xavier Llorà, and Keiki Takadama, editors, *Learning Classifier Systems. 10th and 11th International Workshops (2006-2007)*, volume 4998/2008 of *Lecture Notes in Computer Science*, pages 96–111. Springer, 2008. 24

[223] Albert Orriols-Puig, Jorge Casillas, and Ester Bernadó-Mansilla. Evolving fuzzy rules with ucs: Preliminary results. In Jaume Bacardit, Ester Bernadó-Mansilla, Martin Butz, Tim Kovacs, Xavier Llorà, and Keiki Takadama, editors, *Learning Classifier Systems. 10th and 11th International Workshops (2006-2007)*, volume 4998/2008 of *Lecture Notes in Computer Science*, pages 57–76. Springer, 2008. 34

[224] Albert Orriols-Puig, Jorge Casillas, and Ester Bernadó-Mansilla. Genetic-based machine learning systems are competitive for pattern recognition. *Evolutionary Intelligence*, 1(3):209–232, 2008. 6, 34

[225] S. Pal and D. Bhandari. Genetic algorithms with fuzzy fitness function for object extraction using cellular networks. *Fuzzy Sets and Systems*, 65(2–3):129–139, 1994. 19

[226] Gisele L. Pappa and Alex A. Freitas. *Automating the Design of Data Mining Algorithms. An Evolutionary Computation Approach*. Natural Computing Series. Springer, 2010. 16

[227] G. Paris, D. Robilliard, and C. Fonlupt. Applying boosting techniques to genetic programming. In *Artificial Evolution 2001*, volume 2310 of *LNCS*, pages 267–278. Springer, 2001. 16

[228] F.B. Pereira and E. Costa. Understanding the role of learning in the evolution of busy beaver: A comparison between the Baldwin Effect and Lamarckian strategy. In *Proc. of the Genetic and Evol. Computation Conf. (GECCO–2001)*, pages 884–891, 2001. 11

[229] Christiaan Perneel and Jean-Marc Themlin. Optimization of fuzzy expert systems using genetic algorithms and neural networks. *IEEE Trans. on fuzzy systems*, 3(3):301–312, 1995. 37

[230] D.T. Pham and D. Karaboga. Optimum design of fuzzy logic controllers using genetic algorithms. *J. Systems Eng*, 1:114–118, 1991. 37

[231] R. Poli, W.B. Langdon, and N.F. McPhee. *A field guide to genetic programming, freely available at http://www.gp-field-guide.org.uk*. lulu.com, 2008. 14, 17

[232] W.F. Punch, E.D. Goodman, M. Pei, L. Chia-Shun, P. Hovland, and R. Enbody. Further research on feature selection and classification using genetic algorithms. In Stephanie Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms (ICGA93)*, pages 557–564. Morgan Kaufmann, 1993. 13

[233] Amr Radi and Riccardo Poli. Discovering efficient learning rules for feedforward neural networks using genetic programming. In Ajith Abraham, Lakhmi Jain, and Janusz Kacprzyk, editors, *Recent Advances in Intelligent Paradigms and Applications*, pages 133–159. Springer Verlag, 2003. 21

[234] M.L. Raymer, W.F. Punch, E.D. Goodman, L.A. Kuhn, and A.K. Jain. Dimensionality reduction using genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 4(2):164–171, 2000. 13

[235] C.R. Reeves and J.E. Rowe. *Genetic Algorithms – Principles and Perspectives. A Guide to GA Theory*. Kluwer, 2002. 25

[236] Rick L. Riolo. Bucket Brigade Performance: I. Long Sequences of Classifiers. In John J. Grefenstette, editor, *Proceedings of the 2nd International Conference on Genetic Algorithms (ICGA87)*, pages 184–195, Cambridge, MA, July 1987. Lawrence Erlbaum Associates. 32

[237] R.L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987. 26

[238] S. Romaniuk. Towards minimal network architectures with evolutionary growth networks. In *Proc. IEEE Int. Conf. on NNs, IEEE World Congress on Computational Intelligence*, volume 3, pages 1710–1713. IEEE, 1994. 13

[239] S. E. Rouwhorst and A. P. Engelbrecht. 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)* [136], pages 633–638. 16

[240] Grzegorz Rozenberg, Thomas Bäck, and Joost Kok, editors. *Handbook of Natural Computing: Theory, Experiments, and Applications*. Springer Verlag, 2010. 11

[241] D. Ruta and B. Gabrys. Application of the evolutionary algorithms for classifier selection in multiple classifier systems with majority voting. In J. Kittler and F. Roli, editors, *Proc. 2nd International Workshop on Multiple Classifier Systems*, volume 2096 of *LNCS*, pages 399–408. Springer–Verlag, 2001. See Kuncheva2004a p.321. 18

[242] L. Sánchez and I. Couso. Advocating the use of imprecisely observed data in genetic fuzzy systems. *IEEE Transactions on Fuzzy Systems*, 15(4):551–562, 2007. 36

[243] T. Sasaki and M. Tokoro. Adaptation toward changing environments: Why darwinian in nature? In P. Husbands and I. Harvey, editors, *Proceedings of the 4th European conference on artificial life*, pages 145–153. MIT Pess, 1997. 11

[244] Shaun Saxon and Alwyn Barry. XCS and the Monk's Problems. In Lanzi et al. [181], pages 223–242. 34

[245] Cullen Schaffer. A conservation law for generalization performance. In Haym Hirsh and Willian W. Cohen, editors, *Machine Learning: Proceedings of the Eleventh International Conference*, pages 259–265, San Francisco, CA, 1994. Morgan Kaufmann. 3

[246] J. David Schaffer, editor. *Proceedings of the 3rd International Conference on Genetic Algorithms (ICGA-89)*, George Mason University, June 1989. Morgan Kaufmann. 41, 57, 61

[247] Jürgen Schmidhuber. *Evolutionary principles in self-referential learning. (On learning how to learn: The meta-meta-... hook.)*. PhD thesis, Institut f. Informatik, Tech. Univ. Munich, 1987. 16

[248] Dale Schuurmans and Jonathan Schaeffer. Representational Difficulties with Classifier Systems. In Schaffer [246], pages 328–333. 25

[249] A.J.C. Sharkey. On combining artificial neural nets. *Connection Science*, 8(3–4):299–313, 1996. 17

[250] P.K. Sharpe and R.P. Glover. Efficient ga based techniques for classification. *Applied Intelligence*, 11:277–284, 1999. 13

[251] K. Sirlantzis, M.C. Fairhurst, and M.S. Hoque. Genetic algorithms for multi-classifier system configuration: a case study in character recognition. In J. Kittler and F. Roli, editors, *Proc. 2nd International Workshop on Multiple Classifier Systems*, volume 2096 of *LNCS*, pages 99–108. Springer–Verlag, 2001. See Kuncheva2004a p.321. 18

[252] J.E. Smith. Coevolving memetic algorithms: A review and progress report. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 37(1):6–17, 2007. 11

[253] M.G. Smith and L. Bull. Genetic programming with a genetic algorithm for feature construction and selection. *Genetic Programming and Evolvable Machines*, 6(3):265–281, 2005. 13

[254] Robert E. Smith. A Report on The First International Workshop on Learning Classifier Systems (IWLCS-92). NASA Johnson Space Center, Houston, Texas, Oct. 6-9. ftp://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 (ftp://ftp.krl.caltech.edu/pub/EC/Welcome.html) in the section on Classifier Systems, 1992. 24

[255] Robert E. Smith. Memory Exploitation in Learning Classifier Systems. *Evolutionary Computation*, 2(3):199–220, 1994. 32

[256] Robert E. Smith and H. Brown Cribbs. Is a Learning Classifier System a Type of Neural Network? *Evolutionary Computation*, 2(1):19–36, 1994. 19

[257] Robert E. Smith and H. Brown Cribbs. Is a Learning Classifier System a Type of Neural Network? *Evolutionary Computation*, 2(1):19–36, 1994. 27

[258] Robert E. Smith and David E. Goldberg. Variable default hierarchy separation in a classifier system. In Gregory J. E. Rawlins, editor, *Proceedings of the First Workshop on Foundations of Genetic Algorithms*, pages 148–170, San Mateo, July 15–18 1991. Morgan Kaufmann. 26

[259] Robert E. Smith and H. B. Cribbs III. Combined biological paradigms. *Robotics and Autonomous Systems*, 22(1):65–74, 1997. 19, 27

[260] D. Song, M.I. Heywood, and A.N. Zincir-Heywood. Training genetic programming on half a million patterns: an example from anomaly detection. *IEEE Transactions on Evolutionary Computation*, 9(3):225–239, 2005. 16

[261] N. Srinivas and K. Deb. Multi-objective function optimization using non-dominated sorting genetic algorithm. *Evolutionary Computation*, 2(3):221–248, 1994. 22

[262] Peter Stagge. Averaging efficiently in the presence of noise. In *Parallel problem solving from nature*, volume 5, pages 188–197, 1998. 12

[263] Wolfgang Stolzmann. Learning classifier systems using the cognitive mechanism of anticipatory behavioral control, detailed version. In *Proceedings of the First European Workshop on Cognitive Modelling*, pages 82–89. Berlin: TU, 1996. 32

[264] Chris Stone and Larry Bull. For real! XCS with continuous-valued inputs. *Evolutionary Computation*, 11(3):298–336, 2003. 25

[265] R. Storn and K. Price. Minimizing the real functions of the icec'96 contest by differential evolution. In *Proc. of the IEEE Int. Conf. on Evolutionary Computation*, pages 842–844. IEEE, 1996. 22

[266] M. Stout, J. Bacardit, J.D. Hirst, and N. Krasnogor. Prediction of recursive convex hull class assignment for protein residues. *Bioinformatics*, 24(7):916–923, 2008. 13

[267] R.S. Sutton. Two problems with backpropagation and other steepest-descent learning procedures for networks. In *Proc. 8th Annual Conf. Cognitive Science Society*, pages 823–831. Erlbaum, 1986. 19

[268] T. Sziranyi. Robustness of cellular neural networks in image deblurring and texture segmentation. *Int. J. Circuit Theory App.*, 24(3):381–396, 1996. 19

[269] A. Tamaddoni-Nezhad and S.H. Muggleton. Searching the subsumption lattice by a genetic algorithm. In J. Cussens and A. Frisch, editors, *Proceedings of the 10th International Conference on Inductive Logic Programming*, pages 243–252. Springer-Verlag, 2000. 14

[270] Alireza Tamaddoni-Nezhad and Stephen Muggleton. A Genetic Algorithms Approach to ILP. In *Inductive Logic Programming*, volume 2583/2003 of *LNCS*, pages 285–300. Springer, 2003. 14

[271] K. Tharakannel and D. Goldberg. XCS with average reward criterion in multi-step environment. Technical report, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, 2002. 33

[272] S. Thompson. Pruning boosted classifiers with a real valued genetic algorithm. In *Research and Development in Expert Systems XV – Proceedings of ES'98*, pages 133–146. Springer, 1998. 13, 17

[273] S. Thompson. 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*, pages 18–22. AAAI Press, 1999. 13, 17

[274] P. Thrift. Fuzzy logic synthesis with genetic algorithms. In Lashon B. Booker and Richard K. Belew, editors, *Proceedings of 4th international conference on genetic algorithms (ICGA'91)*, pages 509–513. Morgan Kaufmann, 1991. 37

[275] Andy Tomlinson. *Corporate Classifier Systems*. PhD thesis, University of the West of England, 1999. 32

[276] Andy Tomlinson and Larry Bull. A Corporate Classifier System. In A. E. Eiben, T. Bäck, M. Shoenauer, and H.-P. Schwefel, editors, *Proceedings of the Fifth International Conference on Parallel Problem Solving From Nature – PPSN V*, number 1498 in LNCS, pages 550–559. Springer Verlag, 1998. 32

[277] Andy Tomlinson and Larry Bull. An accuracy-based corporate classifier system. *Journal of Soft Computing*, 6(3–4):200–215, 2002. 32

[278] T.H. Tran, C. Sanza, Y. Duthen, and T.D. Nguyen. XCSF with computed continuous action. In *Genetic and evolutionary computation conference (GECCO 2007)*, pages 1861–1869. ACM, 2007. 27

[279] K. Tumer and J. Ghosh. Analysis of decision boundaries in linearly combined neural classifiers. *Pattern Recognition*, 29(2):341–348, 1996. 17

[280] Peter Turney. How to shift bias: Lessons from the baldwin effect. *Evolutionary Computation*, 4(3):271–295, 1996. 11

[281] Giorgio Valentini and Francesco Masulli. Ensembles of learning machines. In *WIRN VIETRI 2002: Proceedings of the 13th Italian Workshop on Neural Nets-Revised Papers*, pages 3–22. Springer-Verlag, 2002. 17

[282] Manuel Valenzuela-Rendón. *Two analysis tools to describe the operation of classifier systems*. PhD thesis, University of Alabama, 1989. Also TCGA technical report 89005. 25

[283] Manuel Valenzuela-Rendón. The Fuzzy Classifier System: a Classifier System for Continuously Varying Variables. In Booker and Belew [30], pages 346–353. 34, 37

[284] Manuel Valenzuela-Rendón. Reinforcement learning in the fuzzy classifier system. *Expert Systems Applications*, 14:237–247, 1998. 34

[285] R. Vallim, D. Goldberg, X. Llorà, T. Duque, and A. Carvalho. A new approach for multi-label classification based on default hierarchies and organizational learning. In *Proceedings of the Genetic and Evolutionary Computation Conference, Worrkshop Sessions: Learning Classifier Systems*, pages 2017–2022, 2003. 26

[286] Leonardo Vanneschi and Riccardo Poli. Genetic programming: Introduction, applications, theory and open issues. In Grzegorz Rozenberg, Thomas Bäck, and Joost Kok, editors, *Handbook of Natural Computing: Theory, Experiments, and Applications*. Springer Verlag, 2010. 14, 16, 17

[287] G. Venturini. SIA: A supervised inductive algorithm with genetic search for learning attributes based concepts. In P.B. Brazdil, editor, *ECML-93 - Proc. of the European Conference on Machine Learning*, pages 280–296. Springer-Verlag, 1993. 10, 32

[288] R. Vilalta and Y. Drissi. A perspective view and survey of meta-learning. *Artificial Intelligence Review*, 18(2):77–95, 2002. 8

[289] A. Wada, K. Takadama, K. Shimohara, and O. Katai. Learning classifier systems with convergence and generalization. In L. Bull and T. Kovacs, editors, *Foundations of learning classifier systems*, pages 285–304. Springer, 2005. 33

[290] Atsushi Wada, Keiki Takadama, and Katsunori Shimohara. Counter example for Q-bucket-brigade under prediction problem. In *GECCO Workshops 2005*, pages 94–99. ACM Press, 2005. 33

[291] Atsushi Wada, Keiki Takadama, and Katsunori Shimohara. Learning classifier system equivalent with reinforcement learning with function approximation. In *GECCO Workshops 2005*, pages 92–93. ACM Press, 2005. 33

[292] Atsushi Wada, Keiki Takadama, and Katsunori Shimohara. Counter Example for Q-Bucket-Brigade Under Prediction Problem. In Tim Kovacs, Xavier LLòra, Keiki Takadama, Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors, *Learning Classifier Systems. International Workshops, IWLCS 2003-2005, Revised Selected Papers*, volume 4399 of *LNCS*, pages 128–143. Springer, 2007. 33

[293] Shimon Whiteson and Peter Stone. Evolutionary function approximation for reinforcement learning. *J. Mach. Learn. Res.*, 7:877–917, 2006. 11, 12

[294] D. Whitley, T. Starkweather, and C. Bogart. Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel Comput.*, 14(3):347–361, 1990. 19

[295] Darrell Whitley, David Goldberg, Erick Cantú-Paz, Lee Spector, Ian Parmee, and Hans-Georg Beyer, editors. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*. Morgan Kaufmann, 2000. 42

[296] Darrell Whitley, V. Scott Gordon, and Keith Mathias. Lamarckian evolution, the Baldwin effect and function optimization. In *Parallel Problem Solving from Nature (PPSN-III)*, pages 6–15. Springer-Verlag, 1994. 11

[297] Jason R. Wilcox. Organizational Learning within a Learning Classifier System. Master's thesis, University of Illinois, 1995. Also Technical Report No. 95003 IlliGAL. 9, 10

[298] S. W. Wilson. Mining oblique data with XCS. In P.L. Lanzi, W. Stolzmann, and S.W. Wilson, editors, *Advances in learning classifier systems, third international workshop, IWLCS 2000*, volume 1996 of *LNCS*, pages 158–176. Springer, 2001. 25

[299] Stewart W. Wilson. Bid competition and specificity reconsidered. *Complex Systems*, 2:705–723, 1989. 26

[300] Stewart W. Wilson. ZCS: A zeroth level classifier system. *Evolutionary Computation*, 2(1):1–18, 1994. 26, 32

[301] Stewart W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995. 11, 24, 25, 28, 29, 30, 32, 33

[302] Stewart W. Wilson. Generalization in the XCS classifier system. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 665–674. Morgan Kaufmann, 1998. 29

[303] Stewart W. Wilson. Get real! XCS with continuous-valued inputs. In L. Booker, Stephanie Forrest, M. Mitchell, and Rick L. Riolo, editors, *Festschrift in Honor of John H. Holland*, pages 111–121. Center for the Study of Complex Systems, 1999. 25

[304] Stewart W. Wilson. Mining Oblique Data with XCS. In *Proceedings of the International Workshop on Learning Classifier Systems (IWLCS-2000), in the Joint Workshops of SAB 2000 and PPSN 2000*, 2000. Extended abstract. 34

[305] Stewart W. Wilson. Function approximation with a classifier system. In Lee Spector, Erik D. Goodman, Annie Wu, W.B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 974–981, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann. 24, 33

[306] Stewart W. Wilson. Classifiers that approximate functions. *Natural Computing*, 1(2–3):211–234, 2002. 24, 33

[307] Stewart W. Wilson. Compact Rulesets from XCSI. In Lanzi et al. [183], pages 196–208. 30

[308] Stewart W. Wilson. Three architectures for continuous action. In Tim Kovacs, Xavier LLòra, Keiki Takadama, Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors, *Learning Classifier Systems. International Workshops, IWLCS 2003-2005, Revised Selected Papers*, volume 4399 of *LNCS*, pages 239–257. Springer, 2007. 27

[309] Stewart W. Wilson. Classifier conditions using gene expression programming. In Jaume Bacardit, Ester Bernadó-Mansilla, Martin Butz, Tim Kovacs, Xavier Llorà, and Keiki Takadama, editors, *Learning Classifier Systems. 10th and 11th International Workshops (2006-2007)*, volume 4998/2008 of *Lecture Notes in Computer Science*, pages 206–217. Springer, 2008. 27

[310] Stewart W. Wilson and David E. Goldberg. A Critical Review of Classifier Systems. In Schaffer [246], pages 244–255. 32, 34

[311] David H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390, 1996. 3

[312] M.L. Wong and K.S. Leung. *Data mining using grammar based genetic programming and applications*. Kluwer, 2000. 10, 17

[313] K. Woods, W. Kegelmeyer, and K. Bowyer. Combination of multiple classifiers using local accuracy estimates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:405–410, 1997. 17

[314] John R. Woodward. GA or GP? That is not the question. In *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003*, pages 1056–1063. IEEE, 2003. 14

[315] K. Yamasaki and M. Sekiguchi. Clear explanation of different adaptive behaviors between Darwinian population and Larmarckian population in changing environment. In *Proc. Fifth Int. Symp. on Artificial Life and Robotics*, pages 120–123, 2000. 11

[316] X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999. 11, 19, 20, 21, 22, 24

[317] X. Yao and M.M. Islam. Evolving artificial neural network ensembles. *IEEE Computational Intelligence Magazine*, 3(1):31–42, 2008. 18, 21, 24

[318] X. Yao and Y. Liu. A new evolutionary system for evolving artificial neural networks. *IEEE Trans. Neural Networks*, 8:694–713, 1997. 20, 21

[319] X. Yao and Y. Liu. Making use of population information in evolutionary artificial neural networks. *IEEE Transactions on Systems, Man and Cybernetics B*, 28(3):417–425, 1998. 21

[320] Zhanna V. Zatuchna. *AgentP: a learning classifier system with associative perception in maze environments*. PhD thesis, University of East Anglia, 2005. 32

[321] Z.V. Zatuchna. AgentP model: Learning Classifer System with Associative Perception. In *8th Parallel Problem Solving from Nature International Conference (PPSN VIII)*, pages 1172–1182, 2004. 32

[322] B.-T. Zhang and G. Veenker. Neural networks that teach themselves through genetic discovery of novel examples. In *Proc. 1991 IEEE Int. Joint Conf. on Neural Networks (IJCNN'91)*, volume 1, pages 690–695. IEEE, 1991. 13