

Neural-Based Learning Classifier Systems

Hai H. Dam, Hussein A. Abbass *Senior Member, IEEE*, Chris Lokan and Xin Yao *Fellow, IEEE*

Abstract—UCS is a supervised learning classifier system that was introduced in 2003 for classification in data mining tasks. The representation of a rule in UCS as a univariate classification rule is straightforward for a human to understand. However, the system may require a large number of rules to cover the input space. Artificial neural networks, on the other hand, normally provide a more compact representation. However, it is not a straightforward task to understand the network.

In this paper, we propose a novel way to incorporate neural networks into UCS. The approach offers a good compromise between compactness, expressiveness, and accuracy. By using a simple artificial neural network as the classifier's action, we obtain a more compact population size, better generalization, and the same or better accuracy, while maintaining a reasonable level of expressiveness.

We also apply negative correlation learning (NCL) during the training of the resultant neural network ensemble. NCL is shown to improve the generalization of the ensemble.

Index Terms—Representations, evolutionary computing and genetic algorithms, neural nets, rule-based processing, data mining, classification.

I. INTRODUCTION

CLASSIFICATION can loosely be defined as the process of identifying commonalities in a data set sufficient for discriminating among a finite number of groups. Classification accuracy is a key factor. Other important factors include compactness and expressiveness. Compactness refers to the length of the learned model, while expressiveness relates to the understandability of the knowledge represented by the learned model. There is usually a tradeoff among accuracy, compactness and expressiveness [1]. This paper proposes a novel integration of genetics based machine learning systems and artificial neural networks to balance the three criteria.

Genetics Based Machine Learning (GBML) systems were introduced by Holland [14] in the 1970s. In the rest of this paper we use the term *learning classifier system* (LCS) to refer to GBML. In LCSs, a set of rules (called the population of classifiers) is evolved to guide the system to achieve some tasks in an arbitrary environment. LCSs employ two biological metaphors: *evolution* and *learning*. The evolutionary component plays a key role in discovering novel and potentially useful rules, while the learning component is responsible for assigning credit to rules in the population based on their (estimated) individual contribution to achieving the task. Thereby,

Manuscript received xxxx, 2006; revised xx-xx-xxxx. Work reported in this paper was funded by the Australian Research Council Linkage grant number LP0453657 and Linkage International grant number LX0561255

H.H. Dam, H.A. Abbass and C. Lokan are with the Artificial Life and Adaptive Robotics Laboratory (ALAR), School of Information Technology and Electrical Engineering, University of New South Wales at Australian Defence Force Academy, Canberra, 2600, Australia. (e-mail: {h.dam, h.abbass, c.lokan}@adfa.edu.au)

X. Yao is with CERCA, School of Computer Science, University of Birmingham, Birmingham B15 2TT, U.K. (e-mail: x.yao@cs.bham.ac.uk)

learning guides the evolutionary component to move towards a better set of rules.

LCSs have been applied in many problems. Goldberg [12] mentioned a wide range of areas where LCSs have been investigated, including biology and medicine; business; computer science; engineering and operations research; machine learning; parallel implementations; and social sciences. Bernado [3] and Butz [8] have shown that LCSs can be successfully applied to various data-mining problems, and can achieve competitive accuracy in comparison with other machine learning algorithms such as the decision tree learner C4.5, Naïve Bayes classifier, rule extraction methods such as PART, support vector machines, etc.

Work on LCSs normally falls in one of two categories: the Pittsburgh [34] and the Michigan [16] approaches. The main difference is that an individual in the Pittsburgh approach is a set of rules representing a complete solution to the learning problem, while an individual in the Michigan approach is a single rule that represents a partial solution to the overall learning task. Our current research focuses on mining data streams, where data instances are continuously arriving in real time. Due to memory constraints when faced with an infinite stream of data, each instance is presented to the system at most once before being discarded. We consider that Michigan-style LCSs are more suitable in this domain, due to their abilities to learn incrementally on the fly and their smaller space and time complexity.

Many models of Michigan-style LCSs have been introduced in the last two decades. Recently, accuracy-based fitness models have captured the attention of researchers in the field because of their good performance. XCS [35][36] is the first system of this type. XCS works for both supervised learning and reinforcement learning problems. UCS [2] is a derivation of XCS that specializes on classification tasks. Since we are working on classification, UCS was chosen as our baseline LCS learner.

A rule in UCS is represented as stimulus-response (*i.e.* condition-action). Traditionally, the condition is encoded in ternary (0,1,#) and the action is encoded in binary. Other representations of the condition have been introduced, to accommodate more complex problems in the real world. Among them, the interval encoding introduced by Wilson [38] is a popular one. The action is normally denoted by a scalar (or more precisely a member of a finite countable set) for classification problems. The major benefit of this representation is that the rules can be easily interpreted by a human. This can be important: in many data mining problems, the ability to understand the learnt knowledge is sometimes as important as obtaining an accurate model.

The main drawback of UCS is that it normally requires a large population of specific rules (or a large model) to

cover the whole input space. This makes the system less efficient in terms of computational time, memory usage, and comprehensibility. This motivated us to come up with a new representation, aiming for a population with fewer classifiers while still maintaining the predictive accuracy.

Artificial neural networks, commonly referred to as neural networks (NN's), are inspired by the human brain. Haykin [13] presented several useful properties and capabilities of neural networks, such as nonlinearity, built-in adaptivity to changes in the surrounding environment, capability of robust computation, and a compact model. These properties motivated us to investigate the use of neural networks in UCS. Their disadvantage is in expressiveness. They are normally treated as black boxes, due to their complicated representation.

The use of neural networks in LCS was first proposed by Bull et. al. [6][7]. In their approach (called NCS) the *condition-action* parts of classifiers were replaced by a fully connected multi-layer perceptron. Experiments showed that NCS is able to learn appropriate structure in simple robotics applications [18], and can solve several maze problems [7]. However, by replacing a rule completely by a neural network, NCS lacks the main advantage of LCS; that is, being a rule-based system that is potentially easy to understand.

In order to use neural networks without losing too much expressiveness, we decided to only modify the action part of classifiers. In our proposed representation, the conditions are unchanged but each action is replaced by a simple neural network. The conditions are used to decompose a complex problem into a number of relatively simple tasks. Each task is then learnt by a complete but simple neural network.

A smaller number of more general classifiers can be the result of this representation. Those classifiers might not be accurate, and can cover an area containing some boundaries. It is then the responsibility of the neural network to capture the decision boundaries inside the local area and provide an appropriate outcome when required by the system. This in theory should result in a smaller population size than the traditional univariate rule representation by UCS.

We also employ negative correlation learning (NCL) [24] in the neural-based LCS (NLCS) for maintaining diversity in the population in order to improve the system's performance.

This paper addresses two primary research questions:

- Is the proposed neural-based representation beneficial to UCS? We hypothesize that the neural-based representation will help to compact the population while still maintaining an equivalent predictive accuracy.
- Can negative correlation learning (NCL) help NLCS to achieve better accuracy? We hypothesize that NCL is able to diversify individuals in the neural network ensemble in order to improve the overall predictive accuracy. This is the first attempt to apply NCL in LCS; hence it is important to study the effect of NCL on the system.

The paper is structured as follows. The next section will provide a short review of learning classifier systems and negative correlation learning. Section III describes our proposed representation and discusses the framework of NLCS. The experimental setup is explained in Section IV. Sections V and

VI investigate each of our research questions. The final section provides the conclusion of the paper.

II. BACKGROUND MATERIALS

A. Learning Classifier Systems

LCS is a rule-based evolutionary learning classifier system, in which each classifier implements a partial solution to the target problem. A typical goal of LCS is to evolve a population of classifiers $[P]$ to represent a complete solution to the target problem. LCS can employ either reinforcement learning (e.g XCS) or supervised learning (e.g UCS) for evaluating classifiers in the population. This paper focusses on UCS.

Each classifier consists of a rule and a set of parameters. A rule is made up of a *Condition* (the body of the rule), and an *Action* (the prediction of the classifier). The *Condition* embodies several environmental states, which the classifier may match to. The *Action* is a proposed outcome of the classifier if it is fired/activated. The main parameter is *fitness* that measures the classifier's goodness relative to the rest of the population.

During both the exploration (training) and exploitation (testing) cycles, UCS repeatedly receives input from the environment. A match set $[M]$ is formed for each input, containing all classifiers in the population $[P]$ whose *condition* match the input. Classifiers in $[M]$ will work together to decide on the system's outcome.

In the exploitation phase, a *prediction array* $[PA]$ is formed estimating the probability of each possible outcome in $[M]$ with regards to their fitness. The action with the highest probability in $[PA]$ is selected as the system's prediction.

The exploration phase is more complicated, as it involves both learning and searching. Since UCS is a supervised learner, a desired class/outcome accompanies the input. A *correct set* $[C]$ is formed, containing those classifiers in $[M]$ that have the same *action* as the input. If $[C]$ is empty, *covering* is applied, where a classifier that matches the input is created and assigned the same outcome as the input. UCS is an incremental learner, where knowledge is updated as more data becomes available. All parameters of the classifiers in $[M]$ are revised for each training instance, reflecting the system's response to new knowledge.

GA is invoked in $[C]$ if the average time since the last GA activation of classifiers in $[C]$ surpasses a user-defined threshold. Two parents are selected from $[C]$ with probability proportional to their fitness. Two offspring are generated by reproducing, crossing-over, and mutating the parents with certain probabilities. Offspring are inserted in $[P]$ if they are not subsumed by the parents. If the population size hits a predefined limit, some classifiers are removed by voting within the population.

B. Knowledge Representation

The classifier's action is normally encoded by a scalar representing one of the finite number of possible classes.

The condition can be represented in many ways. Traditionally, the condition is denoted by ternary alphabets (using binary bits and a don't care (#) symbol which means either

0 or 1) for solving binary problems. The # symbol is used to generalize the rule. The ternary representation has been extended to other representations for solving real-world data. A simple but effective one is the interval predicate representation introduced by Wilson [37][38]. He proposed the center-spread representation for continuous-valued inputs and the min-max representation for integer-valued inputs [10]. These representations are known in traditional data mining, but they were new for LCS.

An interval predicate in the center-spread representation takes the form (c_i, s_i) where $c_i, s_i \in [p_{min}, q_{max})$, p_{min} and q_{max} are the lower and upper bound of an interval. c_i is the center of the interval and s_i is the width of the interval from the center. An interval predicate in the min-max representation is represented as (l_i, u_i) where l_i and u_i are the minimum and maximum bounds of the interval.

The major issue with interval predicate representations is the compactness of the model. Butz [9] found that it is difficult for LCS, with the hyper-rectangular condition, to evolve an effective space partitioning when dealing with oblique decision boundaries. He showed that learning takes longer and the consequent population size is much higher for oblique data.

Figure 1 illustrates three representations for the condition part of a classifier: univariate rules, multivariate oblique, and our proposed representation for approximating a non-linear function. The learning task is to distinguish class '+' (inside the circle) and class '-' (outside the circle). An example of the

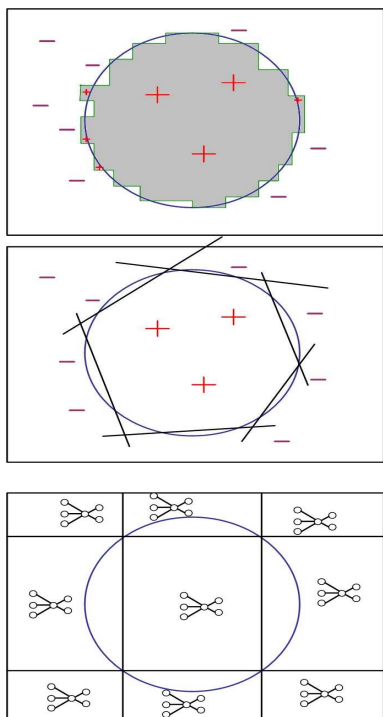


Fig. 1. Three representations for the condition part of a classifier - From top to bottom: univariate rules, multivariate oblique, and our proposed representation

first representation is univariate decision trees, which partition the feature space with axis-parallel hyperplanes. The input-output mapping, or the underlying function, is approximated by the hyper-rectangles obtained from these partitions. An

enormous number of rules (hyper-rectangles) is required for smooth approximation of a nonlinear surface such as a circle.

The multivariate oblique representation (e.g. neural networks, multivariate decision trees) has more complicated partitioning methodology and therefore is more powerful than the univariate rules. Fewer oblique lines are required to approximate a circle than with the univariate representation.

In this paper, we propose a different representation, where the search space is divided into several hyper-rectangles. Each hyper-rectangle has a neural network specializing on local data. We do not need as many hyper-rectangles as in the first representation to approximate a non-linear boundary, because the hyper-rectangles in our case can be more general. The learning workload is shared with the neural network in order to achieve high predictive accuracy.

Neural networks are popular in machine learning because they can provide a compact model and generalize well. Bull et al. [6][7] proposed the neural-based learning classifier system (NCS). In their framework, the *condition-action* of classifiers is replaced by a fully connected multi-layer perceptron (MLP). The classifier consists of an MLP and a fitness value. Each neuron at the output layer is responsible for each possible action. There is also an extra neuron at the output layer for signifying its membership of a match set $[M]$. Given an input from the environment, all neural networks in the population feed forward the input values through hidden layers by sigmoid transfer functions. If the extra neuron has the highest activation value, the classifier does not form part of the match set $[M]$. Otherwise, it will belong to the match set, proposing the action corresponding to the output neuron with the highest activation value. Action selection in NCS is performed by a simple roulette wheel selection policy based on fitness.

The neural networks in NCS are built around the neurobiological theory of neural constructivism [30]. NCS starts with a small network. An appropriate structure is then added through the learning process, particularly through growing/pruning dendritic connectivity, until some satisfactory level of utility is reached. The use of self-adaptive constructivism helps the realization of appropriate autonomous behavior. Experiments show that NCS is able to learn appropriate structure in simple robotics applications [18], and can solve several maze problems [7].

However, by replacing a rule completely by a neural network, NCS lacks the main advantage of LCSs; that is, being a rule-based system that is potentially easy to understand. In many data mining problems, the ability to understand learnt knowledge can be as important as obtaining an accurate model. For instance, a company might want to profile customers' expenditures in terms of their consumption, services, location, income, season, etc. LCS will provide a set of rules drawing the relationship between those features with regards to customers' spending. Understanding their purchase behavior might help managers to identify the best segments that influence their spending so that they can be used for future prediction as well as investment decisions.

Our proposed representation suggests a simple neural network for each action. The network should be simple enough (0 or 1 hidden unit) to be explained easily. Biologically, the con-

dition part is similar to a receptive field which represents the conditions for the associated neural network to be activated.

Overlapping classifiers normally co-exist in the population of traditional LCSs and also in our framework. A default hierarchy of classifiers [15] is a good example showing the benefit of overlapping over non-overlapping classifiers. In this example (Figure 2), the learning task is to distinguish between the ‘+’ region (inside the small rectangle) and ‘-’ region (outside the small rectangle). As many as nine non-overlapping classifiers might be required to approximate correctly the feature space. On the other hand, as few as two overlapping classifiers can represent the whole space. While finding default hierarchies remains challenging in LCSs, this example shows that overlapping classifiers might benefit the system.

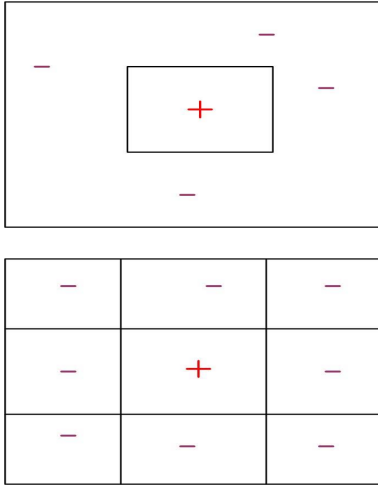


Fig. 2. A demonstration of 2 overlapping classifiers (top) and 9 non-overlapping classifiers (bottom) for the same learning task

Kovacs states that there is no clear evidence to support the advantages of non-overlapping over overlapping classifiers in LCSs [20]. But he believes that in some cases, having overlaps is more advantageous (e.g. when the training data has missing values or an imbalance in class distribution). A non-overlapping population of rules can be derived from the overlapping population by reducing the generality of some of the rules. However, having general rules would provide additional hypotheses about the unknown subset, which can be confirmed or rejected later once the data becomes available. LCSs penalize overlapping rules by using fitness sharing to implicitly bias the population towards non-overlapping rules. Hence, non-overlapping rules are encouraged to evolve while still maintaining diversity.

After receiving an instance from the environment, a match set $[M]$ is formed of those classifiers in the population $[P]$ that match with the input. Ignoring the classifiers’ conditions now, the match set consists of a set of actions or MLPs. Thus, we have an ensemble of NNs which might have a conflict. Each network in an ensemble could perform the task independently, but better generalization might be obtained from the combination.

In order to build a good NN ensemble, one has to answer two questions: (i) how to resolve the conflict and deliver a

good prediction based on the expertise of the rules in the match set, (ii) how to promote diversity to improve the accuracy of the ensemble?

We employ Negative Correlation Learning (NCL) to promote diversity.

C. Negative Correlation Learning

NCL [24][26] was introduced in NN ensembles to improve the learning performance in classification, regression and time-series problems. The idea of NCL is to introduce a correlation penalty term into the error function of each individual network, so that all networks can be trained simultaneously and interactively [25]. The success of NCL was explained by Brown [4]: training individual networks with error functions plus their contributions to overall ensemble error helps to diversify individuals in the ensemble, and therefore improves the learning performance.

We explain briefly now how negative correlation works. Please refer to [24] for more details.

Given a training data set $D = (x_1, y_1), \dots, (x_n, y_n), \dots, (x_N, y_N)$, we want to estimate the output \hat{y} and update the knowledge of the population for each instance.

Consider the n^{th} training instance. Firstly, a match set $[M]$ is formed of the M rules in the population which match x_n . This match set is implicitly a NN ensemble, because the condition part can be ignored at this stage and the outcome is decided by the action parts (a set of neural networks). We determine \hat{y}_{en} (the output of the ensemble on the n^{th} training instance) by averaging the outputs from the networks in the ensemble:

$$\hat{y}_{en}(n) = \frac{1}{M} \sum_{i=1}^M \hat{y}_i(n) \quad (1)$$

where $\hat{y}_i(n)$ is the output of the i^{th} classifier in $[M]$ on the n^{th} training instance.

Negative correlation is added to the error function E_i of each network i in $[M]$ before back-propagation is executed. If the mean-squared error is used, the error function of each individual network i is defined by

$$E_i = \frac{1}{N} \sum_{n=1}^N E_i(n) \quad (2)$$

$$= \frac{1}{N} \sum_{n=1}^N \left[\frac{1}{2} (\hat{y}_i(n) - y(n))^2 \right] \quad (3)$$

where N is the number of training instances, $E_i(n)$ is the value of the error function of network i on the n^{th} training instance, and $y(n)$ is the desired output of the n^{th} training instance. A negative correlation term is added to the error function:

$$E_i = \frac{1}{N} \sum_{n=1}^N \left[\frac{1}{2} (\hat{y}_i(n) - y(n))^2 + \lambda p_i(n) \right] \quad (4)$$

where the parameter λ is used to adjust the strength of the penalty and p_i is a penalty term. The purpose of the back-propagation is to minimize the error in the future by adjusting

the weights accordingly. Minimizing p_i is to negatively correlate each individual's error with the rest of the ensemble. The penalty function p_i is defined by

$$p_i(n) = (\hat{y}_i(n) - \hat{y}_{en}(n)) \sum_{j \neq i} (\hat{y}_j(n) - \hat{y}_{en}(n)) \quad (5)$$

The partial derivative of E_i with respect to the output of individual i on the n^{th} training instance is

$$\frac{\partial E_i(n)}{\partial \hat{y}_i(n)} = \hat{y}_i(n) - y(n) + \lambda \frac{\partial p_i(n)}{\partial \hat{y}_i(n)} \quad (6)$$

$$= \hat{y}_i(n) - y(n) + \lambda \sum_{j \neq i} (\hat{y}_j(n) - \hat{y}_{en}(n)) \quad (7)$$

$$= \hat{y}_i(n) - y(n) - \lambda(\hat{y}_i(n) - \hat{y}_{en}(n)) \quad (8)$$

under the assumption that the output of ensemble \hat{y} has constant value with respect to $y_i(n)$. λ is a parameter used to adjust the negative term in each individual. $\lambda = 0$ means that negative correlation is not enforced in the system. The higher λ is, the more strongly the errors of the rest of the ensemble are correlated in each individual's error. This parameter is important because it controls the weight of the negative correlation term when adding it to the error function.

A single weight connected to the output layer in the network is updated by the formula

$$\Delta w_i(n) = \beta [(\hat{y}_i(n) - y(n)) - \lambda(\hat{y}_i(n) - \hat{y}_{en}(n))] \frac{\partial \hat{y}_i(n)}{\partial w_i(n)} \quad (9)$$

where β is a learning rate to decide on the update step.

The value of λ was initially forced to lie inside the range $[0, 1]$. Brown [4] later showed that this constraint is not necessary. He also introduced a strength parameter γ , a function of λ and a number of networks in the ensemble (in our system, the size of the match set):

$$\gamma = \lambda \left[\frac{M}{2(M-1)} \right] \quad (10)$$

In this paper we shall refer to the γ parameter instead of λ .

III. NEURAL-BASED LEARNING CLASSIFIER SYSTEMS (NLCS)

In our proposed representation, a rule in NLCS consists of two main components as in traditional LCS: the *condition* and the *action*. The classifier condition can be encoded by any existing representation of LCS such as interval predicates [38], kernel-based ellipsoid [9], messy coding [21], S-expression [22], to name a few. In this paper, we use interval predicates for expressiveness. Our major modification is with respect to the *action* of NLCS, where the traditional class value is replaced by a complete, but simple, neural network (NN).

Figure 3 illustrates a population of neural-based classifiers for a binary classification. In this example, the population has six rules. Each rule is responsible for a small input region matching its condition. The neural network of a rule is only fed by inputs belonging to the local region.

The idea of our representation is that the whole input space is divided into relatively small areas by the classifier conditions. Each local region is then handled by one or more NNs.

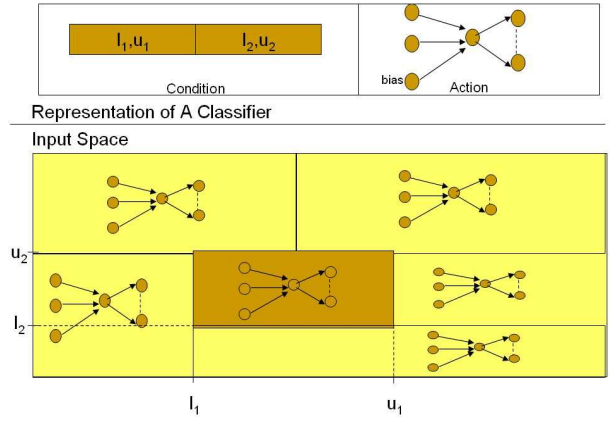


Fig. 3. An example of a classifier in NLCS. The input space is divided into six local spaces by the upper-bound and lower-bound of classifiers' intervals. A neural network of each classifier is responsible for learning its local area and deciding on the possible outcome of the classifier depending on the input values.

The type of NN used in this paper is the MLP, which has been shown to be a universal approximator [17]. This is a directed graph with several layers, including an input layer, hidden layers, and an output layer. A bias unit is connected to each hidden unit. To retain expressiveness, we use static and simple MLPs with only one hidden layer and one hidden node. The number of neurons in the input layer is the same as the number of input features. The outputs are encoded using m output neurons corresponding to m classes. The output node with the highest activation designates the class. A hidden unit employs a sigmoid function to introduce nonlinearity into the network.

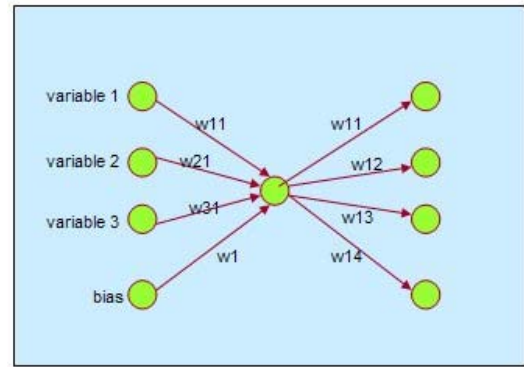


Fig. 4. A neural network for classification, with three input variables and four possible outcomes.

Figure 4 shows our simple neural network in each classifier. In this example, each input is a vector of three variables and an outcome needs to be drawn from four distinct groups (classes). The decision is made based on the network's knowledge (a set of weights) as the input is fed-forward from the input layer through the network to the output layer. Once the output layer is reached, the output neuron with the highest activation value is chosen as the prediction. Usually, each output neuron is associated with a group (or class) defined by users.

A. The NLCS Algorithm

The algorithm of NLCS is presented in Algorithm 1. NLCS inherits most parameters from UCS, except the deletion and subsumption parameters. In short, the system is evolved and guided by the discovery and update components. These are described in the next two subsections respectively.

```

Initialize NLCS parameters;
Initialize [P] to empty;
repeat
  Given the training data set  $D = (x_1, y_1), \dots, (x_N, y_N)$ 
  from the environment;
  for each training instance  $(x_i : y_i)$  do
    Form a match set [M] of those classifiers in [P]
    that match the input  $x_i$ ;
    if [M] is empty then
      Create a general classifier  $C_{cover}$ ;
      Insert  $C_{cover}$  to [P] and [M];
    end
    if the population size is less than N and at the
    end of the covering_window then
      Create a classifier that matches all
      misclassified instances during the last
      covering_window;
      Insert the classifier to [P];
    end
    Update the fitness of classifiers in [M];
    Adjust networks' weights of classifiers in [M];
    if the average time since last GA activation of
    classifiers in [M] is higher than  $\theta_{GA}$  and the
    population size is less than N then
      Select and reproduce (mutation and
      crossover) two classifiers in [M];
      Insert offspring in [P];
    end
  end
until the termination conditions are met ;

```

Algorithm 1: Algorithm of NLCS

During the training process, the error is computed by Equation 3 which compares the system prediction against the desired class. The error is then back-propagated and the network weights are altered as in Equation 9. The key goal of back-propagation is to determine a set of weights that minimizes the error in the future.

B. The Discovery Component

As in UCS, the rule discovery of NLCS is conducted by the covering and evolutionary components.

The covering component is activated if one of the following two conditions is true: (i) the match set [M] is empty; or (ii) the current population size has not exceeded the predefined maximum population size. The first condition occurs when the system starts with an empty population. The covering function will create the most general rule, which matches all instances. With the latter condition, the system will generate a rule to cover all instances misclassified by the system during a previous covering_window. The size of this window affects

the level of generality of newly-created classifiers. Section V-D investigates in detail the effect of this window size.

The evolutionary component is applied to the correct set [C] by selecting two parents, with probability proportional to fitness, for crossover and mutation.

Children are produced by crossing-over and then mutating parents' genes. The offspring are inserted into the population if their genes are distinct from the parents. This process occurs when the mean *experience* of rules in [C] is greater than a predefined threshold and the current population size is less than a predefined maximum.

C. The Update Component

Rules are updated incrementally whenever they appear in the match set [M]. Their parameters such as fitness and weights of MLPs are updated appropriately.

As in UCS, the fitness of classifiers is based on their accuracy, which is computed as the ratio of correct classifications by the classifier to the number of times the classifier has been in the match set:

$$acc = \frac{N_{correct}}{N_{matches}} \quad (11)$$

The fitness is computed as a function of accuracy:

$$F = (acc)^v \quad (12)$$

where v is a predefined constant.

The MLPs learn to perform various tasks by adaptation of their weights using the back-propagation algorithm.

The error of the output at neuron i when presented with the n^{th} training instance is defined by:

$$e_i(n) = y_i(n) - \hat{y}_i(n) \quad (13)$$

where $y_i(n)$ refers to the desired outcome and $\hat{y}_i(n)$ is the actual outcome. The instantaneous value of the error of neuron i is $\frac{1}{2}e_i(n)^2$. The instantaneous value of the total error of the network with M classes (M neurons in the output layer) for instance n is:

$$E(n) = \frac{1}{2} \sum_{i=1}^M e_i(n)^2 \quad (14)$$

The back-propagation algorithm updates the weight w_i connecting the input of neuron i by $\Delta w_i(n)$, which is proportional to the partial derivative $\frac{\partial E(n)}{\partial w_i(n)}$. It can be expressed as:

$$\frac{\partial E(n)}{\partial w_i(n)} = \frac{\partial E(n)}{\partial e_i(n)} \frac{\partial e_i(n)}{\partial \hat{y}_i(n)} \frac{\partial \hat{y}_i(n)}{\partial w_i(n)} \quad (15)$$

Differentiating both sides of Equations 13 and 14, we get

$$\frac{\partial e_i(n)}{\partial \hat{y}_i(n)} = -1 \quad (16)$$

$$\frac{\partial E(n)}{\partial e_i(n)} = e_i(n) \quad (17)$$

The Δw_i is applied to $w_i(n)$ by the delta rule as following:

$$\Delta w_i(n) = -\beta \frac{\partial E(n)}{\partial w_i(n)} \quad (18)$$

where β is the learning rate parameter. Thus, the correction Δw used for a single update of each weight in the network is

$$\Delta w_i(n) = \beta e_i(n) \frac{\partial \hat{y}_i(n)}{\partial w_i(n)} \quad (19)$$

Equation 19 indicates that the calculation of Δw depends on the error function at the output neuron i . There are two ways to calculate $e_i(n)$ depending on its layer. Equation 13 can be used to compute the error of neurons at the output layer. If a neuron is a hidden node, the error needs to be computed recursively in terms of the errors of all neurons to which that hidden node is directly connected.

IV. EXPERIMENTAL SETUP

A. Data sets

To investigate the performance of our proposed neural-based representation, we conducted experiments on thirteen data sets available from the University of California at Irvine (UCI) repository [28] and one artificial data set. All represent classification problems.

Table I shows the properties of the data sets used in this paper: *Inst* (the number of instances in the data set), *Fs* (the number of features), *R* (the number of real-valued features), *I* (the number of integer-valued features), *N* (the number of nominal-valued features), and *C* (the number of possible outcomes).

TABLE I
THE PROPERTIES OF TESTING DATA SETS

Problem	# Inst	# Fs	# R	# I	# N	# C
balance-scale	625	4	4	0	0	3
breast-w	699	9	0	9	0	2
bupa	345	6	6	0	0	2
credit-a	690	15	10	0	5	2
diabetes	768	8	8	0	0	2
glass	214	9	0	0	0	6
heart-statlog	270	13	13	0	0	2
ionosphere	351	34	34	0	0	2
iris	150	4	4	0	0	3
lymph	148	18	3	6	9	4
segment	2310	19	19	0	0	7
sonar	208	60	60	0	0	2
tao	1888	2	2	0	0	2
vowel	990	13	10	0	3	11

B. Systems Setup

UCS and NLCS are developed in C++. If not stated differently, UCS is setup with the same parameter values used by Wilson [38] and Bernado [2] as follows: $v = 5$, $\theta_{GA} = 50$, $\chi = 1$, $\mu = 0.04$, $\theta_{del} = 50$, $\theta_{sub} = 50$, $m_0 = 0.1$, $s_0 = 0.6$, $N = 6400$. Two points crossover and roulette wheel selection are used. For NLCS, each MLP has one hidden layer and one hidden node. The learning rate of MLPs is $\beta = 0.1$. Other parameters are: $v = 5$, $\theta_{GA} = 50$, $\chi = 1$, $\mu = 0.04$, $m_0 = 0.1$, $s_0 = 0.1$, $covering_window = 50$, $\gamma = 0.5$, population size $N = 25$.

Each learner performs three stratified ten-fold cross validation in each data set (that is 30 runs). Each run uses different random seeds which are consistent in all experiments. The

results reported in this paper are averaged over those 30 runs. We use the term *iteration* to refer to a single pass through the training set. The statistical data is collected after 500 iterations in each experiment.

V. AN INVESTIGATION OF THE NEURAL-BASED REPRESENTATION

This section is designed to answer our first research question about NLCS. We start by investigating the effect of the neural representation on UCS, followed by a comparison of UCS and NLCS in terms of accuracy and compactness. Several factors such as the population size, the size of the covering window, and the conflict resolution methods among NNs in the match set are also considered to provide a better understanding of NLCS.

A. Effects of the neural representation

To understand the effect of the neural representation on UCS, we need to compare the prediction accuracy of UCS with the traditional representation and our proposed neural representation.

For the first experiment, the genetic algorithm is switched off and a simple covering technique is used. If covering alone is sufficient to give high accuracy, the whole UCS mechanism within NLCS is not useful.

The top half of Table II presents the mean and the standard deviation (over 30 runs) of the predictive accuracy of the traditional and neural representations without GA. The statistical test of significance used is the t-test with a significance level of 0.05.

There is a statistically significant improvement in accuracy with the neural representation, in nine of the fourteen data sets (balance-scale, bupa, credit-a, glass, heart-statlog, iris, segment, tao, and vowel). The opposite trend is not observed in any data set. Moreover, ten data sets (balance-scale, bupa, credit-a, diabetes, ionosphere, iris, segment, sonar, tao, and vowel) have a smaller standard deviation with the neural representation. This experiment suggests that the neural representation has an advantage over the traditional representation, with improved and more consistent prediction accuracy.

The performance on several data sets is bad (e.g. heart-statlog (31%), ionosphere (29%), sonar (4%)). This can be attributed to disabling the GA component, which means the input space is explored only by the covering technique. If the covering process is not able to generalize between instances, the system will not be able to handle test cases in areas of the input space that are not represented in the training set.

The bottom half of Table II presents the mean and the standard deviation of the predictive accuracy of the traditional and neural representations with GA enabled. It can be easily observed that the accuracy of both UCS and NLCS with GA are better than accuracy without GA. This confirms that covering alone is not sufficient for accurate predictions, and that GA indeed plays a positive role in both UCS and NLCS.

NLCS performs significantly better than UCS on six data sets: the balance scale, bupa, Australian credit card, ionosphere, lymph, and sonar problems. Both systems have

TABLE II

THE MEAN AND STANDARD DEVIATION OF ACCURACY ON DIFFERENT PROBLEMS. ■(□) SYMBOLS INDICATE THAT THE NEURAL REPRESENTATION IS BETTER(WORSE) THAN THE TRADITIONAL REPRESENTATION AT A SIGNIFICANCE LEVEL OF .05

Problem	No GA	
	UCS	NLCS
balance-scale	0.788 ± 0.040	0.825 ± 0.034 ■
breast-w	0.759 ± 0.033	0.761 ± 0.040
bupa	0.549 ± 0.077	0.612 ± 0.066 ■
credit-a	0.716 ± 0.064	0.796 ± 0.048 ■
diabetes	0.676 ± 0.043	0.693 ± 0.043
glass	0.491 ± 0.079	0.555 ± 0.094 ■
heart-statlog	0.275 ± 0.091	0.319 ± 0.104 ■
ionosphere	0.288 ± 0.064	0.298 ± 0.064
iris	0.809 ± 0.128	0.929 ± 0.052 ■
lymph	0.566 ± 0.114	0.563 ± 0.126
segment	0.810 ± 0.036	0.829 ± 0.033 ■
sonar	0.045 ± 0.050	0.042 ± 0.047
tao	0.768 ± 0.087	0.847 ± 0.029 ■
vowel	0.487 ± 0.050	0.575 ± 0.050 ■

Problem	With GA	
	UCS	NLCS
balance-scale	0.815 ± 0.038	0.886 ± 0.022 ■
breast-w	0.969 ± 0.012	0.970 ± 0.015
bupa	0.683 ± 0.062	0.723 ± 0.055 ■
credit-a	0.835 ± 0.028	0.860 ± 0.032 ■
diabetes	0.748 ± 0.044	0.765 ± 0.042
glass	0.825 ± 0.078	0.830 ± 0.079
heart-statlog	0.648 ± 0.116	0.612 ± 0.071
ionosphere	0.729 ± 0.051	0.874 ± 0.065 ■
iris	0.949 ± 0.042	0.949 ± 0.067
lymph	0.759 ± 0.112	0.846 ± 0.094 ■
segment	0.968 ± 0.008	0.809 ± 0.055 □
sonar	0.736 ± 0.073	0.799 ± 0.073 ■
tao	0.887 ± 0.015	0.867 ± 0.023 □
vowel	0.910 ± 0.031	0.560 ± 0.072 □

achieved similar accuracy on five data sets: breast cancer, diabetes, heart statlog, glass, and iris. In three of these five the accuracy is marginally better with NLCS, but the difference is not statistically significant.

If we take a closer look at the result, the accuracy gain typically lies in the range [0.02, 0.08], as in balance-scale, bupa, credit-a, lymph, and sonar. Ionosphere shows the biggest gain of around 0.14.

UCS, on the other hand, does significantly better than NLCS on three data sets: segment, vowel, and tao. The segment and vowel problems have many classes (7 classes for the segment and 11 classes for the vowel) and high number of features (19 features for the segment and 13 features for the vowel). The Tao problem challenges NLCS due to its non-linear boundaries in the data.

We hypothesize that the reason for NLCS's low performance on these data sets is the use of a small population size, and low training time for the neural network.

NLCS learns more slowly than UCS in many data sets such as bupa, glass, iris, lymph, and vowel. Figure 5 shows the learning curves of both UCS and NLCS over time, for one illustrative data set. Each iteration is a complete feed of a training set (or exploration phase) following by a testing set (or exploitation phase).

NLCS normally requires extra time at the beginning for adjusting the weights of neural networks; hence the slow climb

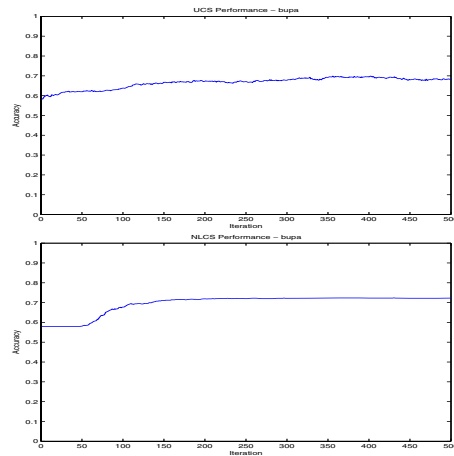


Fig. 5. The learning curves of UCS (top) and NLCS (bottom) over time. An iteration refers to a single pass of the training data set through the system.

in accuracy at the start of the training for NLCS. UCS, on the other hand, has only one action and therefore does not need this time. Training neural networks is slowest in problems with many classes, as a high number of weights needs to be adjusted. This is seen on the segment and vowel problems, for which accuracy grows gradually at the beginning. Longer training times are needed for problems with many features and classes. (This is investigated further in Section VI-D.)

Overall, our results show that NLCS has obtained an equivalent or better predictive accuracy than UCS on most of the tested data sets. Hence, a neural-based representation of the classifiers' action can be considered to be applicable, with good accuracy, to be used as a LCS for classification tasks.

B. The Effect of Population Size

UCS requires a few thousand classifiers to capture the whole input space on each problem. The results confirm the finding by Butz [9] that lower and upper boundaries in the classifiers require a large population size to approximate non-linear classification boundaries such as those in real-world data sets.

The results for NLCS in Table II are all obtained using a population of only 25 macro-classifiers, which even so yields similar or better predictions than UCS in many cases. It is clear that the neural representation can help to compact the population.

A classifier generated by the covering technique can be either general or specific. Since the covering spread used in our experiments is 0.6, it is likely to create more general classifiers than specific ones. The traditional representation allows each classifier to have one action, while our proposed method allows all possible actions to be associated with a classifier. If a classifier is too general, more than one class can be observed in the area. The traditional representation can only predict one action correctly, while the neural representation can predict multiple actions as long as enough training is provided. That might explain why the neural representation helps to compact the population, by allowing general classifiers in the population instead of only accurate classifiers as in UCS.

TABLE III

THE MEAN ACCURACY AND STANDARD DEVIATION WITH DIFFERENT POPULATION SIZES. ■ MEANS THAT ACCURACY WITH THAT POPULATION SIZE IS SIGNIFICANTLY BETTER THAN WITH A POPULATION SIZE OF 25

Population	1	10	25
balance-scale	0.865±0.023	0.884±0.021	0.886±0.022
breast-w	0.966±0.015	0.969±0.015	0.970±0.015
bupa	0.714±0.050	0.726±0.057	0.723±0.055
credit-a	0.858±0.040	0.857±0.034	0.860±0.032
diabetes	0.767±0.047	0.766±0.047	0.765±0.042
glass	0.597±0.073	0.607±0.083	0.612±0.071
heart-statlog	0.816±0.078	0.832±0.085	0.830±0.079
ionosphere	0.863±0.061	0.865±0.069	0.874±0.065
iris	0.967±0.034	0.944±0.066	0.949±0.067
lymph	0.842±0.101	0.844±0.099	0.846±0.094
segment	0.581±0.037	0.780±0.069	0.809±0.055
sonar	0.785±0.075	0.797±0.082	0.799±0.073
tao	0.842±0.014	0.863±0.023	0.867±0.023
vowel	0.415±0.046	0.523±0.065	0.569±0.072

Population	50	75	100
balance-scale	0.889±0.021	0.890±0.022	0.891±0.022
breast-w	0.968±0.016	0.967±0.015	0.968±0.016
bupa	0.721±0.055	0.721±0.053	0.723±0.054
credit-a	0.861±0.032	0.859±0.032	0.857±0.033
diabetes	0.764±0.041	0.766±0.043	0.766±0.043
glass	0.607±0.078	0.607±0.064	0.613±0.059
heart-statlog	0.826±0.080	0.825±0.078	0.824±0.076
ionosphere	0.874±0.068	0.870±0.068	0.875±0.066
iris	0.953±0.056	0.962±0.038	0.964±0.038
lymph	0.848±0.094	0.855±0.092	0.855±0.092
segment	0.831±0.044■	0.858±0.041■	0.847±0.043■
sonar	0.810±0.075	0.812±0.077	0.815±0.077
tao	0.871±0.020	0.872±0.020	0.874±0.022
vowel	0.614±0.069■	0.632±0.061■	0.632±0.060■

To understand the effect of the population size on the performance of NLCS, we experimented with population sizes of 1, 10, 25, 50, 75, and 100. Table III presents the performance of NLCS with different population sizes.

With many data sets there is a slight trend in favor of larger population sizes. In most cases, though, the difference is small and not statistically significant.

A larger population size is clearly helpful with the vowel and segment problems. This supports our hypothesis that the low accuracy of NLCS on segment, tao, and vowel problems is due to the population size.

In essence, the population size of 25 classifiers yields an acceptable performance on most data sets. From this point on, all our experiments use a population of 25 rules unless stated differently.

The population size has an impact on execution time. LCS starts by first comparing the condition of all classifiers in the population against each input instance. A population of m classifiers would require m times of comparison against the whole input instance. If there are n features in the input, we need mn comparisons for each input instance. Increasing m raises dramatically the computational cost of LCS. Thus, it clearly indicates that NLCS is better than UCS in terms of computation time. However, we also notice the cost of NLCS to back-propagate the error in order to adapt the weights of its MLPs. Since there are only 25 classifiers in the population and even less in the match set, we expect this time to be much smaller than the comparison time. In addition, with a

simple neural network architecture, back-propagation can be optimized to work faster than the traditional generic algorithm. The population size of 25 classifiers would be much faster in terms of execution time in comparison to thousands of classifiers.

C. The Match set $[M]$

Figure 6 plots the average match set size in proportion to the population size. The trend is that as the population size increases, the size of the match set decreases as a proportion of the total population size. This is associated with a reduction in the variance, thus the networks are almost distributed uniformly.

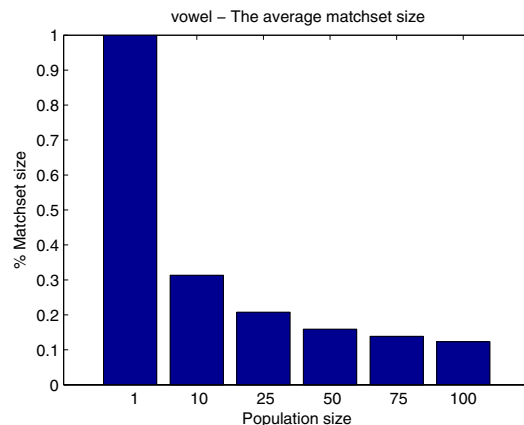


Fig. 6. The average match set size as a proportion of the population size in NLCS on the training data set

D. The Effect of the Covering Window

The covering approach is used in NLCS in two cases: when the match set $[M]$ is empty, and when it fails to classify a training instance.

In the first case, a new classifier is created to cover an instance that the system currently has no knowledge about. The covering classifier is added straight to the population and the match set. This process happens mainly at the beginning of the learning cycle.

The second case happens when the system produces incorrect predictions. The covering approach in this case aims to put extra knowledge of the area into the system, hoping that instances around that area will be predicted correctly in the future. Since the system already holds some knowledge about the instance (because $[M]$ is not empty), it is not as urgent to insert a new classifier into the population. Instead we use a covering window, which represents a particular number of training instances: at the end of each window, a new classifier is added to cover all instances that were classified incorrectly during that window. The size of the covering window affects the generality of the classifier: the larger the window, the more general the new classifier becomes as it must cover more instances.

Table IV shows the accuracy achieved on each data set with four different sizes of covering window: 10, 50, 75 and 200

TABLE IV

THE MEAN ACCURACY AND STANDARD DEVIATION WITH DIFFERENT COVERING WINDOW SIZES

Window Size	10	50	75	200
balance-scale	0.90 ± 0.02	0.89 ± 0.02	0.89 ± 0.02	0.89 ± 0.02
breast-w	0.97 ± 0.01	0.97 ± 0.02	0.97 ± 0.02	0.96 ± 0.02
bupa	0.72 ± 0.06	0.72 ± 0.06	0.73 ± 0.06	0.72 ± 0.05
credit-a	0.86 ± 0.03	0.86 ± 0.03	0.86 ± 0.03	0.86 ± 0.04
diabetes	0.76 ± 0.04	0.77 ± 0.04	0.76 ± 0.04	0.77 ± 0.05
glass	0.60 ± 0.08	0.61 ± 0.07	0.61 ± 0.08	0.60 ± 0.07
heart-statlog	0.83 ± 0.07	0.83 ± 0.08	0.83 ± 0.08	0.83 ± 0.09
ionosphere	0.88 ± 0.06	0.87 ± 0.06	0.87 ± 0.06	0.87 ± 0.07
iris	0.96 ± 0.04	0.95 ± 0.07	0.96 ± 0.05	0.97 ± 0.04
lymph	0.83 ± 0.08	0.85 ± 0.09	0.85 ± 0.10	0.84 ± 0.10
segment	0.89 ± 0.03	0.81 ± 0.05	0.79 ± 0.04	0.76 ± 0.05
sonar	0.79 ± 0.07	0.80 ± 0.07	0.81 ± 0.08	0.80 ± 0.07
tao	0.89 ± 0.02	0.87 ± 0.02	0.86 ± 0.02	0.85 ± 0.02
vowel	0.65 ± 0.04	0.57 ± 0.07	0.51 ± 0.08	0.46 ± 0.07

instances. On most data sets, the covering window does not seem to have much effect on the predictive accuracy of the system.

Once again, tao, segment and vowel behave differently. A small covering window, which means specific classifiers are more likely to be added than more general classifiers, is clearly better for these problems.

For the tao problem, this is explained by the problem containing non-linear boundaries. Specific classifiers are more accurate than general classifiers in this situation, because they are responsible for a smaller area. For vowel and segment, the explanation could be that it is hard for general classifiers to be accurate in problems with many output classes, so there is a preference for specific classifiers.

The covering window of size 10 or 50 tends to obtain good performance in all data sets.

E. Neural Network Ensembles

This section addresses the question of how to combine knowledge from different rules in the match set in the exploitation phase. We investigate three simple methods, which are quite popular for decision making at the gate level in ensemble learning:

- Majority voting (VOT) [32]: The output of the ensemble is the class which receives the vote of the majority of networks in the ensemble.
- Simple averaging (AVG) [23]: The output of the ensemble is the mean of the individual outputs of the network.

$$\hat{y}_{ens} = \sum_{i=1}^M \hat{y}_i / M \quad (20)$$

- Winner take all (WTA): The output of the ensemble is the output of the network whose output is maximally different from the classification threshold.

$$\hat{y}_{ens} = \arg_{max,i} (|\hat{y}_i - threshold|) i \in [1, M] \quad (21)$$

Each of these three methods was tested on the fourteen data sets. ANOVA tests were used to determine whether there were statistically significant differences in prediction accuracy using the different gates. There were statistically significant

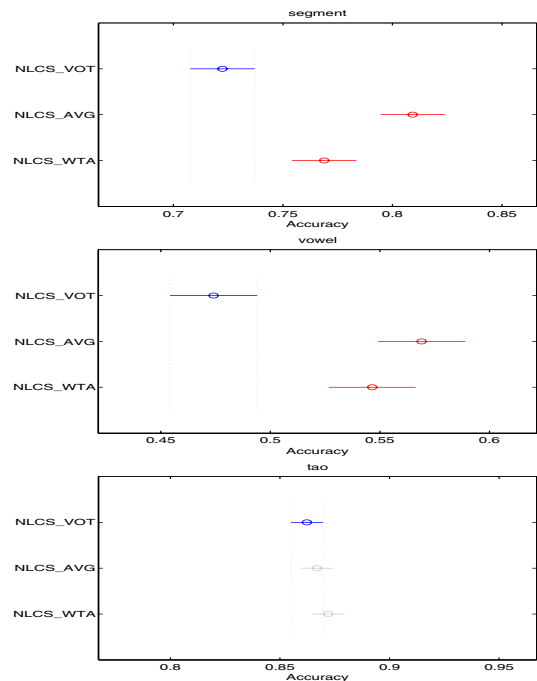


Fig. 7. Predictive accuracy of NLCS using different gates: majority voting (VOT), simple averaging (AVG), winner-takes-all (WTA)

differences in three data sets: segment, vowel, and tao. Figure 7 shows predictive accuracy with each gate on these three data sets.

The segment and vowel problems show the best accuracy when using the simple averaging approach. The winner-takes-all method is the second best, significantly outperforming the majority voting method.

The tao problem favors the winner-take-all approach, because this favors the more specific classifiers that suit a problem with non-linear boundaries. The difference is small though; it is statistically significant because it is consistent rather than because it is large.

There is no significant difference in the accuracy on the other eleven data sets. Six (breast-w, credit-a, diabetes, glass, iris, and lymph) do best with the simple averaging method; three (balance-scale, ionosphere, and sonar) do worst with simple averaging. The differences in mean accuracy with the different gates are small.

In summary, in the only data sets where the gate makes an important difference to mean accuracy, the simple averaging approach performs best. Hence, from this point in this paper, we only report the results of NLCS using the simple averaging approach.

F. Summary

This section has investigated the neural representation in learning classifier systems. We tested NLCS on several data sets under several conditions. The experiments show that:

- The neural representation does help to improve predictive accuracy, compared to the traditional representation;
- A searching method like genetic algorithms is essential to cover the input space;

- NLCS is able to obtain equal or even better accuracy than UCS, while requiring a much smaller population;
- Increasing the population size generally improves the accuracy in NLCS, but a population of only about 25 classifiers seems to be sufficient for acceptable accuracy;
- The size of the covering window does not have a major effect on the overall predictive performance of NLCS.
- To resolve conflicts between neural networks in the match set, the simple averaging approach is the best choice.

VI. NEGATIVE CORRELATION NEURAL LEARNING BASED CLASSIFIER SYSTEMS

A. Diversity and Accuracy in Ensembles

The notion of combining networks to form more reliable ensembles was first raised by Nilsson [29]. Ensembles normally perform better than single networks by minimizing loss of information due to bias. Sharkey [32] defined the notion of combining NNs for improving the performance in that we try exploiting rather than losing the information contained by imperfect networks.

In order for an ensemble to generalize well, two vital factors need to be considered: diversity and accuracy. Brown [5] suggests that two neural networks are diverse if they make different errors on the same data points/inputs. Accuracy refers to how good the learning model is in comparison to random guessing on a new input [5].

Two common methods to maintain the diversity within an ensemble are Bagging and Boosting. The Bagging method will randomly generate a new training set with a uniform distribution for each network member, from the original data set. The Boosting approach, on the other hand, re-samples the data set with a non-uniform distribution for each ensemble member. The whole idea of boosting and bagging is to improve the performance by creating some weak and biased classifiers. When aggregating these classifiers, using an average or other mechanisms, the bias of the ensemble is hoped to be less than the bias of an individual classifier. In short, boosting and bagging is used to diversify the ensemble in order to perform better in comparison to a non-diverse ensemble [33]. In LCS, the training data is inherently re-sampled by classifier conditions. Each NN is trained by partial data which belongs to its local region. Therefore, LCSs implicitly maintain diversity in the population without using Bagging or Boosting techniques.

Each time a rule appears in the match set $[M]$, its weights and fitness are updated. This raises another question in our research: how will the system perform if those individual networks in the match set are trained interactively like [19], [24], etc? It is similar to anti-correlation learning, where members in the ensemble are trained interactively so that each member specializes on a part of the task. Anti-correlation learning adds an additional penalty term into the error function in order to maximize the distance between all individuals in an ensemble to achieve a nice spread in the ensemble space. The negative correlation term should not have a larger magnitude than the original error function and is dimensionally consistent with the error function. Negative correlation learning (NCL) [24] is an approach of this type. McKay and Abbass [27] reveal

that negative correlation learning acts to push the members of the ensemble away from their mean, but not necessarily away from each other. We decided to blend NCL with NLCS by adding a negative term into the error function before back-propagating in those classifiers in the match set $[M]$.

B. The Effect of Negative Correlation Learning

This section focusses on learning the effects of NCL on NLCS by experimenting with the strength parameter γ . Several experiments were carried out, each with different values of γ in the range $[0.0, 0.7]$. $\gamma = 0.0$ implies that NCL is not used in the system, which was the case in all the experiments reported in Section V.

Table V shows accuracy achieved on each data set with each value of γ from 0.0 to 0.7. The statistical t-test was used to compare the means of predictive accuracy between NLCS with NCL ($\gamma > 0$) and NLCS without NCL ($\gamma = 0$).

In five out of fourteen data sets, NCL produces a statistically significant improvement in predictive accuracy. Within those five data sets, there is little significant difference in the performance when γ is small ($\gamma = 0.1$). As γ increases, we can observe significant improvement. However, there is also a significant decrease in the performance in most of the data sets when γ is high ($\gamma > 0.6$).

In general, five out of the fourteen data sets present an identifiable trend: increasing γ results in improvement of the predictive performance, up to a particular point, beyond which the performance starts decreasing. This result is consistent with the finding by Brown [4].

Some problems such as balance-scale, bupa, diabetes, glass, segment, and vowel suffer a huge decrease in accuracy when γ surpasses 0.6. With other data sets the decrease in accuracy is more gradual when γ is bigger than 0.6.

In conclusion, NCL does help to improve the accuracy of NLCS for medium γ values. If γ is greater than 0.6, it yields a dramatic decrease in the performance of the system. Thus, the key requirement for achieving better performance using NCL in NLCS is to tune the γ parameter carefully. As suggested by Brown [4] and also from our experimental results, $\gamma = 0.5$ seems to give the best accuracy.

C. Comparisons with NN Ensembles using NCL

In this section, we will compare NLCS with other NN ensembles using NCL in the literature. Evolutionary ensembles with negative correlation learning (EENCL) [26] is the first system of this type that combines global evolution with a local search based on gradient descent. The average testing accuracy of EENCL reported in [26] for the Australian credit card problem and the diabetes problem are respectively 0.865 and 0.779. Island model with Negative Correlation Learning (INCL) [11] is another system based on EENCL without implicit fitness sharing. INCL is reported to achieve the testing accuracy of 0.878 and 0.767 for the Australian credit card data set and the diabetes data set. NLCS has been able to achieve the testing accuracy of 0.860 and 0.768 for the Australian credit card problem and the diabetes problem. Hence, it is quite fair to state that NLCS is able to achieve the comparable accuracy as both EENCL and INCL on two data sets.

TABLE V

THE MEAN ACCURACY AND STANDARD DEVIATION WITH DIFFERENT VALUES OF $\gamma = 0$. ■(□) INDICATE THAT ACCURACY IS BETTER(WORSE) THAN ACCURACY WITHOUT NCL AT A SIGNIFICANCE LEVEL OF .05

γ	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
balance-scale	0.89±0.02	0.89±0.02	0.90±0.02■	0.90±0.02■	0.90±0.02■	0.90±0.01■	0.90±0.01■	0.79±0.07□
breast-w	0.97±0.02	0.97±0.01	0.97±0.01	0.97±0.01	0.97±0.01	0.97±0.01	0.97±0.02	0.96±0.02□
bupa	0.72±0.06	0.73±0.05	0.72±0.06	0.73±0.05	0.73±0.06	0.73±0.06	0.58±0.04□	0.58±0.04□
credit-a	0.86±0.03	0.86±0.03	0.86±0.03	0.86±0.03	0.86±0.03	0.86±0.04	0.86±0.03	0.85±0.03
diabetes	0.77±0.04	0.76±0.04	0.77±0.05	0.77±0.04	0.77±0.04	0.77±0.04	0.74±0.05□	0.66±0.02□
glass	0.61±0.07	0.61±0.07	0.62±0.07	0.62±0.05	0.61±0.06	0.60±0.07	0.45±0.06□	0.36±0.07□
heart-statlog	0.83±0.08	0.83±0.08	0.83±0.08	0.84±0.08	0.83±0.08	0.84±0.08	0.83±0.06	0.81±0.07
ionosphere	0.87±0.06	0.87±0.06	0.88±0.07	0.88±0.06	0.87±0.07	0.88±0.06	0.87±0.06	0.83±0.08□
iris	0.95±0.07	0.95±0.06	0.95±0.06	0.96±0.05	0.96±0.05	0.96±0.05	0.94±0.07	0.87±0.12□
lymph	0.85±0.09	0.85±0.10	0.85±0.09	0.85±0.09	0.84±0.08	0.84±0.08	0.84±0.08	0.85±0.08
segment	0.81±0.05	0.85±0.04■	0.88±0.03■	0.90±0.02■	0.92±0.02■	0.92±0.02■	0.76±0.07□	0.44±0.07□
sonar	0.80±0.07	0.80±0.08	0.82±0.07	0.82±0.07	0.83±0.07■	0.83±0.06■	0.81±0.07	0.75±0.08□
tao	0.87±0.02	0.87±0.02	0.88±0.02■	0.89±0.02■	0.89±0.02■	0.89±0.02■	0.88±0.02■	0.81±0.05□
vowel	0.57±0.07	0.61±0.05■	0.62±0.06■	0.63±0.06■	0.64±0.05■	0.66±0.05■	0.67±0.06■	0.27±0.07□

D. Population Size and Training Epochs

This section investigates three special data sets — segment, tao, and vowel — because of the low performance of NLCS compared to the traditional UCS.

NCL helps to improve significantly the predictive accuracy of these data sets especially when $\gamma = 0.5$ as shown in Table V. The accuracy of the segment problem raises from 81% without NCL to 92% with NCL. The accuracy of the tao problem increases from 87% to 89%. The accuracy of the vowel problem increases from 57% to 66%. The tao problem seems to be able to achieve the same level of accuracy as UCS (average accuracy of UCS is 88% and average accuracy of NLCS is 89%). However the accuracy of the segment problem is still worse than UCS at 92% compared to 96%. Also, the accuracy of the vowel problem is much worse than UCS at 66% compared to 91%.

We hypothesize that these data sets perform significantly worse than UCS due to two main reasons: small population size and small number of training epochs. We noted in section V-A that NLCS learns more slowly than UCS in several data sets.

The vowel and segment problems are similar in that they have many possible outcomes (seven for segment and eleven for vowel), and many learning features (nineteen for segment and thirteen for vowel). Therefore the number of weights of each neural network increases dramatically in both problems. It requires more time for each neural network to adjust these weights correctly before they can be used. The tao problem, in contrast, is a binary classification problem with a small number of features. Its challenge is the non-linear boundaries that favor specific classifiers.

Table VI presents the mean and standard deviation of accuracy on these three data sets using a larger population size (of 100). Accuracy improves on all three data sets (as also seen earlier in table III), and peaks at $\gamma = 0.5$. The tao problem reaches the same level of accuracy as UCS. However, the performance of NLCS on the segment and vowel problems is still not as good as UCS.

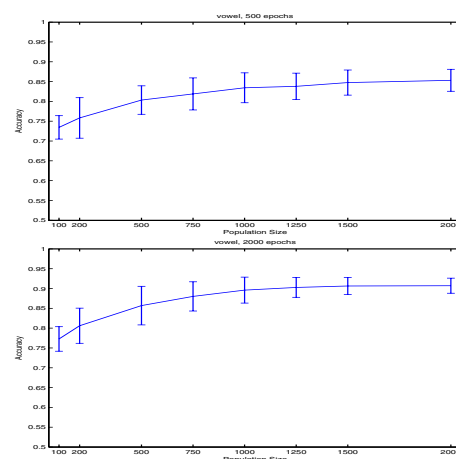


Fig. 8. Predictive accuracy of NLCS with different population sizes and training time

Figure 8 shows the mean and standard deviation of accuracy on the vowel problem, with different population sizes and different numbers of training epochs. Increasing the population size clearly leads to an improvement in the predictive accuracy.

Moreover, the number of epochs (or the number of time a whole training set is fed through the system) has a strong influence on the accuracy of the vowel problem. As mentioned before, neural networks in the vowel problem are more complex than in other problems since there are more weights to tune. In order for each neural network to gain enough information, a longer training time is needed. As we can see, feeding training data through the system 2000 times instead of 500 times improves the accuracy, to a level that now matches UCS.

A similar pattern applies for segment, though it is not as strong and still does not quite achieve the accuracy of UCS.

Another factor that would degrade the performance of the system is over-fitting. In fact, NLCS employs single hidden networks, which are very simple and would not be over-fitted.

TABLE VI

POPULATION SIZE 100. THE MEAN ACCURACY AND STANDARD DEVIATION WITH DIFFERENT VALUES OF $\gamma = 0$. ■(□) INDICATE THAT ACCURACY IS BETTER(WORSE) THAN ACCURACY WITHOUT NCL AT A SIGNIFICANCE LEVEL OF .05

γ	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
segment	0.84±0.04	0.88±0.03 ■	0.89±0.03 ■	0.92±0.02 ■	0.93±0.02 ■	0.93±0.02 ■	0.75±0.06 □	0.58±0.10 □
tao	0.87±0.02	0.88±0.02	0.89±0.02 ■	0.90±0.01 ■	0.90±0.01 ■	0.90±0.02 ■	0.88±0.02 ■	0.83±0.02 □
vowel	0.63±0.06	0.64±0.05	0.64±0.06	0.65±0.06 ■	0.68±0.04 ■	0.74±0.05 ■	0.28±0.04 □	0.26±0.04 □

Also, the generalization pressure in non-enhanced UCS/LCS systems prevents it from over-fitting as presented in [31]. Therefore over-fitting in NLCS is not a concern.

Hence, the results in this section support our hypothesis that the population size and the training time have some impact on the predictive accuracy.

VII. CONCLUSION

In this paper, we argued that the predictive accuracy is not the only factor to judge a classifier system. The compactness and expressiveness of the system are other important issues in traditional machine learning. We proposed a neural representation in LCS to balance these factors.

Our first research question was whether the neural representation is beneficial. We considered this by testing UCS and NLCS on fourteen data sets. The t-test results show that NLCS performs equivalently to UCS on five data sets, significantly better on six data sets, but significantly worse on three data sets.

The population size also plays an important role in comparing the performance of UCS and also NLCS. In all tested data sets, UCS requires at least a few thousand rules to capture the whole search space. NLCS, in general, requires only 25 rules to perform as well as UCS. The population size of 25 classifiers would be much faster in terms of execution time in comparison to thousands of classifiers. Thus, the neural-based representation is indeed beneficial to LCS.

Our second research question was whether negative correlation learning (NCL) will improve the performance of NLCS. We tested NLCS with different values for the discount rate, γ . The results show that NCL improve the predictive performance of NLCS on several data sets when $\gamma < 0.6$. We conclude that $\gamma = 0.5$ is the best value to use in NLCS in order to achieve good performance in most problems.

Three data sets (tao, vowel, and segment) were investigated further. We found that the population size and the training time have an impact on these three data sets, especially on the vowel data set.

Future research will involve introducing an adaptive population size in our model, investigating the system in dynamic environments, and developing ways to visualize the area covered by classifiers and the neural network's boundaries.

REFERENCES

- [1] H. A. Abbass, M. Towsey, and G. D. Finn. C-Net: A method for generating non-deterministic and dynamic multivariate decision trees. *Knowledge and Information Systems*, 3(2):184–197, 2001.
- [2] E. Bernadó-Mansilla and J. M. Garrell-Guiu. Accuracy-based learning classifier systems: models, analysis and applications to classification tasks. *Evolutionary Computation*, 11(3):209–238, 2003.
- [3] E. Bernadó-Mansilla, X. Llorà, and J. M. Garrell-Guiu. XCS and GALE: a comparative study of two learning classifier systems with six other learning algorithms on classification tasks. In *Proceedings of the 4th International Workshop on Learning Classifier Systems (IWLCS-2001)*, pages 337–341, 2001. Short version published in Genetic and Evolutionary Computation Conference (GECCO2001).
- [4] G. Brown. *Diversity in Neural Network Ensembles*. PhD thesis, School of Computer Science, The University of Birmingham, 2004.
- [5] G. Brown, J. Wyatt, R. Harris, and X. Yao. Diversity creation methods: a survey and categorisation. *Information Fusion*, 6(1):5–20, 2005.
- [6] L. Bull. On using constructivism in neural classifier systems. In *Parallel Problem Solving from Nature - PPSN VII*, pages 558–567, 2002.
- [7] L. Bull and T. O'Hara. Accuracy-based neuro and neuro-fuzzy classifier systems. In *GECCO '02: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 905–911, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.
- [8] M. V. Butz. *Rule-based Evolutionary Online Learning Systems: Learning Bounds, Classification, and Prediction*. PhD thesis, University of Illinois at Urbana-Champaign, 2004.
- [9] M. V. Butz. Kernel-based, ellipsoidal conditions in the real-valued XCS classifier system. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1835–1842, New York, NY, USA, 2005. ACM Press.
- [10] H. H. Dam, H. A. Abbass, and C. Lokan. Be real! XCS with continuous valued inputs. In *Proceedings of the Eighth International Workshop on Learning Classifier Systems (IWLCS-2005)*, Washington D.C., USA, 2005.
- [11] P. Duell, I. Fermin, and X. Yao. Speciation techniques in evolved ensembles with negative correlation learning. In *IEEE Congress on Evolutionary Computation*, pages 16–21, Vancouver, Canada, 2006.
- [12] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishing Company, INC., 1989.
- [13] S. Haykin. *Neural networks, a comprehensive foundation*. Prentice-Hall, Inc., second edition, 1999.
- [14] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975. Republished by the MIT press, 1992.
- [15] J. H. Holland. Properties of the bucket brigade. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 1–7, Mahwah, NJ, USA, 1985. Lawrence Erlbaum Associates, Inc.
- [16] J. H. Holland. Escaping Brittleness: The Possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems. In Mitchell, Michalski, and Carbonell, editors, *Machine Learning, an Artificial Intelligence Approach. Volume II*, chapter 20, pages 593–623. Morgan Kaufmann, 1986.
- [17] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Network*, 2(5):359–366, 1989.
- [18] J. Hurst and L. Bull. A self-adaptive neural learning classifier system with constructivism for mobile robot control. In *Parallel Problem Solving from Nature - PPSN VIII*, pages 942–951. Springer, 2004.
- [19] M. M. Islam, X. Yao, and K. Murase. A constructive algorithm for training cooperative neural network ensembles. *IEEE Transactions on Neural Networks*, 14(4):820–834, 2003.
- [20] T. Kovacs. What should a classifier system learn and how should we measure it? *Journal of Soft Computing*, 6(3–4):171–182, June 2002.
- [21] P. L. Lanzi and A. Perrucci. Extending the representation of classifier conditions part I: From messy coding to S-expressions. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 337–344. Morgan Kaufmann, 1999.
- [22] P. L. Lanzi and A. Perrucci. Extending the representation of classifier conditions part II: From messy coding to S-expressions. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and

- R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 345–352. Morgan Kaufmann, 1999.
- [23] W. P. Lincoln and J. Skrzypek. Synergy of clustering multiple back propagation networks. *Advances in neural information processing systems 2*, pages 650–659, 1990.
- [24] Y. Liu. *Negative Correlation Learning and Evolutionary Design of Neural Network Ensembles*. PhD thesis, University College, The University of New South Wales, Australian Defence Force Academy, Canberra, Australia, 1999.
- [25] Y. Liu and X. Yao. Negatively correlated neural networks can produce best ensembles. *Australian journal of intelligent information processing systems*, 4(3/4):176–185, 1997.
- [26] Y. Liu, X. Yao, and T. Higuchi. Evolutionary ensembles with negative correlation learning. *IEEE Transactions on Evolutionary Computation*, 4(4):380–387, 2000.
- [27] R. McKay and H. Abbass. Anti-correlation: A diversity promoting mechanism in ensemble learning. *The Australian Journal of Intelligence Information Processing Systems*, 7(3/4):139–149, 2001.
- [28] D. Newman, S. Hettich, C. Blake, and C. Merz. UCI repository of machine learning databases, 1998.
- [29] N. J. Nilsson. *Learning Machines: Foundations of Trainable Pattern-Classifying Systems*. McGraw-Hill, 1965.
- [30] S. Quartz and T. Sejnowski. The neural basis of cognitive development: A constructivist manifesto. *Brain and Behavioral Sciences*, 20(4):537–596, 1997.
- [31] K. Shafi, H. Abbass, and W. Zhu. The role of early stopping and population size in xcs for intrusion detection. In *6th International Conference on Simulated Evolution and Learning (SEAL'06)*, pages 50–57, Hefei, China, 2006.
- [32] A. Sharkey. On combining artificial neural nets. *Connection Science*, 8:299–313, 1996.
- [33] M. Skurichina, L. Kuncheva, and R. Duin. Bagging and boosting for the nearest mean classifier: Effects of sample size on diversity and accuracy. In *International Workshop on Multiple Classifier Systems*, pages 62–71. Springer, 2002.
- [34] S. F. Smith. *A Learning System Based on Genetic Adaptive Algorithms*. PhD thesis, University of Pittsburgh, 1980.
- [35] S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.
- [36] S. W. Wilson. Generalization in the XCS classifier system. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 665–674, University of Wisconsin, Madison, Wisconsin, USA, 1998. Morgan Kaufmann.
- [37] S. W. Wilson. Get real! XCS with continuous-valued inputs. In P. Lanzi, W. Stolzmann, and S. Wilson, editors, *Learning Classifier Systems, From Foundations to Applications, LNAI-1813*, pages 209–219, Berlin, 2000. Springer-Verlag.
- [38] S. W. Wilson. Mining oblique data with XCS. In P. L. Lanzi, W. Stolzmann, and S. W. Wilson, editors, *Proceedings of the Third International Workshop (IWLCS-2000), Lecture Notes in Artificial Intelligence*, pages 158–174, 2001.



Hai Huang (Helen) Dam received the B.Sc. degree (with honors) from Curtin University of Technology, M.S. degree from University of Western Australia in 2002 and 2004, respectively. She is currently working towards the Ph.D. degree in Computer Science at the Artificial Life and Adaptive Robotics Laboratory, University of New South Wales at the Australian Defence Force Academy, in Canberra, Australia. Her research interests are stream data mining, distributed data mining, learning classifier systems, and evolutionary algorithms.



Hussein Abbass (SM05) is a Professor and Chair of Information Technology at the University of New South Wales at the Australian Defence Force Academy (UNSW@ADFA) in Canberra, Australia. He has a BA, B.Sc., PG-Dip, and Masters all from Cairo University Egypt, a M.Sc. from Edinburgh University Scotland, and a PhD from QUT Australia.

He is the Director of the Artificial Life and Adaptive Robotics Laboratory at UNSW@ADFA, an Advisory Professor at Vietnam National University, Ho-Chi Minh City, a senior member of the IEEE, a senior member of the Australian Computer Society (ACS), the chair of ACS National Committee on Complex Systems, the chair of the IEEE Task Force on Complex Adaptive Systems and Artificial Life, and a member of a number of national and international committees including the IEEE technical committee on Data Mining and the IEEE working group on soft computing in the SMC society.

He has 170+ refereed papers and is particularly interested in modelling and simulation of Complex Systems including free flight air traffic management. On a fundamental level, he works on Artificial Neural Networks, Ensemble Learning, Evolutionary Computation, Multi-Agent Systems, and Multi-objective Optimisation. He has been a technical co-chair and a member of the technical committee for many conferences in the field including a PC co-chair for CEC07, SEAL 06, and IEEE-Alife07.



Australian Software Metrics Association.

Chris Lokan is a Senior Lecturer at the University of New South Wales (Australian Defence Force Academy campus), in Canberra. His teaching and research concentrate on software engineering and software metrics. His main research interests are software size measures, software effort and cost estimation, and software benchmarking. Recently his research has concentrated on the use of multi-company datasets for estimation, and data mining using genetic algorithms. Chris is a member of the ACM, the Computer Society of the IEEE, and the



Xin Yao (M'91-SM'96-F'03) received the B.Sc. degree from the University of Science and Technology of China (USTC), Hefei, Anhui, in 1982, the M.Sc. degree from the North China Institute of Computing Technology, Beijing, in 1985, and the Ph.D. degree from USTC in 1990.

He was an Associate Lecturer and Lecturer from 1985 to 1990 at USTC, while working towards his Ph.D on simulated annealing and evolutionary algorithms. He took up a Postdoctoral Fellowship in the Computer Sciences Laboratory, Australian National University, Canberra, in 1990, and continued his work on simulated annealing and evolutionary algorithms. He joined the Knowledge-Based Systems Group, CSIRO Division of Building, Construction and Engineering, Melbourne, in 1991, working primarily on an industrial project on automatic inspection of sewage pipes. He returned to Canberra in 1992 to take up a lectureship in the School of Computer Science, UNSW@ADFA, where he was later promoted to a Senior Lecturer and Associate Professor. Attracted by the English weather, he moved to the University of Birmingham, U.K., as a Professor of Computer Science in 1999. Currently, he is the Director of the Centre of Excellence for Research in Computational Intelligence and Applications and a Changjiang Chair (Visiting) Professor (Cheung Kong Scholar) at the University of Science and Technology of China, Hefei. He is the Editor-in-Chief of the IEEE Transactions on Evolutionary Computation, an associate editor or editorial board member of several other journals, and the Editor of the World Scientific Book Series on Advances in Natural Computation. He has given more than 50 invited keynote and plenary speeches at conferences and workshops worldwide. His major research interests include evolutionary artificial neural networks, automatic modularization of machine learning systems, evolutionary optimization, constraint handling techniques, computational time complexity of evolutionary algorithms, coevolution, iterated prisoner's dilemma, data mining, and real-world applications. He has more than 250 refereed publications. He was awarded the President's Award for Outstanding Thesis by the Chinese Academy of Sciences for his Ph.D. work on simulated annealing and evolutionary algorithms in 1989. He won the 2001 IEEE Donald G. Fink Prize Paper Award for his work on evolutionary artificial neural networks.