A Self-Organized, Distributed, and Adaptive Rule-Based Induction System

Pornthep Rojanavasu, Hai H. Dam, Hussein A. Abbass Senior Member, IEEE, Chris Lokan and Ouen Pinngern

Abstract—Learning Classifier Systems (LCSs) are rule-based inductive learning systems that have been widely used in the field of supervised and reinforcement learning over the last few years. This paper employs UCS, a supervised learning classifier system, that was introduced in 2003 for classification tasks in data mining.

We present an adaptive framework of UCS on top of a self-organized map (SOM) neural network. The overall classification problem is decomposed adaptively and in real-time by the SOM into subproblems, each of which is handled by a separate UCS. The framework is also tested with replacing UCS by a feed-forward artificial neural network (ANN).

Experiments on several synthetic and real data sets, including a very large real data set, show that the accuracy of classifications in the proposed distributed environment is as good or better than in the non-distributed environment, and execution is faster. In general each UCS attached to a cell in the SOM has a much smaller population size than a single UCS working on the overall problem; since each data instance is exposed to a smaller population size than in the single population approach, the throughput of the overall system increases.

The experiments show that the proposed framework can decompose a problem adaptively into subproblems, maintaining or improving accuracy and increasing speed.

I. INTRODUCTION

Traditionally, a Learning Classifier System (LCS) [22] is a rule-based machine learning system that combines reinforcement learning and genetic algorithms. The reinforcement learning component is responsible for adjusting the strength of rules in the system according to some reward obtained from the environment. The genetic algorithm acts as an innovation discovery component which is responsible for discovering new — hopefully better and/or more general — rules. The main advantages of this approach are: (1) the rule-based representation, which makes it easy to understand the learnt knowledge; (2) the online learning ability, where the system is updated after the presentation of each data instance, which opens opportunities for this approach to be used in stream data mining; and (3) its robustness due to the use of genetic algorithms, which has been shown to explore the rule-space efficiently.

Traditionally, the fitness of a classifier in LCS is a function of the actual prediction. In 1995, XCS [39][40] emerged as a type of LCS where the classifier’s fitness depends on the prediction of its expected payoff, which worked better on both reinforcement and supervised learning problems. An investigation [38] on a benchmark synthetic test-bed, the Monk’s problem, showed that XCS’s performance is at least as good as traditional machine learning techniques. Subsequently, three studies [5], [9][10], [16] undertook comparisons using real-world problems and showed that XCS is competitive for data-mining problems in comparison to other non-evolutionary learning algorithms in terms of their predictive accuracy. The first study compared XCSs with 0-R, IB1, IBk, NBa, C4.5, PART, and SMO on 15 datasets. The second study compared XCS with C4.5, the Naive Bayes classifier, PART, the instance based learning algorithm with one and three nearest neighbor settings, and the support vector machine on 42 datasets. Statistical tests of significance showed that XCS outperforms other algorithms in some datasets. The last study applied XCS to twelve datasets and also found that XCS has the potential to be a powerful data mining tool.

In 2003, Bernadó-Mansilla et.al. [4] proposed a specialized version of XCS for supervised learning, which they called UCS (supervised classifier system). UCS uses the actual labels in the data to directly judge on the accuracy of a classifier. The study found that UCS is able to learn faster than XCS with smaller population size, especially on problems with multiple classes.

With the increase in the volume of data collected in organizations, data mining techniques need to be able to handle large volume of data. When LCSs are applied to these problems, their rule-based representation normally requires a large number of classifiers. For problems with non-linear boundaries, the population size is even larger because of the axis-parallel hyper-plane rule representation. A large population size has a dramatic impact on the speed for processing incoming data traffic. Each time a data instance arrives, the population needs to be scanned to find those rules that match the incoming data instance. Consequently, using LCSs for real-time stream data mining is hindered by the required large population size.

Bagnall and Cawley [3] and [15] investigated the performance of XCS on large datasets, and found that they pose a challenge for XCS. This problem has been studied by a number of researchers, trying to introduce mechanisms to cope with large datasets in LCSs. Some proposed to compact...
the population by changing the representation [14][29], some proposed distributed and ensemble frameworks [12][8], while others offered some efficient encoding schemes [31].

Automatic problem decomposition of the learning problem is one direction that holds much promise, but has not been investigated in LCSs. The idea is to automatically decompose the problem so that each subproblem is handled by a different learning machine. This is hoped to reduce the complexity of the learning task; thus it will require a less complex learning machine, which can then boost the speed of processing incoming traffic. A study by Maimon and Rokach [33] showed several benefits of decomposing data mining problems: decomposition (1) improves the classification accuracy; (2) provides conceptual simplification of the problem; (3) enhances the feasibility of being able to handle larger databases; and (4) provides clearer and more comprehensible results. There are also benefits in terms of reducing the runtime by solving smaller problems, and the opportunity to use different learning machines for individual subproblems.

The objective of this paper is to provide a framework to adaptively decompose a problem using a pre-gate that is tightly coupled with a classifier. In this paper, we hypothesize that having a self-organized map (SOM) as a pre-gate for decomposing the search space would reduce the complexity of the problem. Moreover, having a separate classifier to learn each of these subproblems could improve the accuracy in a few cases because each classifier only handles a possibly simpler problem. We investigate two different machine learning classification methods: UCS as a representative of the evolutionary school, and feed-forward artificial neural network (ANN) as a traditional non-evolutionary classifier. Several visualization techniques are employed to shed light on the resultant decomposition.

The rest of this paper is organized as follows. Section II provides a brief literature review, followed by a description of the proposed framework in Section III. Section IV describes the design of the experiments, while the following three sections focus on the analysis of the results. Conclusions are drawn and future work is discussed in Section VIII.

II. BACKGROUND MATERIALS

A. Problem Decomposition

A decomposition approach in classification can be categorized into one of two types [32]:

- Intermediate concept decomposition, where the problem is projected onto an intermediate space upon which the decomposition takes place to partition that space into smaller sub-spaces. For example, one may use Gaussian kernels to do the mapping to the intermediate space. Another example is decomposing the weight space of a neural network to get the networks specializing on doing different activities.

- Original concept decomposition, where problem decomposition is undertaken directly on the input space without a mediator. The original problem is divided into several subproblems by partitioning the training set. The mixture of experts model is an example of this category, where a gate is used to weight the classifiers differently based on different input instances. Thus, decomposition occurs on the input space.

This paper will focus on the second approach. Original concept decomposition can be divided further into the sample-based and space-based approaches. Bagging is one of the most well-known methods using the sample-based approach [7]. In bagging, sampling with replacement from the training data is used to create different datasets, each with its own bias. A classifier is built using each of these datasets, then a simple voting gate is used to combine the output.

Alternatively, boosting [19] is used to allow data instances which are incorrectly predicted by previous classifiers to be weighted higher than other data instances, thus allowing them to be chosen more often than examples that were correctly predicted to build a new classifier. Therefore, boosting attempts to produce new classifiers that are better able to predict data for which the current ensemble’s performance is poor.

In ensemble learning, a problem is decomposed (either explicitly or implicitly) into several subproblems. Each subproblem is then handled by a classifier. The final decision is made by an appropriate classifier. The mixture of experts (ME) model [23] is an example of the space-based decomposition. In this approach, the mixture of experts decomposes the input space, such that each expert examines a different part of the space. Each output of an expert is seen as a conditional probability for the given input vector. A gating network is responsible for combining the opinions of the various experts by assigning a weight to each network. Jordan and Jacobs [24] introduced the hierarchical mixtures of experts (HME) as an extension of the ME model. HME decomposes the space into sub-spaces, and then recursively decomposes each sub-space to further sub-spaces.

Automatic problem decomposition demands an explicit mechanism to ensure that the ensemble members are specialized on different (but potentially overlapping) sub-spaces. An ensemble of learning machines would normally require a gate. According to Abbass [1], there are two types of gate: a pre-gate and a post-gate. An explanation of each is given below and also see Figure 1.

1) A pre-gate acts as a router that assigns the incoming traffic to different directions based on some criteria. Research on pre-gates is mainly focusing on getting members of the ensemble to specialize on different parts of the input space through a partitioning or clustering function. The decision on which learning machine will get which instances can be done independently from the individual learning machines. However, this approach suffers from its inflexibility in adapting the decomposition when concept drift occurs. The separation between the decomposition from the learning problem also has its own limitation in the sense that the decomposition assumes that all learning machines are homogenous and are performing equally well. Another branch of research on pre-gates is competitive learning, where different learning systems compete on the input until it is taken up by one of them.
Fig. 1. A generic description to ensemble research.

2) A post-gate acts as an aggregate function that combines the (potentially different) opinions made by the individual members of the ensemble. The motivation for this learning approach comes from the area of expert systems, where a committee of experts is used to make decisions. If only one member responds to each situation, there will be no conflict within the committee. The problem becomes similar to the use of pre-gates with one-to-one or many-to-one mapping between clusters in the input space and ensemble members. However, if a group of committee members responds to the same input, conflicts between their opinions may happen. The issue is then how to combine these opinions into a single coherent opinion. Post-gates used in the literature can be very simple such as voting and winner-take-all, or more complex such as mixture of experts [23] and co-evolutionary mixture of experts [34][35].

Problem decomposition does not need to rely on a pre-gate alone or a post-gate alone. It can involve an ensemble of ensembles with a pre-gate to decompose a problem and a post-gate to combine opinions.

Recently, Rokach et. al. [37] proposed the K-classifier algorithm, which employs the k-means clustering algorithm to decompose a dataset into mutually exclusive subsets. Each subset is then handled by a classifier. This type of decomposition can be seen as a pre-gate, where an incoming data instance is routed to an appropriate classifier based on which cluster it belongs to. Their experiments, using decision trees, neural networks, and naive Bayes, showed that the proposed method is well suited for datasets of numeric input attributes.

One main disadvantage of this approach is the separation between the decomposition problem and the classification problem. By using the K-means clustering algorithm, the dataset needs to be processed completely off-line before being fed to the classifier.

In this paper, we overcome this problem by coupling the decomposition with the classifier. The idea is that by using a clustering algorithm that is self-organizing itself on the dataset, the classifiers associated with the different (changing) clusters adaptively learn the classification problem.

The tight coupling between the decomposition and the classifier has a number of advantages: (1) If the decomposition is separated from the classification in a two stage process, all data needs to exist before hand to decompose the problem then classification is applied on each sub-problem. This makes it hard to use this system for stream data mining, where the data arrives in real time in a huge quantity; (2) If there is a concept drift (i.e. the decision boundary changes), a system that separates decomposition from classification would require re-decomposing the problem. But this system would also need a mechanism to detect that the environment has changed, to decompose the problem, and then allow the classifier to recover from the changes in the environment. Tightly coupling the decomposition with the classification would make the system adaptive, where the decomposition is continuously occurring as the classifier learns; (3) the separation between decomposing a problem and learning a proper classification sounds artificial; after all, it is a single problem that we are tackling, of how to learn a proper classification model.

This tight coupling can be seen in the eyes of competitive learning, which is unsupervised learning that uses the concept of competition to group input patterns into classes based on their features. A common goal of competitive learning algorithms is to distribute a certain number of vectors in a possibly high-dimensional space. The distribution of these vectors should reflect (in one of several possible ways) the probability distribution of the input signals which in general is not given explicitly but only through sample vectors [20].

For example, in Figure 2 the symbol ‘■’ represents the weight vectors in the output layer. The initial state of weight vectors is random. After learning, the weights move towards the centroid of each cluster of the input space.

There are many models in the area of competitive learning, which are usually categorized in two main categories: hard and soft competitive learning. In hard competitive learning, there is only one winning unit; that is a winner-take-all strategy. However, in soft competitive learning, there can be more than one unit adapting its weight vector at a time. Each unit will adapt its weight vector relative to its proximity to the input pattern; that is a winner-take-most strategy.

In this paper we use Kohonen’s Self-Organizing Map (SOM) for soft competitive learning, and tightly couple it with an LCS. We experiment with UCS and ANN, as representatives of evolutionary and non-evolutionary classifiers respectively. The next three sections will introduce SOM, UCS and ANN.
B. Self-Organizing Map (SOM)

One of the most well known algorithms for soft competitive learning is SOM artificial neural network [25][26]. Figure 3 shows a 7x7 SOM.

The model consists of two layers. First, the output layer consists of neural elements called nodes. Each node \( i \) is assigned an \( n \)-dimensional weight vector \( m_i \). That is, \( m_i \in \mathbb{R}^n \), where \( \mathbb{R}^n \) is an \( n \)-dimensional space. It is necessary to note that the weight vectors have the same dimensionality as the input vector. Second, the input layer consists of input vectors for training the map. The learning process of a SOM can be seen in terms of continuous adaptation of the nodes for input vectors. In each learning iteration \( t \), input vector \( x(t) \) is fed into every node in the map to identify the output vector’s winning node. It is common to use the Euclidean distance as the basis to measure similarities. The winning node \( c \) can thus be defined as the one with the smallest distance (maximum similarity) to the input vector:

\[
c : m_c(t) = \min_i ||x(t) - m_i(t)||
\]

(1)

The weight of the winning node \( c \) is tuned by the difference between the input vector and the weight vector. Not only the winning node is learning but also its neighborhood nodes are learning as well. Therefore, their weight vectors become more similar to the input pattern. As a result, the respective node is more likely to win at future presentations of this input pattern. The weight vectors of the winning node and its neighbors at time \( t+1 \) are updated as follows:

\[
m_i(t+1) = m_i(t) + \alpha(t) \times h_{ci}(t) \times [x(t) - m_i(t)]
\]

(2)

where \( t \) is the current time, \( x(t) \) is the current input vector, \( m_i(t) \) is the current weight vector, \( \alpha(t) \) is the learning rate, which is reduced gradually after each learning iteration as follows:

\[
\alpha(t) = \alpha(0) \times \frac{T - t}{T}
\]

(3)

where \( T \) is the total number of iterations. \( h_{ci}(t) \) is the neighborhood function. For convergence, it is necessary that \( h_{ci}(t) \to 0 \) as \( t \to \infty \). This means that as learning progresses, the neighborhood within which the nodes are activated will shrink and the rate for modifying the reference vectors will decrease. Usually a Gaussian function is used as the neighborhood function, as follows:

\[
h_{ci}(t) = \exp\left( -\frac{||r_c - r_i||^2}{2\sigma^2(t)} \right)
\]

(4)

where the vectors \( r_c \) and \( r_i \) represent the coordinates of the winning node \( c \) and neighbor node \( i \), \( ||r_c - r_i|| \) is the distance between nodes \( c \) and \( i \), \( \sigma(t) \) is the radius of the neighborhood node, updated as follows:

\[
\sigma(t+1) = 1 + (\sigma(t) - 1) \times \frac{T - t}{T}
\]

(5)

C. UCS

UCS is a rule-based evolutionary learning classifier system, in which each classifier represents a partial solution to the target classification problem. The goal of UCS is to evolve a population of classifiers that represents a complete solution to the problem.
Each classifier in UCS consists of a *Condition* (the body of the rule), an *Action* (the prediction of the classifier) and some parameters. The *Condition* refers to several environmental states, to which the classifier may match. The *Action* is the outcome if the classifier fires. The key parameter associated with a classifier is the fitness $F$, which measures how good the classifier is relative to the rest of the population. Another two important parameters are the *numerosity* and *experience*. A classifier of UCS is a macro–classifier, which holds a distinct rule (a unique pair of *Condition:Action*) within the population $[P]$. The numerosity parameter records the number of copies of the classifier in the population. Whenever a new classifier is introduced, the population is scanned through to check if a copy of the classifier already exists. If it exists the numerosity value is incremented by one, otherwise the classifier is added to the population and its numerosity value is initialized to 1. The *experience* parameter indicates how often the classifier is chosen for making a prediction, which represents how general a classifier is.

During the learning cycle, the system receives a series of training data instances. Since UCS is a supervised learner, a training data instance contains also the target class. First, UCS finds a match set $[M]$ with all classifiers in the population whose condition matches the incoming instance. A *correct set* $[C]$ is then formed, containing those classifiers in $[M]$ that have the same action as the input. If $[C]$ is empty, *covering* is activated, wherein a classifier that matches the input is created and assigned the same class 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 according to whether they belong to $[C]$ or not, reflecting the system’s updated belief after being exposed to new observations.

The fitness of a classifier is based on accuracy, which is computed as the number of times a classifier correctly predicted the class divided by the number of times the classifier matched an incoming data instance:

$$acc = \frac{N_{correct}}{N_{matches}}$$

(6)

The fitness is computed as a function of accuracy:

$$F = (acc)^v$$

(7)

where $v$ is a predefined constant.

GA is invoked in $[C]$ if the average time since the last application of GA to classifiers in $[C]$ exceeds a user-defined threshold. If GA is activated, two parents are selected from $[C]$ with a 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 reaches a predefined limit, some classifiers are removed by voting within the population.

In the exploitation phase, an input instance is given with unknown class. After forming the match set $[M]$, UCS proposes a class based on voting among the classifiers in $[M]$; the votes are weighted by the fitness of each classifier in the match set.

### D. Feed-Forward Artificial Neural Networks

Feed-Forward artificial neural networks (ANNs) are popular in machine learning because of their versatility for solving a wide range of forecasting, classification and pattern recognition problems. Haykin [21] presented several powerful properties and characteristics of neural networks including their ability to learn nonlinear decision boundaries, ability to adapt, being robust, and being a compact representation of the target model.

An ANN 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 in the hidden layers. 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 neuron employs a sigmoid function. The architecture of an ANN is shown in Figure 4.

![Fig. 4. A MLP, with 3 input nodes, 3 output nodes and a single hidden layer with 2 hidden nodes.](image)

Training an ANN occurs by tuning the network weights to minimize the output error of the network. The back-propagation algorithm is popular in training ANN and showed success in many applications such as handwriting character recognition [28].

### III. A Framework for Competitive Classification Environments

Figure 5 depicts the proposed framework. As a pre-gate, we use a self-organized map (SOM) neural network to decompose the problem. A classification algorithm is employed to handle each subproblem independently. We experiment with two different types of classification system: UCS and ANN. We call the combined system Self-Organized UCS (SOUCS) or Self-Organized ANN (SOANN). Decomposition and learning occur hand-in-hand; thus the system decomposes the problem as it learns the class boundaries.

The SOM, the pre-gate in the proposed framework, can be seen in two different ways. First, it decomposes the problem by clustering the data (note that the class label is not presented to the SOM). Second, the SOM can be seen as a router in a distributed environment, which assigns the data as they arrive to a node that will take care of classifying the incoming...
instance. Given any data instance, there is a single UCS responsible for it.

Before starting the framework, the number of nodes in SOM needs to be determined. For example, one can decide to use a $2 \times 2$ or a $3 \times 3$ SOM. Each node in the SOM is associated with a complete classifier. Thus there are four independent classifiers for the $2 \times 2$ SOM, and nine independent classifiers for the $3 \times 3$ SOM.

The system is trained incrementally in the training phase. For each training instance, the winning node is first identified based on the similarity function. The weights of the winning node are updated, based on proximity to the instance. The instance is then fed into the LCS coupled with the winning node. A complete training process is carried out at that node. In the case of UCS, covering might occur if the match set is empty, parameters of classifiers in the match set are updated, and GA might be activated. In the case of ANN, the network weights are updated. With each training instance, both the SOM and its classifiers are updated on the fly.

In the testing phase and given a testing instance, SOM first chooses the winning node. The instance is then routed to the LCS coupled with the winning node to make a prediction. The testing phase does not involve any learning in either the SOM or the LCSs.

It is well-known [11] however, that in distributed environments we can’t simply scale the population size linearly with the number of processors. In other words, a population size of 1000 on a single processor does not map to population sizes of 250 in a 4-processor environment. In the worst case, the population size of each individual node may equal the population size of the single UCS. One may wonder then why we need SOUCS. Apart from the nice features of problem decomposition, discussed in Section II in terms of increased accuracy and better representation of the problem, we present a simple queuing analysis below. The analysis is described for SOUCS, but the idea applies just as well to SOOANN.

Assume that the traditional UCS is able to process $T$ instances per second. Let $R$ instances per second denote the rate of data arrival. Assume a single pass-learner (for stream data mining) as presented in [13]. We now need to compare between the single population approach and the SOUCS environment. In this analysis, we will assume that the inter-arrival time is uniform for simplicity. In the single population approach, if $R > T$, queuing theory tells us that we need an infinite queue to accumulate the data waiting for processing. In a multi-server environment, the condition for handling this data set without the need for any queues is that the router can keep up with the arrival rate, and each node can keep up with its share of the arriving instances. This condition is $R < \min(U, M \times T)$, where $M$ is the number of UCS nodes in SOUCS and $U$ is the number of instances a router can route per second (in SOUCS, the router is a SOM) to distribute the incoming data to one of the $M$ UCS populations.

We assume here that all UCS nodes have the same population size, and therefore the processing time of an instance across the nodes is homogeneous. We also assume that the instances are routed evenly by the SOM to the different nodes. In this case the speed-up in SOUCS is essentially proportional to $M$. If these assumptions do not hold, the mathematics is more complex and the speed-up is smaller. The point still holds that the parallelism in SOUCS increases the throughput of the system.

IV. EXPERIMENTAL DESIGN

A. Testing Problems

To understand the behavior of the proposed framework, we conduct three sets of experiments: one using synthetic problems, one using a number of small and medium size datasets, and a third using a large dataset.

1) Synthetic Datasets: We designed five artificial problems, with different complexity in terms of class boundaries and class distribution. All problems have 2 attributes with boxing constraints between 0 and 4. In each problem, 8000 data points are sampled using a uniform random number generator.

The problems are called “circle 1”, “circle 2”, “circle 3”, “circle 4”, “cross circle” as shown in Figure 6. The idea is to start with a simple classification problem with a nonlinear class boundary (circle 1) and non-uniform class distribution — noting that the ratio of the positive cases to the negative cases is proportional to the area of the circle(s) to the area of the square. The complexity of the learning problem is then increased incrementally by adding more disjoint areas for the positive class (circles 2–4). In the fifth problem, the class boundary is more complex.

2) Small and Medium Datasets: This set of problems was selected to give us a range of important characteristics, such as (i) number of features: low (up to 10 features) or high (more than 10 features); (ii) different representations: real, integer, nominal, or a mixture; (iii) number of classes: binary (2 possible outcomes), small (3 or 4 outcomes), or large (5 or more outcomes).

Table I shows the properties of the chosen datasets used in this paper: Inst (the number of instances in the dataset), $F$s (the number of features), $R$ (the number of real-valued features), $I$ (the number of integer-valued features), $N$ (the...
Fig. 6. The five synthetic problems: circle 1 (first row, left hand side), circle 2 (first row, right hand side), circle 3 (second row, left hand side), circle 4 (second row, right hand side), cross circle (last row). The ‘+’ represents positive data and ‘.’ represents negative data. Each circle represents the boundary for a positive label area except in cross circle, where the positive label area is the intersection of four circles.

The statistical data is collected after 500 iterations in each problem. The parameter settings for UCS are fixed for all problems and all environments, similar to those used in Wilson [41] and Bernadó-Mansilla et al. [4] as follows: $v = 5, \theta_{GA} = 50, \lambda = 0.8, \mu = 0.04, \theta_{del} = 50, \theta_{sub} = 50$. Four systems are tested in this paper to verify the proposed framework: traditional UCS, SOUCS, traditional ANN and SOANN.

For UCS and SOUCS, we first make the sum of the total population sizes of SOUCS equal (or almost equal) to the population size of a single UCS. For example, if we set the population size as $N = n$ in a single-UCS environment, the population size for each UCS in the SOUCS environment is set to $N = n/M$, where $M$ is the number of UCS environments. For a second comparison, we double the population size for each UCS in the SOUCS environment, while leaving the population size of the single UCS unchanged. There is now some “overhead” in SOUCS, since the total population in all UCSs in the SOUCS environment is more than for the single UCS. Each individual UCS in the framework still has a smaller population than the single UCS, however, so the overall system should still be faster. This gives us an indication of the effect on accuracy if we are prepared to accept the overhead. Finally we double the population size of the single UCS as well, to give a direct comparison with the SOUCS environment.

In the first set of experiments, we analyze UCS and SOUCS on the synthetic datasets. The population size of UCS is set to $N = 1000$. For SOUCS-1, the SOM is of size $2 \times 2$ and the population size for each UCS is set to 250. For SOUCS-2, the SOM is of size $3 \times 3$ and the individual population size of each UCS is set to 111. Next, we double the population size of each individual UCS to $N = 500$ and $N = 222$ for SOUCS-1, SOUCS-2 respectively, while leaving the population size of the traditional UCS unchanged. Finally we increase the population size for the traditional UCS to $N = 2000$.

The second set of experiments is carried out on the UCI data sets. We use a population size of 6400 for the single UCS. For SOUCS-1 and SOUCS-2, the SOM is of size $2 \times 2$ and $3 \times 3$, respectively, and the population size for each UCS is set to 1600 and 711 respectively. Next, we double the population size for each UCS to $N = 3200$ and $N = 1422$ for SOUCS-1, SOUCS-2 respectively, while leaving the population size for the single UCS at 6400. Finally we increase the population size for the traditional UCS to $N = 12800$.

The third set of experiments is tested on the Forest dataset. We use a population size of 6400 for the single UCS. For SOUCS-1, SOUCS-2 and SOUCS-3, the SOM is of size $2 \times 2$, $3 \times 3$ and $4 \times 4$, respectively, and the population size for each UCS is set to 1600, 711 and 400 respectively. Next, we increase the population size for each UCS to $N = 3200$ for all SOUCS.

All results, unless stated otherwise, are the average over 10-fold cross validation. Each run uses different random seeds which are consistent in all experiments. 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 iteration.

### Table I: The properties of testing datasets

<table>
<thead>
<tr>
<th>Problem</th>
<th># Inst</th>
<th># Ps</th>
<th># R</th>
<th># I</th>
<th># N</th>
<th># C</th>
</tr>
</thead>
<tbody>
<tr>
<td>breast-w</td>
<td>699</td>
<td>9</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>bupa</td>
<td>345</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>diabetes</td>
<td>768</td>
<td>8</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>glass</td>
<td>214</td>
<td>9</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>iris</td>
<td>150</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>segment</td>
<td>2310</td>
<td>19</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>tao</td>
<td>1888</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>vehicle</td>
<td>946</td>
<td>17</td>
<td>17</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>wine</td>
<td>178</td>
<td>13</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

Except for the Tao problem, all these problems are available from the UCI data repository [2]. The Tao problem was first proposed in [30] and is used widely in LCS research.

3) Large Dataset: The Forest Cover Type data set of the Roosevelt national forest in northern Colorado, available at [2], is chosen for testing. According to [6], the collected data covers an area of 70 miles northwest of Denver in Colorado ($30m \times 30m$ cells obtained from US Forest Service (USFS) Region 2 Resource Information System (RIS)), which has 7 major forest cover types (or seven classes). The data was obtained from the U.S. Geological Survey.

The data set has 581,012 observations, 54 attributes, and no missing values. Each observation is labelled as one of 7 different classes (forest cover types).

### B. Experimental Setup

To evaluate the proposed framework, we experiment with two structures: $2 \times 2$ SOM and $3 \times 3$ SOM.

The parameter settings for UCS are fixed for all problems and all environments, similar to those used in Wilson [41] and Bernadó-Mansilla et al. [4] as follows: $v = 5, \theta_{GA} = 50, \lambda = 0.8, \mu = 0.04, \theta_{del} = 50, \theta_{sub} = 50$. Four systems are tested in this paper to verify the proposed framework: traditional UCS, SOUCS, traditional ANN and SOANN.

For UCS and SOUCS, we first make the sum of the total population sizes of SOUCS equal (or almost equal) to the population size of a single UCS. For example, if we set the population size as $N = n$ in a single-UCS environment, the population size for each UCS in the SOUCS environment is set to $N = n/M$, where $M$ is the number of UCS environments. For a second comparison, we double the population size for each UCS in the SOUCS environment, while leaving the population size of the single UCS unchanged. There is now some “overhead” in SOUCS, since the total population in all UCSs in the SOUCS environment is more than for the single UCS. Each individual UCS in the framework still has a smaller population than the single UCS, however, so the overall system should still be faster. This gives us an indication of the effect on accuracy if we are prepared to accept the overhead. Finally we double the population size of the single UCS as well, to give a direct comparison with the SOUCS environment.

In the first set of experiments, we analyze UCS and SOUCS on the synthetic datasets. The population size of UCS is set to $N = 1000$. For SOUCS-1, the SOM is of size $2 \times 2$ and the population size for each UCS is set to 250. For SOUCS-2, the SOM is of size $3 \times 3$ and the individual population size of each UCS is set to 111. Next, we double the population size of each individual UCS to $N = 500$ and $N = 222$ for SOUCS-1, SOUCS-2 respectively, while leaving the population size of the traditional UCS unchanged. Finally we increase the population size for the traditional UCS to $N = 2000$.

The second set of experiments is carried out on the UCI data sets. We use a population size of 6400 for the single UCS. For SOUCS-1 and SOUCS-2, the SOM is of size $2 \times 2$ and $3 \times 3$, respectively, and the population size for each UCS is set to 1600 and 711 respectively. Next, we double the population size for each UCS to $N = 3200$ and $N = 1422$ for SOUCS-1, SOUCS-2 respectively, while leaving the population size for the single UCS at 6400. Finally we increase the population size for the traditional UCS to $N = 12800$.

The third set of experiments is tested on the Forest dataset. We use a population size of 6400 for the single UCS. For SOUCS-1, SOUCS-2 and SOUCS-3, the SOM is of size $2 \times 2$, $3 \times 3$ and $4 \times 4$, respectively, and the population size for each UCS is set to 1600, 711 and 400 respectively. Next, we increase the population size for each UCS to $N = 3200$ for all SOUCS.

All results, unless stated otherwise, are the average over 10-fold cross validation. Each run uses different random seeds which are consistent in all experiments. 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 iteration.
experiment, except for the Forest dataset; due to the large size of this dataset, we collect the statistical data after 5 iterations.

When UCS is replaced with an ANN, we maintain the same number of hidden units in all networks. For the UCI datasets, five hidden nodes are used, while 20 nodes are used for the Forest dataset.

The statistical test of significance (t-test) is used with a significance level of 0.05.

V. AN INVESTIGATION OF THE PREDICTIVE ACCURACY

A. SOUCS

In this section, we compare the predictive accuracy of UCS and SOUCS using different setups.

Table II presents the mean and the standard deviation (over 10 runs) of the predictive accuracy in different environments on the synthetic data sets.

This table shows that the accuracy of SOUCS-1 is significantly better than that of UCS in two problems and equivalent in three problems. SOUCS-2 performs worse than UCS in one problem and equivalent in the other four problems.

SOUCS-1 performs better than UCS in the circle 3 and circle 4 problems, where the problem complexity increased with the increase in the discontinuity in class boundaries. In the circle 4 problem, however, SOUCS-2 performs significantly worse than a single UCS. We hypothesize that this occurred because of the small population size of each UCS in SOUCS-2.

Therefore, we undertake an extra experiment by increasing the population size of each UCS to 500 individuals for SOUCS-1 and 222 for SOUCS-2. Table III compares the accuracy of SOUCS with a single UCS whose population size is still 1000, and Table IV gives the comparison when the population size for the single UCS is increased to 2000 (called UCS-2).

From Table III and IV, we can see that SOUCS-1 is significantly better than UCS-1 in 3 problems, and better than UCS-2 in 1 problem. Similarly, the accuracy of SOUCS-2 is significantly better than UCS-1 in 3 problems, and better than UCS-2 in 2 problems.

Increasing the population size in each local UCS of SOUCS-1 and SOUCS-2 results in better accuracy. It is worth mentioning that achieving an accuracy of 1 is very hard in this case because UCS uses axis-parallel hyper-planes which will misclassify some data around the non-linear decision boundaries of the circles.

We then test the SOUCS environment on the small and medium size UCI data sets. Table V shows the accuracy of a single UCS and the SOUCS environment.

Once more, the SOUCS environment does well. SOUCS-2 is significantly better on 3 out of 9 problems, comparable results are achieved on 4 problems, and SOUCS-2 performs worse in 2 problems.

SOUCS performs worse than UCS in the segment and vehicle problems, because both problems have a high number of classes (7 classes for segment and 4 classes for vehicle) and a high number of features (19 features for segment and

### Table II

<table>
<thead>
<tr>
<th></th>
<th>UCS</th>
<th>SOUCS-1</th>
<th>SOUCS-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>circle 1</td>
<td>0.981±0.006</td>
<td>0.981±0.004</td>
<td>0.982±0.004</td>
</tr>
<tr>
<td>circle 2</td>
<td>0.96±0.023</td>
<td>0.96±0.009</td>
<td>0.96±0.023</td>
</tr>
<tr>
<td>circle 3</td>
<td>0.94±0.008</td>
<td>0.95±0.012</td>
<td>0.94±0.015</td>
</tr>
<tr>
<td>circle 4</td>
<td>0.94±0.010</td>
<td>0.95±0.007</td>
<td>0.95±0.007</td>
</tr>
<tr>
<td>cross circle</td>
<td>0.94±0.007</td>
<td>0.945±0.019</td>
<td>0.941±0.010</td>
</tr>
</tbody>
</table>

Differences that are statistically significant at a significance level of .05 are denoted by a ◦ if the SOUCS environment is better, and by ○ if the SOUCS environment is worse. N=1000 for each UCI, N=250 for each UCS in SOUCS-1 with 2×2 SOM, N=111 for each UCS in SOUCS-2 with 3×3 SOM.

### Table III

<table>
<thead>
<tr>
<th></th>
<th>UCS</th>
<th>SOUCS-1</th>
<th>SOUCS-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>circle 1</td>
<td>0.981±0.006</td>
<td>0.982±0.012</td>
<td>0.985±0.004</td>
</tr>
<tr>
<td>circle 2</td>
<td>0.96±0.023</td>
<td>0.96±0.011</td>
<td>0.965±0.010</td>
</tr>
<tr>
<td>circle 3</td>
<td>0.94±0.008</td>
<td>0.957±0.005</td>
<td>0.961±0.012</td>
</tr>
<tr>
<td>circle 4</td>
<td>0.94±0.010</td>
<td>0.955±0.007</td>
<td>0.953±0.007</td>
</tr>
<tr>
<td>cross circle</td>
<td>0.94±0.007</td>
<td>0.951±0.005</td>
<td>0.954±0.008</td>
</tr>
</tbody>
</table>

Differences that are statistically significant at a significance level of .05 are denoted by a ◦ if the SOUCS environment is better, and by ○ if the SOUCS environment is worse. N=1000 for each UCS, N=500 for each UCS in SOUCS-1 with 2×2 SOM, N=222 for each UCS in SOUCS-2 with 3×3 SOM.

### Table IV

<table>
<thead>
<tr>
<th></th>
<th>UCS</th>
<th>SOUCS-1</th>
<th>SOUCS-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>circle 1</td>
<td>0.984±0.006</td>
<td>0.982±0.012</td>
<td>0.985±0.004</td>
</tr>
<tr>
<td>circle 2</td>
<td>0.964±0.008</td>
<td>0.96±0.011</td>
<td>0.965±0.010</td>
</tr>
<tr>
<td>circle 3</td>
<td>0.948±0.009</td>
<td>0.957±0.005</td>
<td>0.961±0.012</td>
</tr>
<tr>
<td>circle 4</td>
<td>0.947±0.006</td>
<td>0.955±0.007</td>
<td>0.953±0.007</td>
</tr>
<tr>
<td>cross circle</td>
<td>0.948±0.007</td>
<td>0.951±0.005</td>
<td>0.954±0.008</td>
</tr>
</tbody>
</table>

Differences that are statistically significant at a significance level of .05 are denoted by a ◦ if the SOUCS environment is better, and by ○ if the SOUCS environment is worse. N=2000 for the single UCS, N=500 for each UCS in SOUCS-1 with 2×2 SOM, N=222 for each UCS in SOUCS-2 with 3×3 SOM.

### Table V

<table>
<thead>
<tr>
<th></th>
<th>UCS</th>
<th>SOUCS-1</th>
<th>SOUCS-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breast-w</td>
<td>0.96±0.015</td>
<td>0.966±0.012</td>
<td>0.973±0.016</td>
</tr>
<tr>
<td>Bupa</td>
<td>0.681±0.053</td>
<td>0.715±0.034</td>
<td>0.719±0.035</td>
</tr>
<tr>
<td>Diabets</td>
<td>0.732±0.056</td>
<td>0.736±0.031</td>
<td>0.713±0.043</td>
</tr>
<tr>
<td>Glass</td>
<td>0.710±0.054</td>
<td>0.677±0.058</td>
<td>0.682±0.112</td>
</tr>
<tr>
<td>Iris</td>
<td>0.953±0.045</td>
<td>0.960±0.034</td>
<td>0.947±0.042</td>
</tr>
<tr>
<td>Segment</td>
<td>0.962±0.011</td>
<td>0.965±0.011</td>
<td>0.949±0.050</td>
</tr>
<tr>
<td>Tao</td>
<td>0.880±0.012</td>
<td>0.940±0.021</td>
<td>0.956±0.016</td>
</tr>
<tr>
<td>Vehicle</td>
<td>0.715±0.031</td>
<td>0.690±0.040</td>
<td>0.669±0.041</td>
</tr>
<tr>
<td>Wine</td>
<td>0.966±0.047</td>
<td>0.967±0.060</td>
<td>0.949±0.050</td>
</tr>
</tbody>
</table>

Differences that are statistically significant at a significance level of .05 are denoted by a ◦ if the SOUCS environment is better, and by ○ if the SOUCS environment is worse. N=6400 for the single UCS, N=1600 for each UCS in SOUCS-1 with 2×2 SOM, N=711 for each UCS in SOUCS-2 with 3×3 SOM.
N denotes by a if the SOUCS environment is better, and by o if the SOUCS environment is worse. \(N=3200\) for each UCS in SOUCS-1 with \(2 \times 2\) SOM, \(N=1422\) for each UCS in SOUCS-2 with \(3 \times 3\) SOM.

<table>
<thead>
<tr>
<th>Segment</th>
<th>UCS-1</th>
<th>SOUCS-1</th>
<th>SOUCS-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vehicle</td>
<td>0.715±0.031</td>
<td>0.714±0.034</td>
<td>0.697±0.030</td>
</tr>
</tbody>
</table>

Table VI compares the accuracy of UCS and SOUCS when the population size in each local UCS in SOUCS is doubled to \(N=12800\) (called UCS-2). In the Segment problem, both SOUCS-1 and SOUCS-2 achieve better accuracy than UCS-1 and comparable accuracy with UCS-2. In the Vehicle problem, SOUCS-1 and SOUCS-2 are able to achieve similar accuracy as both UCS-1 and UCS-2.

In conclusion, SOUCS is promising in terms of its ability to generalize better if the population size and the number of cells in the SOM are chosen carefully.

We then challenge SOUCS with a large problem dataset; that is, the Forest Cover Type dataset. Table VIII shows the accuracy with a single UCS and the SOUCS environment. We can see that every SOUCS environment performs worse than the single UCS, especially with the \(4 \times 4\) SOM.

<table>
<thead>
<tr>
<th>UCS</th>
<th>SOUCS-1</th>
<th>SOUCS-2</th>
<th>SOUCS-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.729±0.005</td>
<td>0.719±0.006 o</td>
<td>0.711±0.004 o</td>
<td>0.694±0.006 o</td>
</tr>
</tbody>
</table>

Table VIII shows that SOUCS performs better than the single UCS, especially with the \(4 \times 4\) SOM. We can see that every SOUCS environment performs worse than the single UCS, especially with the \(4 \times 4\) SOM.

Similar to SOUCS, SOANN-1 uses a \(2 \times 2\) SOM, while SOANN-2 uses a \(3 \times 3\) SOM. From Table X we can see that the SOANN environment performs significantly better on 3 out of 9 problems, equivalently on 5 problems, and worse on only one problem. This experiment confirms that using a SOM as a pre-gate can possibly be generalized to other classification methods easily.

B. Artificial Neural Network

Similar to SOUCS, we test SOANN on the Forest Cover Type dataset. As can be seen from Table XI SOANN-1 performs better than ANN.

Table XII shows the performance obtained on the forest data set in this paper, and the performance obtained with other evolutionary and non-evolutionary approaches in other papers (standard deviation is not tabulated because it was not available in all of the papers we compare against). The accuracy we obtained appears to be better than three of the other four papers. However, these results should be seen as indicative only, because it was not possible to unify the experiment setup.

\[\text{Table VI} \quad \text{The mean and deviation of the accuracy of UCS and SOUCS on UCI datasets} \]

\[\begin{array}{|c|c|c|c|}
\hline
\text{UCS-1} & \text{SOUCS-1} & \text{SOUCS-2} \\
\hline
\text{Segment} & 0.971±0.009 o & 0.970±0.011 o \\
\text{Vehicle} & 0.715±0.031 & 0.714±0.034 & 0.697±0.030 \\
\hline
\end{array} \]

\[\text{Table VII} \quad \text{The mean and deviation of the accuracy of UCS and SOUCS on UCI datasets} \]

\[\begin{array}{|c|c|c|c|}
\hline
\text{UCS-2} & \text{SOUCS-1} & \text{SOUCS-2} \\
\hline
\text{Segment} & 0.968±0.014 & 0.971±0.009 & 0.970±0.011 \\
\text{Vehicle} & 0.719±0.021 & 0.714±0.034 & 0.697±0.030 \\
\hline
\end{array} \]

\[\text{Table IX} \quad \text{The mean and deviation of the accuracy of UCS and SOUCS on Forest Cover Type dataset} \]

\[\begin{array}{|c|c|c|c|}
\hline
\text{UCS} & \text{SOUCS-1} & \text{SOUCS-2} & \text{SOUCS-3} \\
\hline
\text{ANN} & 0.966±0.020 & 0.964±0.018 & 0.973±0.017 o \\
\text{SOANN-1} & 0.736±0.053 & 0.701±0.038 o & 0.681±0.054 o \\
\text{SOANN-2} & 0.768±0.050 & 0.764±0.055 & 0.745±0.062 \\
\text{glass} & 0.658±0.083 & 0.650±0.095 & 0.623±0.084 \\
\text{iris} & 0.967±0.047 & 0.960±0.047 & 0.953±0.045 \\
\text{segment} & 0.948±0.015 & 0.958±0.010 o & 0.955±0.015 o \\
\text{taro} & 0.918±0.008 & 0.915±0.010 & 0.940±0.015 o \\
\text{vehicle} & 0.785±0.051 & 0.807±0.046 & 0.786±0.048 \\
\text{wine} & 0.983±0.027 & 0.978±0.029 & 0.977±0.029 o \\
\hline
\end{array} \]

\[\text{Table X} \quad \text{The mean and deviation of the accuracy of ANN and SOANN environments on UCI datasets} \]

\[\begin{array}{|c|c|c|}
\hline
\text{Type dataset} & \text{ANN} & \text{SOANN-1} & \text{SOANN-2} \\
\hline
\text{breast-w} & 0.966±0.020 & 0.964±0.018 & 0.973±0.017 o \\
\text{bupa} & 0.736±0.053 & 0.701±0.038 o & 0.681±0.054 o \\
\text{diabetes} & 0.768±0.050 & 0.764±0.055 & 0.745±0.062 \\
\text{glass} & 0.658±0.083 & 0.650±0.095 & 0.623±0.084 \\
\text{iris} & 0.967±0.047 & 0.960±0.047 & 0.953±0.045 \\
\text{segment} & 0.948±0.015 & 0.958±0.010 o & 0.955±0.015 o \\
\text{taro} & 0.918±0.008 & 0.915±0.010 & 0.940±0.015 o \\
\text{vehicle} & 0.785±0.051 & 0.807±0.046 & 0.786±0.048 \\
\text{wine} & 0.983±0.027 & 0.978±0.029 & 0.977±0.029 o \\
\hline
\end{array} \]
TABLE XI

<table>
<thead>
<tr>
<th></th>
<th>ANN</th>
<th>SOANN-1</th>
<th>SOANN-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>0.739±0.004</td>
<td>0.742±0.005</td>
<td>0.735±0.006</td>
</tr>
<tr>
<td>Dev</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Differences that are statistically significant at a significance level of .05 are denoted by • if the SOANN environment is better, and by ◦ if the SOANN environment is worse.*

TABLE XII

<table>
<thead>
<tr>
<th>Approach</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>BagCGPC [17]</td>
<td>0.619</td>
</tr>
<tr>
<td>BoostCGPC [18]</td>
<td>0.664</td>
</tr>
<tr>
<td>Neural Network Ensemble [36]</td>
<td>0.771</td>
</tr>
<tr>
<td>ANNCAD [27]</td>
<td>0.710</td>
</tr>
<tr>
<td>UCS (N=6400)</td>
<td>0.729</td>
</tr>
<tr>
<td>SOUCS-1 (N=3200)</td>
<td>0.728</td>
</tr>
<tr>
<td>SOUCS-2 (N=3200)</td>
<td>0.725</td>
</tr>
<tr>
<td>SOUCS-3 (N=3200)</td>
<td>0.724</td>
</tr>
<tr>
<td>ANN</td>
<td>0.779</td>
</tr>
<tr>
<td>SOANN-1</td>
<td>0.742</td>
</tr>
<tr>
<td>SOANN-2</td>
<td>0.735</td>
</tr>
</tbody>
</table>

for all methods. The neural ensemble approach presented in [36] achieved higher accuracy, using 20 neural networks each of which had 40 hidden nodes in the hidden layer. This setup implies a huge training cost.

C. Comparing UCS and ANN

Tables V and IX show the accuracy with UCS/SOUCS environments on the real data sets. Tables X and XI show the accuracy with ANN/SOANN environments on the same data sets.

Comparing the results, ANN/SOANN do better than UCS/SOUCS on the data sets that are a challenge for UCS due to the large number of classes or features. Otherwise there is generally either no difference in accuracy, or else the results based on UCS are better.

In other words, the type of LCS that performs better depends on the characteristics of the problem. This applies generally in LCS research.

D. Summary

The results show that the proposed framework has good potential. In some cases, SOUCS/SOANN were able to achieve better accuracy than UCS/ANN, but most of the time they are able to get equivalent accuracy.

The results also showed that initial parameters’ values — in particular, the population size — are important as they can improve or worsen the performance. This is also a problem in traditional UCS.

VI. An Investigation of the Execution Speed

In terms of computational time, it is obvious that SOUCS is much faster. An instance that gets passed from the SOM to one of the UCS would only need to scan little more than one quarter (for SOUCS-1) or one ninth (for SOUCS-2) of the population size of the single UCS. This is a significant reduction in processing time because LCSs spend most of the processing time scanning the population to form the match set.

Figure 7 shows the number of macro classifiers generated in the circle problems. We can see that the number of macro classifiers for both SOUCS environments is less than the corresponding number in the single UCS environment.

Fig. 7. The population size of UCS, SOUCS-1 and SOUCS-2 on the circle 4 problem.

To show the saving in computational time resultant from using SOUCS, we plot the runtime of each system (measured by CPU clock) in Figure 8. The graph confirms that the SOUCS environment uses less computational time than the single UCS, and the SOUCS-2 environment uses still less.

Fig. 8. The computational time of UCS, SOUCS-1 and SOUCS-2 on the circle 4 problem.

Figures 9 and 10 show the number of macro classifiers and the computational time of the UCS and SOUCS environments on the UCI problems. Once more, the SOUCS environments use less computational time than the single-UCS environment. It is important to notice the higher number of rules evolved by UCS after training.
VII. Search Space Decomposition

In this section, we first analyze problem decomposition using a SOM. Figures 11 and 12 illustrate the dynamics of the SOM over time for the circle 4 problem. We use a small box to denote the state of the weight vector after each pass of the data, and a heavily-outlined box to denote the final steady state. The trajectories show the self-organization as it happens for each cell. It is important to remind the reader that the SOM does not see the label for each instance; thus it works on the density of the data in the feature space. Since the data was generated uniformly in this 2-D space, it is logical to see that the steady state for the SOM is simply reached by dividing the space into four and nine uniform quadrants in the case of $2 \times 2$ and $3 \times 3$ SOM respectively.

Figures 13 and 14 show the decomposition of SOUCS as a whole in the last generation. The solid lines show the division of the data using the SOM, the dashed circles show the approximate decision boundaries generated by the individual UCS, and the class labels are shown as reported by the individual UCS. It is interesting to see that in some cases the classification problem in a cluster can be as trivial as everything is positive or negative, while the classification problem in other clusters can have the same complexity as the original problem.

To analyze the decomposition in the case of the UCI datasets, we use the visualization technique developed in [35]. This technique projects the high-dimensional feature space onto lower dimensional principal components space, then uses a convex-hull approximation to visualize the boundaries of the clusters. We only visualize the first 3 principal components by visualizing each combination of the 2D graphs.

Figures 15 and 16 show two example plots for the diabetes and breast-w problems in $2 \times 2$ and $3 \times 3$ SOUCS environments respectively. The convex hull boundary shows the different
clusters, while the dots and crosses show the negative and positive labels respectively. It is worth mentioning that as the clustering is occurring in the original feature space, some clusters may become overlapping in the principal component space. We can conclude that a SOM is able to decompose the problem, as useful patterns can be observed through the visualization techniques.

VIII. CONCLUSION AND FUTURE WORK

In this paper, we introduced a new self-organized adaptive rule-induction system. The framework uses a self-organized map artificial neural network as a pre-gate to decompose a problem into several simpler subproblems. Several classifiers are also employed to learn these subproblems.

The proposed architecture was tested on several synthetic
and real-world datasets. It is shown that the accuracy of the proposed framework is better or equivalent to traditional ANN/UCS if the population size (in the case of UCS) is chosen carefully.

There are two important factors of SOUCS: the number of nodes in the SOM, and the population size in each UCS. In some data sets, we cannot set the aggregate population size for all local UCSs in SOUCS equivalent to the population size of a single UCS, because the sub-problems may have similar complexity as the overall problem.

This suggests a possible future direction, where we will look into how to adapt the population size for each local UCS independently. Also, it will be interesting to come up with a mechanism to dynamically grow the SOM — maybe using the Growing Hierarchical SOM to automatically adjust the size of the SOM and make the interaction between the pre-gate and the classifiers more useful. At some point there is a limit to the size of the SOM that makes sense, given the amount of data — it would be interesting to investigate the relationship between the number of available data instances and possible sizes for the SOM.

REFERENCES


Hai H. Dam Hai Huong (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 A. Abbass Hussein Abbass (SM’05) 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 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, an Adjunct Professor with Cairo University, 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 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 numerous conferences in the field including a PC co-chair for CEC’07, SEAL 06, and IEEE-Alife’07.

Chris Lokan 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. Chris is a member of the ACM, the Computer Society of the IEEE, and the Australian Software metrics Association.

Pornthep Rojanavasu Pornthep Rojanavasu received the B.Eng in computer engineering from Chiang mai University and M.Eng in computer engineering from King Mongkut’s Institute of Technology Ladkrabang in 1999 and 2004, respectively. He is currently working toward the Ph.D. degree in electrical engineering in King Mongkut’s Institute of Technology Ladkrabang, Bangkok, THAILAND. His research interests are large scale data mining, distributed data mining, problem decomposition, learning classifier system and neural network.

Ouen Pinngern Ouen Pinngern received the M.S. degree from Oregon State University in USA in 1983 and the Ph.D. degree from University of Nebraska at Lincoln in USA in 1986 both in computer science. He is an associated professor in the Department of Computer Engineering, Faculty of Engineering, King Mongkut’s Institute of Technology Ladkrabang, Bangkok, THAILAND. His current research interests lie in the areas of soft computing, machine learning, data mining, information retrieval and model of computation.