The Performance of the DXCS System on Continuous-Valued Inputs in Stationary and Dynamic Environments

Hai H. Dam
Artificial Life and Adaptive Robotics Laboratory
School of ITEE
Univesity of New South Wales
ADFA, Canberra ACT 2600, Australia
Email: z3140959@itee.adfa.edu.au

Hussein A. Abbass
ARC Centre on Complex Systems
Artificial Life and Adaptive Robotics Laboratory, School of ITEE
Univesity of New South Wales
ADFA, Canberra ACT 2600, Australia
Email: abbass@itee.adfa.edu.au

Chris Lokan
Artificial Life and Adaptive Robotics Laboratory
School of ITEE
Univesity of New South Wales
ADFA, Canberra ACT 2600, Australia
Email: cjl@itee.adfa.edu.au

Abstract- XCS is widely accepted as one of the most reliable Michigan-style Learning Classifier System for data mining. Many studies found that XCS is able to provide good generalization using a ternary representation for binary inputs as well as interval representation for continuous-valued inputs. Since distributed data mining is becoming more popular due to massive data sets spread across a network at many organizations, we have proposed an XCS system for distributed data mining called DXCS. DXCS has been tested on binary inputs. The results showed that DXCS does not only achieve as good performance as the centralized XCS system, but also reduces data transmission in the network. In this paper, we further examine DXCS with real-valued inputs in stationary and dynamic environments.

1 Introduction

Nowadays, most databases in large companies are distributed physically. For example, a company might have several branches and each branch maintains its own complete database system. The ability of collecting and storing electronic data increases rapidly, which generates massive datasets at each location. Many organizations found that it is impossible to transfer all local databases into the central site for centralized data mining due to security issues, limited network bandwidth, or even because of the internal policies [15]. Distributed Data Mining (DDM) is then used for discovering important hidden patterns in the whole organization without shipping all raw data to the main center.

Traditional machine learning techniques such as C4.5, ID3, CART, etc. are applied in DDM and achieve fair performance [11, 14]. However, most of these techniques require batch learning; that is, a learning model needs to be trained in advance. In a dynamic environment, changes in the environment would occur frequently and learning models are required to be re-trained frequently in order to adapt to the new conditions. If environment conditions change smoothly, it is difficult to decide when to re-learn the model. Hence, an online machine learning technique like XCS [18, 19] is preferable in a dynamic environment.

XCS is widely accepted as one of the most reliable Michigan-style Learning Classifier System (LCS) for data mining [3, 8]. It was introduced by Wilson [18, 19] as an enhanced version of the traditional LCS proposed by Holland [13]. The two major changes of XCS are: a fitness based on the accuracy of the reward prediction instead of the strength (or reward directly received from the environment); and a niche Genetic Algorithm. Many studies showed that XCS performs at least as well as other traditional machine learning techniques on several data mining problems [2, 16].

The advantages of XCS can be counted as being a rule-based representation, which makes it easier to recognize the mined patterns; and being an online-learner with linear O(n) complexity. Online learning is important in DDM in order to handle stream data. Since XCS is an online learner by nature, we decided to base our distributed learning system on XCS. We proposed DXCS, an XCS for DDM [7]. Our experiments showed that DXCS performs as well as a centralized XCS system on the 6-multiplexer problem, and also helps to reduce the traffic load by sending local learning models instead of raw data.

The previous investigation of DXCS used the ternary representation for binary inputs, which is unnatural in real-world data. In this paper, we will investigate DXCS with real valued data using the interval representation.

Since changes occur over time in the real world, we also look at how DXCS copes with dynamic environments. Branke [4] states that a system should be capable of continuously adapting the solution to a changing environment and reusing the information gained in the past. If the problem changes completely without any reference to the history it would be regarded as a simple dynamic environment because the system has to learn from scratch. Changes can be divided into two main categories: changes in the underlying model and changes in the sampling [1]. In this paper, we focus on changes in the underlying model.

This paper concentrates on investigating DXCS, with several representations for continuous-valued inputs in both stationary and dynamic environments, in terms of system performance and data transmission.

The 6-real-multiplexer and checkerboard problems have been used in the literature to test XCS with continuous-valued inputs. We therefore choose them to test DXCS with real-valued inputs in the stationary and dynamic environments.

The remainder of this paper is structured as follows. In the next section, we will present a brief review of DXCS. Several existing representations of XCS for continuous-valued inputs are discussed in Section 3. Section 4 describes our experimental setup and Section 5 shows the results. Finally, Section 6 concludes this study and provides some fu-

0-7803-9363-5/05/$20.00 ©2005 IEEE.

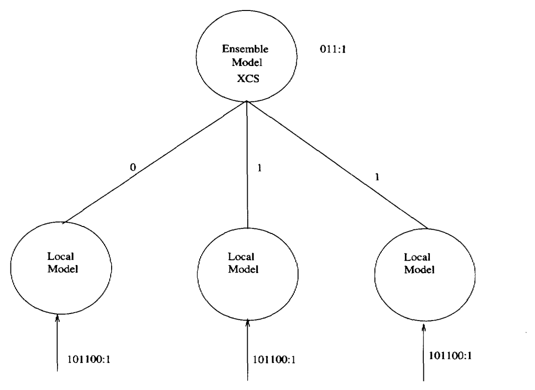


Figure 1: Knowledge Combination at the Server

ture directions of the research.

2 Review of DXCS

In this section we will describe how the DXCS system works. The system consists of a number of clients and a server.

2.1 Clients

Each client has a complete XCS that is trained independently using only the local data. The population of XCS starts from an empty set and keeps evolving during the training. A training instance is input into the local XCS, which will choose a suitable output. The output from XCS is then compared against the real class of that input instance. If the system predicts correctly, the training instance is discarded; otherwise, it is kept in a memory so that it gets transferred to the server at a later stage. The client's models are then sent to the server by the end of each training window. For training at the server, each client sends also a small sample of its local data in addition to its own misclassified instances.

2.2 Central Site (Server)

The server holds copies of all clients' models, then trains a gate using another XCS. The client's model is replaced completely by a new arrived model from that client. The server combines the misclassified and untrained instances from clients and uses them as a training set.

After receiving the updated models from all clients, the server applies the *knowledge probing* approach [10] to combine the local models. The key idea of this approach is to derive a descriptive model using the output of the local models as in Figure 1. All instances in the training set are used as inputs for all copies of local models available at the server, then the server trains an XCS to learn the mapping between the output of these local models and the target class.

2.3 Data Transmission

The complexity of the data transmitted between the client and the server has been estimated by the MDL principle in [7] for binary numbers. This session will describe how the metric can be extended for real numbers.

The data needed for the transmission between the client and the server includes the model, misclassified samples, and training samples. Therefore, the cost of transmission is equivalent to the number of bits needed to encode the model (theory bits) plus misclassified and training samples (exception bits).

$$MDL = \text{theory bits} + \text{exception bits} \tag{1}$$

The theory bits (TL) is the length of those classifiers traveling to the server. The classifiers have a common structure: $Condition \longrightarrow Action$ and therefore their lengths are defined as follows:

$$TL = \sum_{i=1}^{nr}(TL_i) + \sum_{i=1}^{nr}(CL_i) \tag{2}$$

where nr is the number of classifiers of the model; TL_i and CL_i are the length of condition and class respectively. They can be estimated as follows:

$$TL_i = nb \tag{3}$$

$$CL_i = log_2(nc) \tag{4}$$

where nb is the number of bits required to encode a complete set of conditions and nc is the cardinality of the set of possible classes. For example, in the 6-real-multiplexer problem, assuming 16-bit binary is used to encode a real number, $nc = 2$ (2 classes) while $nb = 6 * 2 * 16$ since we have 6 attributes in the *condition* and each attribute requires 2 real numbers for the interval representation.

$$TL = nr(nb + log_2(nc)) \tag{5}$$

Similarly, the exception part of the MDL principle (EL) is estimated as follows:

$$EL = (nm + nu)(na + log_2(nc)) \tag{6}$$

where nm is the number of misclassified samples, nu is the number of unused training samples at the clients and na is the number of bits required to encode a training example.

Thus, the length of data sent from one client to its server is:

$$MDL = (nr)(nb) + (nm + nu)(na) + (nr + nm + nu)(log_2(nc)) \tag{7}$$

3 Review of Representations

In this section, we provide a quick review on several existing representations for continuous-valued inputs as well as their bias and problems. A full investigation can be found in [6, 17].

Wilson [20] proposed the interval predicate for dealing with continuous numbers. XCS maintains a set of classifiers, which each consists of two components: *Condition* and *Action*. A ternary bit in a *Condition* string is replaced by a half-open interval $[p_i, q_i)$. An interval matches an environment attribute x_i if $p_i \leq x_i < q_i$. Therefore, an environment state is considered to match a classifier only if all environment attributes are matched to the corresponding intervals of that classifier.

Wilson [20] introduces a Center-Spread Representation (CSR). An interval is represented in genotype as (c_i, s_i), where $c_i, s_i \in [p_{min}, q_{max})$; p_{min}, q_{max} are the minimum and maximum bounds of the interval; c_i is considered as a center of the interval and s_i specifies a width of the interval from center. The use of this representation requires a truncation when mapping a genotype to phenotype due to the bounded solution space $[p_{min}, q_{max})$. For example, the following genotypes $(0.8, 0.2)$, $(0.9, 0.3)$ are mapped to the same phenotype $(0.6, 1.0)$ if the interval is bounded in $[0, 1)$. Since bounded space is unavoidable, a non-linear mapping from genotype to phenotype always exists which means that many genotypes may be mapped to one phenotype. Therefore, this representation introduces bias to the sample distribution of intervals in phenotype that it generates. Stone at. el. [17] show that 75% of random intervals contain either maximum or minimum bound. Therefore, this representation is biased toward problems in which an interval contains a maximum or minimum bound such as the multiplexer problem.

Wilson [21] proposed a Min-Max Representation (MMR) which overcomes the bias of phenotype sample distribution. The interval predicate is encoded as (l_i, u_i), where l_i, u_i are the lower and upper bounds of the interval. The only condition needs to be hold is: $(l_i \leq u_i)$. This representation provides a linear mapping from genotype to phenotype since no truncation is needed. However, the problem of this representation occurs after the use of Genetic Algorithm operators where interval predicates might be altered. It is possible that an infeasible interval with $l_i > u_i$ is generated. This interval then violates the definition of a genotype in this representation.

Stone and Bull [17] proposed an Unordered-Bound Representation (UBR) to overcome the problem of the MMR. The UBR allows an interval to be presented as $(b1_i, b2_i)$ where either $b1_i$ or $b2_i$ can be the lower or upper bounds. This representation has solved the problem of infeasible intervals but raises another challenge.

Holland's schema theorem and the building block hypothesis [12] explain why Genetic Algorithm is an adaptive and efficient search method [9]. XCS depends heavily on the evolutionary learning process to drive the classifiers of low accuracy to those of high accuracy [5]. However, the UBR can change the semantics of the chromosome by alternating between the lower and upper genes; that is, in one generation the genes representing the lower bound of an interval can become the genes representing the upper bound of an interval in the following generation. This discrepancy will generate a challenge to building blocks.

Dam et. al. [6] proposed a Min-Percentage Representation (MPR) which overcomes the previous problems of the CSR, MMR, and UBR. An interval is presented in the form of (l_i, r_i); where l_i is the lower bound of an interval in phenotype; and r_i is the ratio of a distance between the lower and upper bounds of the interval and a distance between the maximum bound and lower bound of the interval in the phenotype.

We decided to modify DXCS for continuous-valued inputs using the UBR, CSR, and MPR. Not only the representation of XCS at the clients needs to be changed but also its operations such as covering, mutation, crossover, and subsumption need to be modified. The modifications of these operators can be found in [6, 17].

4 Experiments

4.1 Test Problems

The multiplexer problem has been considered as a challenging artificial problem in LCSs. To extend this problem for continuous-valued inputs, Wilson [20] introduces a threshold for converting a real number to a binary number. For example, assume the real numbers are in the range $[0, 1)$ and the threshold is 0.5. The real number r will be converted to 0 in binary if $r < 0.5$, otherwise it will be set to 1 in binary.

Stone and Bull [17] claimed that the real-multiplexer problem is biased because each interval in the classifier contains either the maximum or minimum bound, which is easy for the system to learn. This is not always the case in a real world problem, where an interval might not contain either bound. They introduced an alternative problem called the checkerboard. This problem involves an n-dimensional solution space which is divided into several equal sized hypercubes. Each hypercube is assigned a black or white color with the color alternating in all dimensions. The hypercubes are divided based on n_d, the number of divisions of each dimension of the solution space. For 2 dimensions, the solution space looks like a chess or checker board.

In this paper, we test DXCS on the 6-real-multiplexer problem and checkerboard problem with $n = 3$ and $n_d = 3$.

4.2 Experimental Setup

In order to simulate a dynamic environment in the 6-real-multiplexer problem, we set the threshold to different values after each 20,000 steps. We choose 20,000 time steps to allow the system enough time to adapt to the new environment. Two sets of the threshold values are used. One set produces smooth changes, and the other results in severe changes in the environment. In the first case, the threshold is set to the following values: 0.3, 0.4, 0.5, 0.6, 0.7, 0.8. In the second case, the threshold is set to the following values: 0.5, 0.25, 0.75, 0.5, 0.25, 0.75.

In this paper, we assume that all clients synchronize with each other in terms of environment changes, and data transmission to the server occurs during the same learning steps. A training window of size 50 is used for XCS at the clients.

4.3 System Setup

Previously we tested DXCS with two clients, obtaining good performance. In this study, DXCS is constructed with four clients. XCS at the clients are setup with the same parameter values used by Wilson [20], Stone and Bull [17], and Dam et al [7] as follows: $N = 800, \beta = 0.2, \alpha = 0.1, \epsilon_0 =$

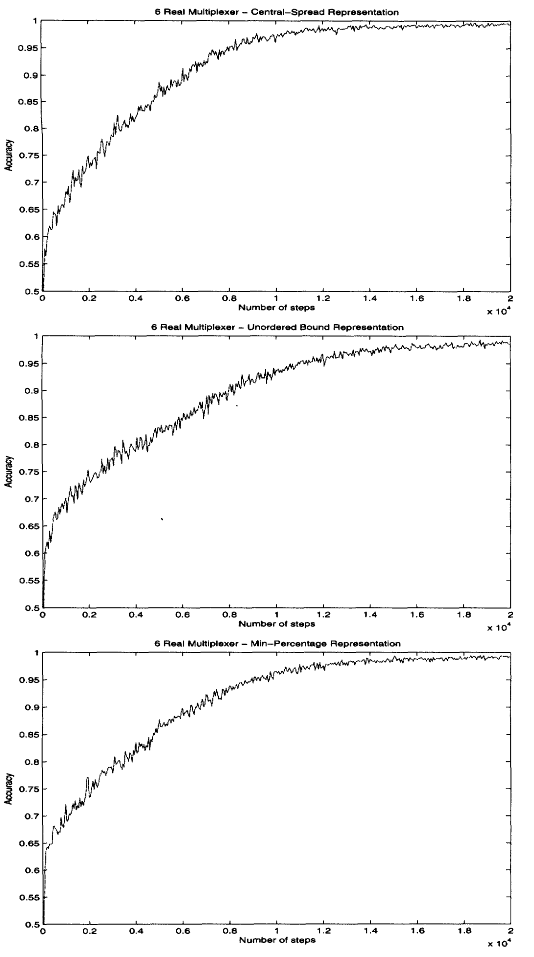


Figure 2: Performance at Clients, 6-real-multiplexer problem

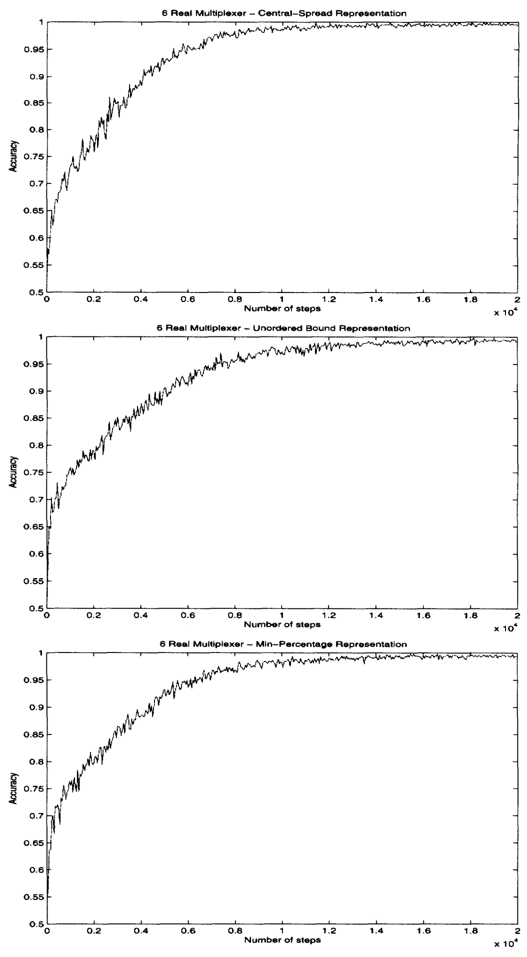


Figure 3: Performance at Server, 6-real-multiplexer problem

$10, v = 5, \theta_{GA} = 12, \chi = 0.8, \mu = 0.04, \theta_{del} = 20, \delta = 0.1, \theta sub = 20, p_I = 10, \varepsilon_I = 0, f_I = 0.01, \theta nma = 2, m = 0.1, s_0 = 1.0$.

The default parameters for XCS at the server are almost similar to those at clients except for the population size $N = 100$ and $P_{\#} = 0.3$ due to the smaller size of the search space.

All experiments presented are averaged over 30 independent runs. The random number generators for all representations are synchronized so that they all deal with the same data.

5 Results

5.1 Static environment with different interval representation

In this section, we investigate DXCS for continuous-valued inputs using several existing representations in the stationary environment. Figure 2 shows performance of the clients of DXCS on 6-real-multiplexer problem. The result shows that the MPR and CSR converge a little bit faster than the UBR in the beginning, then the UBR catches up and all representations achieve equivalent performance at the end. Figure 4 reveals that the MPR produces less macro-classifiers

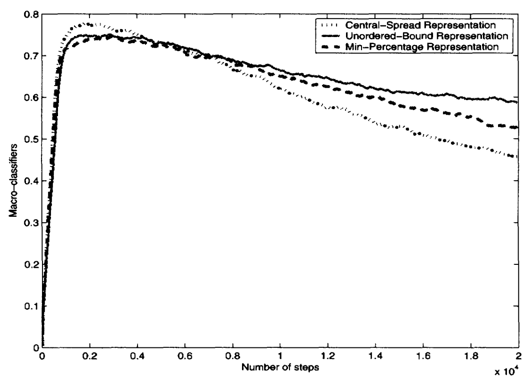


Figure 4: Number of macro-classifiers of XCS at Clients, 6-real-multiplexer problem

than the UBR. Also, CSR produces less macro-classifiers than both MPR and UBR.

Figure 3 shows the system performance at the server. The results show that all representations perform equally well. The CSP and MPR seem to learn quicker than the UBR at the beginning due to the faster convergence at the clients' models. However, they achieve equivalent performance by the end.

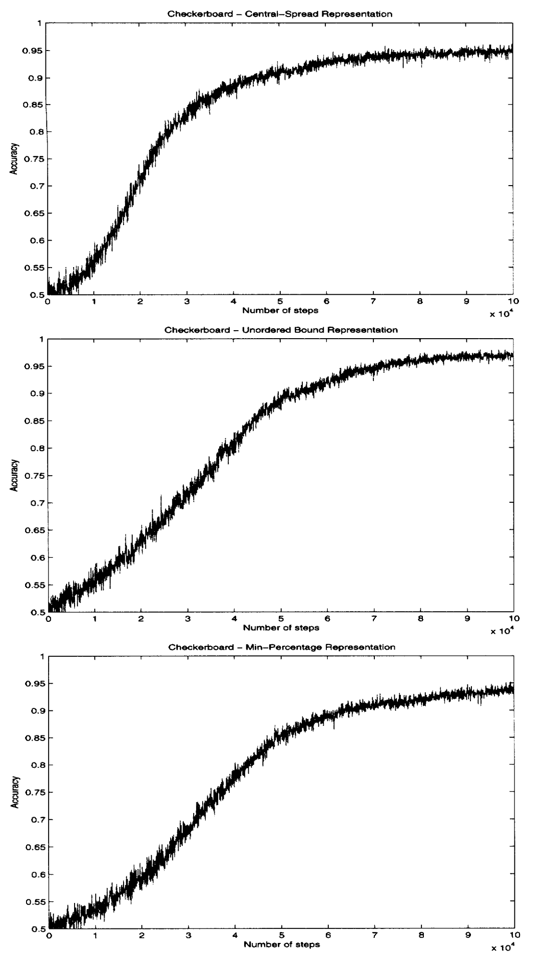


Figure 5: Performance at Clients, checkerboard problem

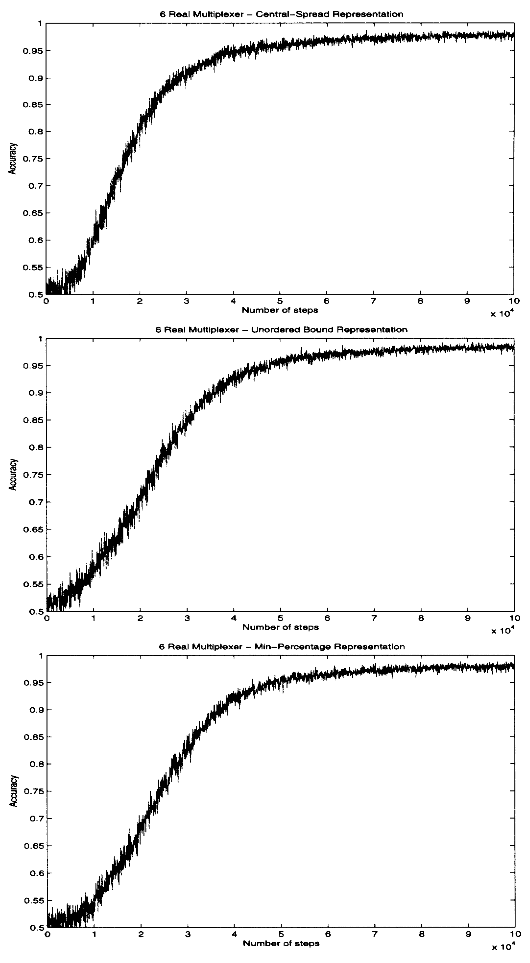


Figure 6: Performance at Server, checkerboard problem

Figure 5 shows the system performance at the clients for the checkerboard problem using the CSR, UBR and MPR respectively. Similar to the 6-real-multiplexer problem, the CSR and MPR converge quicker than the UBR up to 50,000 steps, then all obtain almost equivalent performance at the end. Again, the number of macro-classifiers of the MPR seems to be slightly less than the UBR and the number of macro-classifiers of CSR seems to be lower than the others in Figure 7.

Figure 6 shows the system performance at the server on the checkerboard problem using the CSR, UBR, and MPR. The results show that CSR performs a little bit better than both UBR and MPR at the beginning. But all representations achieve equivalent performance by the end.

In all three representations, the server seems to outperform and converge quicker than the clients, mainly because the server fuses the knowledge learnt at each client. The clients are trained with different training sets and therefore each client might obtain different knowledge over time. Hence, when the server combines knowledge of the local learning models, it seems to minimize the overall bias in the system.

In general, all representations behave similarly in both problems and achieve equivalent performance. Since the MPR avoids several problems of the other representations, we decided to continue testing DXCS using the MPR in the dynamic environment.

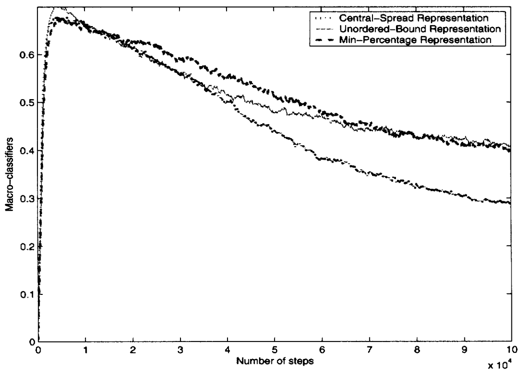


Figure 7: Number of macro-classifiers of XCS at Clients, checkerboard problem

5.2 Smooth change in the environment

Figure 8 shows the client and server performance on the 6-real-multiplexer in the dynamic environment with smooth changes over time. Each point in the graph represents the classification accuracy of 50 time steps. In this experiment, the threshold is changed 6 times as 0.3, 0.4, 0.5, 0.6, 0.7, 0.8. The graphs of both the clients and server display 6 cycles of performance curves, with each cycle corresponds to a threshold value.

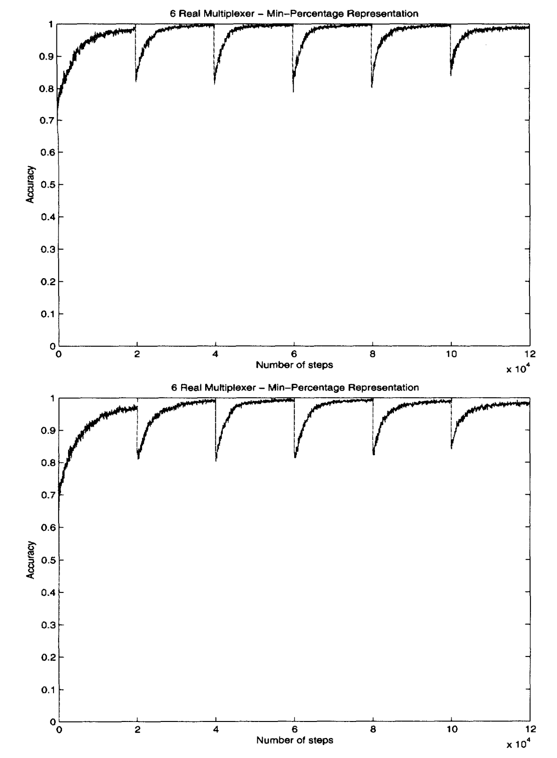


Figure 8: Performance of the server (above) and the clients (below) in smooth dynamic environment, 6-real-multiplexer problem

The server and the clients are synchronized with each other because the server gains its knowledge from the local models obtained at the clients. When the environment changes, the client models of the previous time step become unsuitable. It results in lots of misclassified instances at that time. It requires a period of time for the clients to recover and adapt to the new condition. Since the clients keep sending incorrect learning models to the server, the server's performance suffers when the environment changes. Again, the server seems to converge faster and better than the clients because knowledge from the clients may be obtained differently depending on the training data. One client might learn something which an other client at that time has not learnt.

At time step 20000, the threshold changed from 0.3 to 0.4. Any real number in the range $[0.3, 0.4)$ in the previous time step was considered as 1, but now becomes 0. Since data is generated with a normal distribution, about 10% of the real numbers are converted to the binary number differently in the new environment. There are $50 \times 6 = 300$ real numbers required for 50 training instances (a training window). 10% of them, which is about 30 real numbers, would be converted differently. Therefore, at least 5 instances (6 real numbers form an instance) and at most 30 instances contain the real numbers which are converted differently. At least 20 instances (40%) in the training window are unchanged in the new condition. This explains why the learning curve does not drop as low as the starting point at the time step 0, but drops down to about 80% accuracy. Hence, it shows that DXCS reuses the information gained from the previous cycle.

Moreover, XCS evolves a *complete action map*, which contains all accurate classifiers either low-rewarded (incorrect classification) or high-rewarded (correct classification) [3]. The advantage of the complete action map is that the system does not need to create a new classifier, but only has to update the quality of its fitness to adapt to the changes in the environment. As the result shows, the system would adapt quickly to the environment changes. Thus, the learning curve in all cycles of environment changes after 20,000 time steps, converge faster than the first cycle.

A difference in performance can be noticed between the first cycle (0-20,000 time steps), the last cycle (100,000-120,000 time steps) and other cycles. These cycles could not reach 100% accuracy as other cycles did due to sampling bias made by the threshold. Stone and Bull [17] found that XCS takes approximately 2.5 times longer to solve the real multiplexer with threshold 0.75, than when the threshold is 0.5. This is because the intervals in the range [0, 0.75) are sampled with three times the frequency than intervals of the range $[0.75, 1)$. The thresholds of the first and the last cycle are 0.3 and 0.8 respectively, which then produce a bias in the sampling and therefore require more time to learn than other cycles.

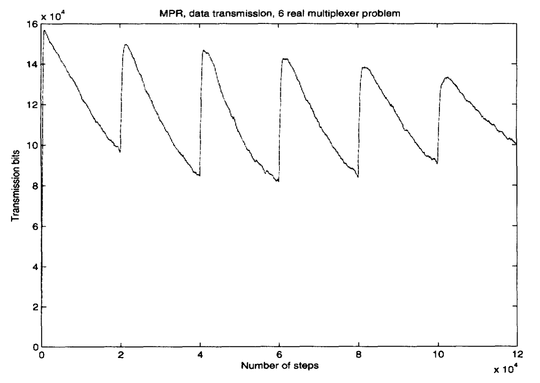


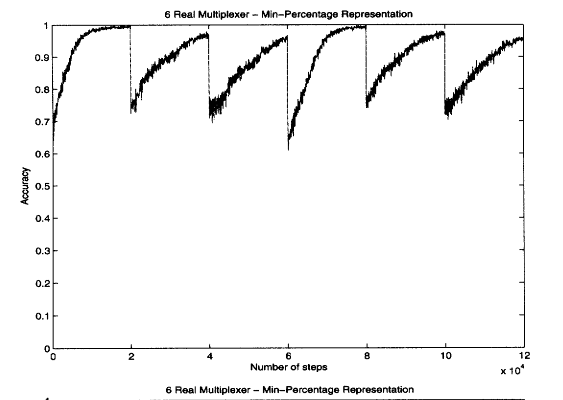
Figure 9: MDL data transmission in smooth dynamic environment, 6-real-multiplexer problem

Figure 9 shows the traffic load from one client to the server for the 6-real-multiplexer problem. The improvement in the client performance results in less data transmission to the server.

At the beginning of each cycle, the size of the client learning models is increased due to the fact that the Genetic Algorithm in XCS introduces more classifiers into the system in order to expand the search space to adapt to the new conditions. The more training an XCS receives, the more compact its model becomes. This is because XCS keeps filtering out irrelevant classifiers, and also applies a subsumption technique in which a correct but less general classifier is subsumed by a correct and more general classifier. Also, at the beginning of the cycle the number of misclassified instances is increased due to an inappropriate learning model of the previous cycle. But as the system improves its knowledge of the new environment conditions, it can classify more accurately and therefore less misclassified instances are sent to the server.

DXCS seems to be capable of adapting quickly to the environment changes by reusing the information gained from the past instead of learning from scratch. Also, the data transmission is reduced in reverse to the system knowledge of the environment.

5.3 Severe change in the environment



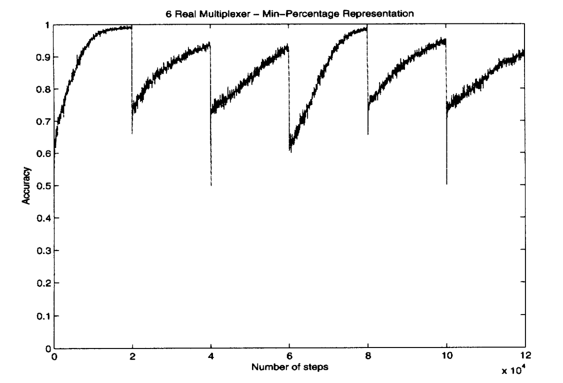


Figure 10: Performance at the server (above) and the client (below) with severe changes in the dynamic environment, 6-real-multiplexer problem

Figure 10 shows the performance of the server and the client on the 6-real-multiplexer problem with severe changes in the environment. In this experiment, the threshold is changed in the order of 0.5, 0.25, 0.75, 0.5, 0.25, 0.75 after each 20,000 time steps.

It is obvious that the server achieves better generalization than the clients in this experiment. The difference in performance between the clients and the server shows clearly at each 20,000 time steps, where the environment starts to change. It is clear that when the threshold is changed rapidly from 0.25 to 0.75 as in 40,000 and 100,000 time steps, the client performance drops quickly as low as 50% accuracy. It indicates that the client starts a random search. However, the random search process does not last long, as XCS is an online learner and contains a complete action map. It seems to take one training window (50 training steps) to increase the performance from 50% to 75%. Performance of the server does not drop as badly as the clients because the client experiences the severe changes in the environment before the server. XCS at the clients continuously adapts to the environment by updating the fitness of active classifiers. The clients send their learning models, which have been adapted partially to the environment changes, to the server after a training window. It results in the better performance at the server when compared to the clients.

The effect of sampling on the performance of the clients and server shows clearly in this experiment. The 0.5 threshold does not produce the sampling bias because the system performance can achieve as good results as 100% accuracy in cycles 0-20,000 and 60,000-80,000. Other cycles, where the threshold is 0.25 or 0.75, achieve up to 95% accuracy and require long time to learn in order to achieve 100% accuracy.

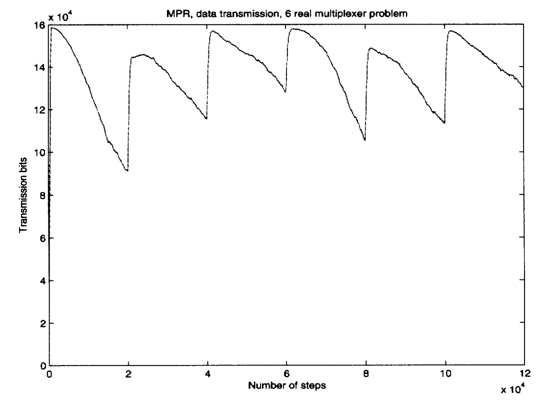


Figure 11: MDL data transmission in severe dynamic environment, 6-real-multiplexer problem

Figure 11 shows the data transmission from the clients to the server over time. Clearly, data transmission in the first and fourth cycles reduce much more than other cycles because the client models in these cycles achieve better generalization than other cycles due to unbias in sampling. As a result, the clients in these cycles become more compact and produce less misclassified instances then other cycles.

6 Conclusion

In this paper, we tested DXCS, an XCS system for distributed data mining, for real-valued inputs in stationary and dynamic environments. We evaluated three existing interval representations in the literature of XCS on DXCS: the Central-Spread representation, the Unordered-Bound representation, and the Min-Percentage representation. Tests on the 6-real-multiplexer and checkerboard problems reveal that the DXCS performs well using these representations in a stationary environments.

We also tested DXCS, using the Min-Percentage representation, in dynamic environments with smooth and severe changes. The results reveal that DXCS is capable of continuous adaptation to the environmental changes and reusing information gained from the past. Also, the data transmission from the clients to the server varies depending on the system performance at the clients.

In the future, we plan to investigate DXCS further with several issues. The first one is how the server performs in heterogeneous and hierarchically distributed environments, which means each client has different underlying problem, the problem is structured in a hierarchy, and the environment of each client is changed differently. The second issue is how to reduce the data transmission by only sending the good XCS models to the server. Also we plan to test DXCS on several data sets from the UCI repository.

Acknowledgement

Work reported in this paper was funded by the Australian Research Council Linkage grant number LP0453657.

Bibliography

[1] H. A. Abbass, J. Bacardit, M. V. Butz, and X. Llora. *Online Adaptation in Learning Classifier Systems: Stream Data Mining*. Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, June 2004. IlliGAL Report No. 2004031.

[2] J. Bacardit and M. V. Butz. *Data Mining in Learning Classifier Systems: Comparing XCS with GAssist*. Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, June 2004. IlliGAL Report No. 2004030.

[3] E. Bernadó, X. Llorà, and J. M. Garrell. 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.

[4] J. Branke. *Evolutionary Optimization in Dynamic Environments*. Kluwer Academic Publishers, 2002.

[5] M. V. Butz, D. E. Goldberg, and K. Tharakunnel. Analysis and improvement of fitness exploitation in XCS: Bounding models, tournament selection, and bilateral accuracy. *Evolutionary Computation*, 11(3):239–277, 2003.

[6] H. H. Dam, H. A. Abbass, and C. Lokan. Be real! XCS with continuous valued inputs. In *Proceedings of Eighth International Workshop on Learning Classifier Systems*, Washington D.C., 2005.

[7] H. H. Dam, H. A. Abbass, and C. Lokan. DXCS: an XCS system for distributed data mining. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2005)*, Washington D.C., USA, 2005.

[8] P. W. Dixon, D. Corne, and M. J. Oates. A preliminary investigation of modified XCS as a generic data mining tool. In *Advances in Learning Classifier Systems: 4th International Workshop, IWLCS*, pages 133–150. Berlin Heidelberg: Springer-Verlag, 2001.

[9] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addision-Wesley Publishing Company, INC., 1989.

[10] Y. Guo, S. Rueger, J. Sutiwaraphun, and J. Forbes-Millott. Meta-learning for parallel data mining. In *Proceedings of the Seventh Parallel Computing Worksop*, 1997.

[11] Y. Guo and J. Sutiwaraphun. Distributed classification with knowledge probing. In H. Kargupta and P. Chan, editors, *Advances in Distributed and Parallel Knowledge Discovery*, pages 115–132. The MIT Press, 2000.

[12] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975. Republished by the MIT press, 1992.

[13] 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.

[14] A. L. Prodromidis, P. K. Chan, and S. J. Stolfo. Meta-learning in distributed data mining systems: Issues and approaches. In H. Kargupta and P. Chan, editors, *Advances in Distributed and Parallel Knowledge Discovery*, pages 81–114. The MIT Press, 2000.

[15] F. Provost. Distributed data mining: Scaling up and beyond. In H. Kargupta and P. Chan, editors, *Advances in Distributed and Parallel Knowledge Discovery*, pages 3–29. The MIT Press, 2000.

[16] S. Saxon and A. Barry. XCS and the Monk's problems. In *Learning Classifier Systems, From Foundations to Applications*, pages 223–242, London, UK, 2000. Springer-Verlag.

[17] C. Stone and L. Bull. For real! XCS with continuous-valued inputs. *Evolutionary Computation*, 11(3):299–336, 2003.

[18] S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.

[19] S. W. Wilson. Generalization in the XCS classifier system. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 665–674, University of Wisconsin, Madison, Wisconsin, USA, 1998. Morgan Kaufmann.

[20] 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.

[21] 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.