Investigating Scaling of an Abstracted LCS Utilising Ternary and S-Expression Alphabets

Charalambos Ioannides
Cybernetics,
University of Reading,
Reading, UK
siu03ci@rdg.ac.uk

Will Browne
Cybernetics,
University of Reading,
Reading, UK
w.n.browne@rdg.ac.uk

ABSTRACT
Utilising the expressive power of S-Expressions in Learning Classifier Systems often prohibitively increases the search space due to increased flexibility over the ternary alphabet. Selection of appropriate S-Expressions functions through domain knowledge improves scaling, as expected. Considering the Cognitive Systems roots, abstraction was included in LCS - episodic learning generalises prior to abstraction for semantic learning. This novel method is shown to provide compact results (135-MUX) and exhibits potential for scaling well (1034-MUX).

Categories and Subject Descriptors
I.2.6 [**Artificial Intelligence**]: Learning

General Terms
Algorithms, Design, Experimentation.

Keywords
Learning Classifier Systems, Abstraction.

1. INTRODUCTION
Although the scalability of Learning Classifier System [1][2] technique is good (70 Multiplexer MUX problem solvable) the increased dimensionality eventually restricts the technique (135 MUX problem is difficult to solve). The search space increase is inherent in the problem and the representative space of the alphabet. One of the difficulties in introducing S-Expressions as an LCS alphabet is that the good building blocks of information are infrequent, especially in large size domains.

Learning Classifier Systems (LCS) were originally designed as Artificial Cognitive Systems [3], but are now established as an effective machine learning technique. LCS's power lies in combining Reinforcement Learning (RL) and Evolutionary Algorithms in order to create and promote a set of production (IF condition THEN action) rules which eventually form the solution to the presented problem. Humans create semantic memory by generalisation of episodes.

GECCO'07, July 7–11, 2007, London, England, United Kingdom.

Generalisation is core to LCS, but it is hypothesized here that the learning of concept-based knowledge related to experiences can be improved by developing abstraction methods.

This work attempts to mimic human behaviour by generating solutions to higher complexity problems from smaller problems through utilising appropriate alphabets to facilitate abstraction. More practically; it is to be investigated if domain-relevant functions will assist learning of multiplexers both in the sense of faster learning and smaller s-expressions.

It is anticipated that tailored functions will 'short-circuit' the problem by reducing dimensionality/search space. However, without domain knowledge the appropriate function(s) for a problem need to be autonomously discovered.

It is to be determined whether it is better to discover the most appropriate functions through generalization (learning directly on the messages) or through abstraction (learning on the generalized rules learnt in a ternary alphabet). Abstraction is thus a higher order process as it involves two stages.

The widely used LCS, XCS (Accuracy-based Fitness LCS) [4], has been well tested on Multiplexer problems with the main limiting factors to the performance being [5][6]:

1. String Length
2. Optimal Population Size [O]
3. Reward Function
4. Minimum Hamming Distance (MHD)

Due to the above issues the XCS is not able to scale well as the problem complexity increases above the 70-MUX problem. The aim of this work is to develop a method based on abstraction so that XCS scales to increasing sized problems of similar type.

2. BACKGROUND
In the classical XCS framework [2] the classifiers that represent the environment use ternary alphabet {0, 1, #}, the assignment of an Action to a classifier is inflexible and uses the # (don't care) symbol in order to present a positional relation acting as the OR function (i.e. '0' OR '1'). In effect this (#) facilitates the important feature of generalisation.

The main characteristics that any single classifier possesses are:

1. Condition part $C - C \in \{0,1,\#\}^l$
2. Action part $A - A \in \{0,1\}$

3. Reward Prediction R
4. Estimated Error ε
5. Fitness F

The Multiplexer problem, see figure 1, is an effective measure for classifier systems due to the properties of multi-modality, scalability and epistasis it possesses. In short, it is because the correct action of a classifier is a function of a function inside its environmental encoding.

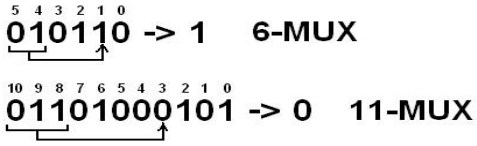

Figure 1. Multiplexer problem

Illustrating this above, the correct output is taken from the data bits according to the index dictated by a set of address bits. If the number of address bits is k=4 the environment's string length is $l=k+2^k = 20$ forming the 20-MUX problem.

3. DESIGN OF THE S-XCS SYSTEM

The XCS framework was developed in this work to enable abstraction, see figure 2.

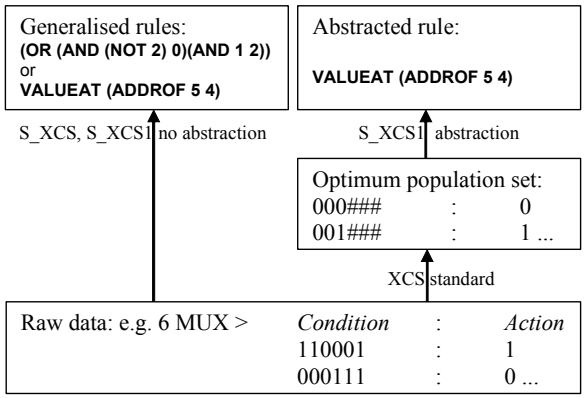

Figure 2. Developed framework for investigating abstraction

The episodes were learnt as in a classical XCS to produce a maximally general accurate population, termed Optimum population set (OPS). This OPS was then the environment where each generalised rule was a message passed to the S-Expression based XCS, termed S-XCS. This could be contrasted with a single population version (without OPS) that directly learnt from the raw environmental messages, again utilising the S-Expression alphabet, in order to discover any benefits of abstraction.

Following the S-Expression paradigm [7][8] the classifiers' building blocks consists of either non-terminals (i.e. functions) or terminals (i.e. variables or constants), see figure 3. These can be tailored to the domain, e.g. VALUEAT which returns the bit value of a position within the classifier and ADDROF which returns the integer counterpart of a string of bits. Tailored functions rely upon domain knowledge, which reduces the applicability of the technique unless used with other general functions.

```
<cond>:= AND <cond> <cond> |
         OR <cond> <cond> |
         NOT <cond> |
         PLUS <cond> <cond> |
         MINUS <cond> <cond> |
         MULTIPLY <cond> <cond> |
         DIVIDE <cond> <cond> |
         POWEROF <cond> |
         VALUEAT <cond> |
         ADDROF <cond> <cond> |
         <var>

<var>:= "0" | "1" | "2" | ... "k"
```

Figure 3. Backus–Naur form grammar that generates classifiers

- AND, OR, NOT – Binary functions; the first two receive two arguments and the last one only one. Arguments can be either binary values directly from the leaf nodes of the condition trees, or integers (values >1 are treated as the binary 'true' or '1') if the functions are located in the middle of the trees.
- PLUS, MINUS, MULTIPLY, DIVIDE, POWEROF – Arithmetic functions; they all receive two arguments apart from the final one that receives only one argument. These arguments can be either binary values or integers as above. The POWEROF function treats its argument as an exponent of base two.
- VALUEAT, ADDROF – Domain specific functions; VALUEAT receives one argument (pointer to an address, which is capped to the string length) and returns a value (environmental 'value' at the referenced address). ADDROF receives two arguments symbolising the beginning and end (as positions in the environmental string) for specifying the binary string to be converted to an integer. There is no ordering as the lowest index is considered the least significant bit and the highest index as the most significant bit.

There is no match set using S-Expressions since the classifiers represent a complete solution by using variables and not predetermined bit strings with fixed actions. Thus the actions are computed (similar to piece-wise linear approximators). It is noted that VALUEAT points to a value and that it is impossible to consistently point to an incorrect value, which does not favour the low prediction classifiers that occur in a classical XCS.

Other methods were also appropriately altered, e.g. Classifier Covering or completely removed, e.g. Subsumption Deletion. Coverage was triggered when the average fitness was below 0.5 in an action set and created a random valid tree. In initial tests numerosity tended to be small due to the diversity of the S-Expressions, which tended to distort fitness measurements. Thus, numerosity was removed and absolute accuracy was used as fitness instead of the relative accuracy.

The update function, including accuracy-based fitness and action selection in exploit trials are the same as in classical XCS.

4. RESULTS

The first system created was the S_XCS which uses only three Binary Functions (AND, OR, NOT). The Covering procedure created one classifier per invocation. The second system was the S_XCS1 which was created in order to investigate the effect of tailored building blocks (e.g. ADDROF, VALUEAT) and to determine if an increased variety of functions causes bloating or prevents learning.

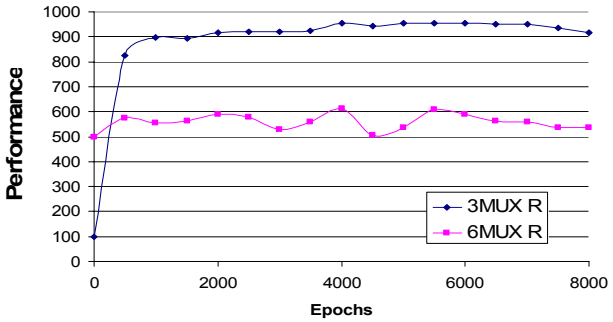

Figure 4. S_XCS on 3-MUX & 6-MUX problem

System parameters are learning rate β=0.2, initial error ε=0.5, accuracy parameters α=0.1, ε_0=10, ν=5, GA threshold θ_{GA}=25, deletion probabilities θ_{del}=20, mutation rate μ=0.01, τ=0.2, crossover possibility χ=1, population size N=400 [2].

Table 1. S_XCS on 3-MUX Optimal Population extract

Condition	Length
(OR (AND (NOT 2) 0) (AND 1 2))	8
(OR (AND (AND (NOT 2) 0) 0) (AND 1 2))	10

S_XCS discovers the Disjunctive Normal Form of the 3-MUX problem (see figure 4 & table 1), but fails to scale. However, S_XCS1 does scale, see figures 5 & 6.

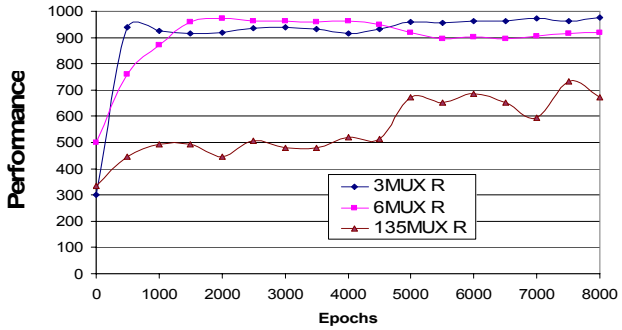

Figure 5. S_XCS1 without OPS on 3-MUX , 6-MUX & 135-MUX problem (8000 epochs)

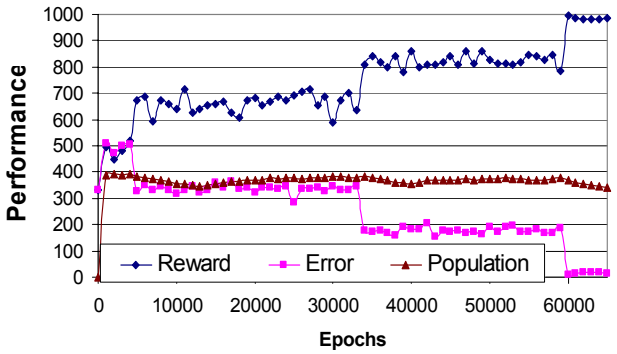

Figure 6. S_XCS1 without OPS on 135-MUX problem (65000 epochs)

Provided sufficient time is allowed the system will discover the most compact functions suited to the domain, see table 2. This time may be greatly reduced by the use abstraction, where generalisation is accomplished using the ternary alphabet (where possible) and then semantic knowledge is gained by training on the optimum set, see figure 7.

Table 2a. S_XCS1 without OPS on 3-MUX. 6-MUX, 135-MUX

MUX	Condition	Length
3	VALUEAT AND 2 2	4
3	**VALUEAT ADDROF 2 2**	**4**
3	VALUEAT ADDROF POWEROF 2 2	5
3	VALUEAT OR POWEROF 2 2	5
6	**VALUEAT ADDROF 5 4**	**4**
6	VALUEAT ADDROF 4 5	4
6	VALUEAT ADDROF 4 POWEROF 3	5
6	VALUEAT ADDROF 4 POWEROF POWEROF 4	6
135	**VALUEAT ADDROF 128 POWEROF 47**	**5**
135	VALUEAT ADDROF POWEROF 7 POWEROF 36	6
135	VALUEAT ADDROF 128 MULTIPLY 7 22	6
135	VALUEAT ADDROF MULTIPLY 27 75 128	6
135	VALUEAT ADDROF MULTIPLY 27 27 128	6

The OPS can not be learnt practically for >135 MUX and thus the known ternary solution was used in figure 7 and table 3, which only shows the potential scalability of S-Expressions. The best discovered solutions are shown in tables 2 and 3 (highest fitness in bold). Unnecessary functions, such as POWEROF, are more likely to appear without abstraction.

The results of using S_XCS1 with a limited set of the most suited functions (discovered though abstraction) did not provide

significant advantage, comparing figure 8 with figure 4. It is only the hypothetical case of figure 7, which demonstrates optimum performance is possible as the domain scales.

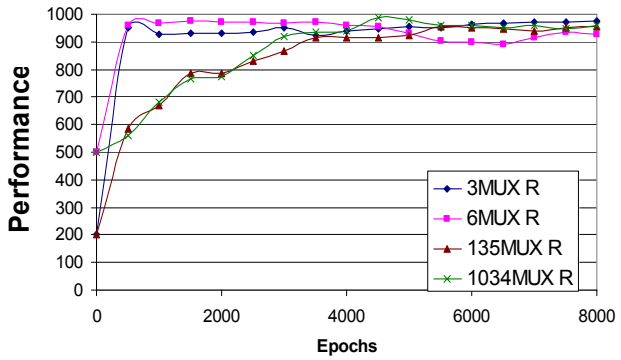

Figure 7. S_XCS1 with OPS on 3-MUX, 6-MUX, 135-MUX & 1034-MUX problem

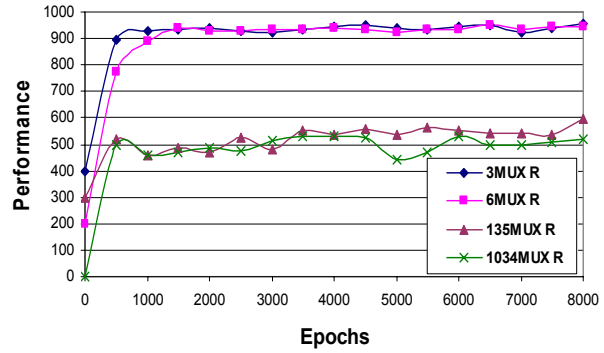

Figure 8. S_XCS1 without OPS on 3-MUX, 6-MUX, 135-MUX & 1034-MUX problem, limited functionality.

5. DISCUSSION

XCS scales well on the MUX problems as the learning is polynomial complexity [9]. However, XCS still struggles to solve the 135-MUX problem whereas humans are capable of understanding the concept behind all MUX problems.

The S_XCS system does exhibit potential in developing solutions that resemble the Disjunctive or the Conjunctive Normal Form of the k-MUX problem, but it did not scale well. The reason for this was that no partially correct classifiers entered the population rendering the Genetic Algorithm redundant and only Covering could create a correct classifier. The problem with competent classifiers not entering the population was that the environmental variables changed at every epoch and the functions (AND, OR, NOT) were too granular to cover the change. Furthermore, the maximum depth of a classifier's condition required limiting (e.g., to 3 in 6-MUX problem) in order to prevent bloat.

The S_XCS1 without abstraction system overcame the S_XCS problems and discovered optimal classifiers, including 135-MUX problem The reason for this behaviour despite ten operators instead of only three – a situation that would normally incur bloating and larger search times – was that the operators were more powerful in their expression and functionality.

Table 3. S_XCS1 with OPS on 3-MUX, 6-MUX, 135-MUX & 1034-MUX

MUX	Condition	Length
3	VALUEAT OR 2 2	4
3	VALUEAT AND 2 2	4
3	**VALUEAT ADDROF 2 2**	**4**
3	VALUEAT AND 2 POWEROF 1	5
3	VALUEAT OR POWEROF 2 2	5
3	VALUEAT OR POWEROF 1 2	5
6	VALUEAT ADDROF 4 5	4
6	**VALUEAT ADDROF 5 4**	**4**
6	VALUEAT ADDROF 4 POWEROF 5	5
6	VALUEAT ADDROF POWEROF 3 4	5
6	VALUEAT ADDROF POWEROF 5 4	5
135	**VALUEAT ADDROF 128 134**	**4**
135	VALUEAT ADDROF 134 128	4
135	VALUEAT ADDROF POWEROF 22 128	5
135	VALUEAT ADDROF 128 PLUS 133 134	6
1034	**VALUEAT ADDROF 1033 1024**	**4**
1034	VALUEAT ADDROF PLUS 1029 1029 1024	6
1034	VALUEAT ADDROF MULTIPLY 1025 324 1024	6
1034	VALUEAT ADDROF PLUS 1029 1024 1024	6
1034	VALUEAT ADDROF MULTIPLY 1029 324 1024	6
1034	VALUEAT ADDROF PLUS 1033 1033 1024	6

As one classifier represents the complete, accurate solution, instead of part solutions, the generalisation pressure exists on a global scale. If a classifier's condition is not general and optimal it will be replaced by the Action-set based genetic algorithm. Ultimately, there will be one macroclassifier for each reward returned by the environment.

The classical XCS produces a complete mapping of conditions to actions, including actions that are always incorrect (hence completely accurate in their prediction). The S_XCS system, which uses a restricted S-Expression alphabet, also produces incorrect, but accurate classifiers. However, S_XCS1, which has multiple functions available, only produces correct, accurate classifiers. It is hypothesised that multiple partially-incorrect classifiers and the complexity of the expression describing an incorrect classifier prevent their survival in the population. It is more likely to find a correct, accurate classifier quickly and thus these will dominate the population.

Redesigning subsumption deletion for S-Expressions is unnecessary for compacting the rule base as the coverage of each classifier is much greater, i.e. only one classifier is needed for the correct, accurate mapping. However, subsumption deletion may be needed for stabilising the performance close to optimum and

potentially for the control of bloat. With many different functions, it is difficult to determine the coverage and hence overlap of each individual classifier. Thus the evolved population consists of many similar classifiers with similar fitness, which would require a tuned deletion operator to remove slightly suboptimal classifiers. Hence, subsumption was not included in the developed techniques.

Utilising the same alphabet as Genetic Programming (GP) results in many similarities between GP and the techniques developed here, e.g. a single classifier solution instead of a cooperation of classifiers. However, a guided population based search is employed instead of a beam search, where the current best classifiers are evolved [10]. The online learning abilities together with the inherent generalisation pressure of XCS are still preserved. The need for a multistage approach to learning is demonstrated in the layered learning genetic programming approach [11].

Comparing the use of abstraction on the 6MUX problem (figures 7 and 8) shows that abstraction assists the S-Expression XCS to learn more effectively and faster - although not more efficiently as the optimum population set has to be created initially. This was due to the search space having already been reduced by generalisation. This was then followed by abstraction, instead of attempting both simultaneously [12]. Abstraction reduced the quantity of irrelevant functions in a classifier during the initial stages of training, which may prevent the system becoming trapped in local optima. Abstraction also scaled well on hypothesised OPSs, which were tested up to 1034MUX.

It is not surprising that the use of tailored operators improved the scalability of the LCS technique. Nor would it be difficult to create the functions using domain knowledge in this toy problem. However, in unknown problems, where scaling is feasible, the ability to automatically determine the functions that are required as the problem difficulty increases is considered valuable. This was most easily achieved using abstraction.

6. CONCLUSIONS

The classical XCS approach has difficulties scaling to problems over the 70-MUX problem due to growth in environmental search space. The use of pre-constructed functions such as the VALUEAT and ADDROF requires domain knowledge to reduce the search space, but the alphabet search space, when many such functions are included, now becomes prohibitive. Utilising the classical XCS to form an Optimal population set reduced the possible search space for the alphabet. Hypothetically, the novel system can solve the 1034-MUX problems, but the required OPS can not be leant. The use of such abstraction could assist the scalability of many Machine Learning techniques, e.g. allowing building blocks useful to a genetic algorithm to form.

7. REFERENCES

[1] Wilson, S. W., ZCS: A Zeroth-level Classifier System. *Evolutionary Computation, 2, 1* (1994), 1-18.

[2] Wilson, S. W., Classifier Fitness Based on Accuracy. *Evolutionary Computation*, *3, 2,* (1995), 149-76.

[3] Holland, J. H., *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[4] Wilson, S. W., Generalization in the XCS Classifier System. In J. Koza et al editors, *Genetic Programming: Proceedings of the Third Annual Conference*, (San Francisco, CA), Morgan Kaufmann, 1998, 665-674.

[5] Kovacs, T., *Evolving Optimal Populations with XCS Classifier Systems*. MSc Thesis, University of Birmingham. Also Technical Report CSR-96-17 and CSRP-96-17, School of Computer Science, University of Birmingham, Birmingham, U.K., 1996.

[6] Kovacs, T., XCS Classifier System Reliably Evolves Accurate, Complete, and Minimal Representations for Boolean Functions. *In Roy, Chawdhry, and Pant, editors, Soft Computing in Engineering Design and Manufacturing,* Springer-Verlag, 1997, 59-68.

[7] Koza, J., *Genetic Programming*. MIT Press, 1992.

[8] Lanzi, P-L., and Perrucci, A., Extending the Representation of Classifier Conditions Part II: From Messy Coding to S-Expressions. *In Proceedings of the Genetic and Evolutionary Computation Conference* (Orlando, Florida, 1999), Morgan Kaufmann, 1999, 345-352.

[9] Butz, M. V., *Rule-Based Evolutionary Online Learning Systems: A Principled Approach to LCS Analysis and Design*, Series: Studies in Fuzziness and Soft Computing, Vol. 191, 2006.

[10] Banzhaf, W., Nordin, P., Keller, R. E., and Francone, F D., *Genetic Programming - An Introduction*. San Francisco, CA: Morgan Kaufmann Publishers and Heidelberg, Germany: dpunkt.verlag, 1998.

[11] Gustafson, S .M. and Hsu, W. H., Layered learning in genetic programming for a cooperative robot soccer problem. *In J.F. Miller et al, editor, Genetic Programming, Proceedings of EuroGP,* 2001, 291-301.

[12] Browne, W., and Scott, D., An abstraction agorithm for genetics-based reinforcement learning. *In Proceedings of the Genetic and Evolutionary Computation Conference*, (Washington D. C., USA, 2005), ACM Press 2005, 1875-1882.