A Representational Ecology for Learning Classifier Systems

James A. R. Marshall
Department of Computer Science
University of Bristol
Bristol BS8 1UB, U.K.
marshall@cs.bris.ac.uk

Tim Kovacs
Department of Computer Science
University of Bristol
Bristol BS8 1UB, U.K.
kovacs@cs.bris.ac.uk

ABSTRACT

The representation used by a learning algorithm introduces a bias which is more or less well-suited to any given learning problem. It is well known that, across all possible problems, one algorithm is no better than any other. Accordingly, the traditional approach in machine learning is to choose an appropriate representation making use of some domain-specific knowledge, and this representation is then used exclusively during the learning process. To reduce reliance on domain-knowledge and its appropriate use it would be desirable for the learning algorithm to select its own representation for the problem. We investigate this with XCS, a Michigan-style Learning Classifier System. We begin with an analysis of two representations from the literature: hyperplanes and hyperspheres. We then apply XCS with either one or the other representation to two Boolean functions, the well-known multiplexer function and a function defined by hyperspheres, and confirm that planes are better suited to the multiplexer and spheres to the sphere-based function. Finally, we allow both representations to compete within XCS, which learns the most appropriate representation for problem thanks to the pressure against overlapping rules which its niche GA supplies. The result is an ecology in which the representations are species.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*Concept learning*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*

General Terms

Algorithms

Keywords

bias, GBML, hyperplanes, hyperspheres, Learning Classifier Systems, No Free Lunch, representation, XCS

1. INTRODUCTION

The No Free Lunch theorem [15, 16, 13], in its various forms, is a central result for Machine Learning (ML) and for optimisation. One form of the theorem states that no one learning algorithm is any better than another, across all possible problems to which it might be applied [15]. Consequently, naïvely applying an ML algorithm 'off the shelf' is unpromising. The problem for the ML practitioner, then, is to acquire some domain knowledge on the problem to be learned, and apply this in the design of critical aspects of the algorithm. One such critical aspect of algorithm design is selection of the representation to use. Examples of such practice are now widespread in the literature, such as the careful selection of representation and operators when applying Genetic Algorithms to combinatorial optimisation problems [8]. However, one possible approach to mitigating this impasse suggests itself; by providing an algorithm with several different representations, it may be able to learn to apply the most suitable without recourse to domain knowledge. To demonstrate this concept, we investigate two different representations, hyperplanes and hyperspheres, for learning a well known boolean function, the multiplexer. We examine the differences between the representations, and demonstrate the superiority of hyperplanes for this problem. We then introduce a new boolean function, designed to be most efficiently represented with hyperspheres. We conclude by introducing a modified version of the Learning Classifier System (LCS) XCS, able to simultaneously learn with classifiers of different representations, and present numerical results on its application to the two problems considered. As this extended LCS does not allow genetic recombination between classifiers of different representational type, it may be thought of as an ecology of species competing within the same environment.

2. HYPERPLANES AND HYPERSPHERES FOR BOOLEAN FUNCTIONS

Different representations are prevalent in different classification algorithms. For simplicity we restrict ourselves to considering classifiers for boolean functions, which provides a good starting point for future analytical and empirical work on richer representations. However, even the boolean domain includes the key feature of representational bias that we want to study.

For Learning Classifier Systems, such as XCS [14], a ternary alphabet is commonly used to define classifiers. Classifiers are strings of 0s and 1s, indicating that a match between the corresponding bits in the classifier and the instance being classified is required at that position, and wildcard symbols (#s), indicating that the classifier will match any instance at that particular position. A classifier matches an instance if and only if all of the specified bits on the classifier's string match all the corresponding bits on the instance string. Thus, the traditional LCS representation defines classifiers as hyperplanes of dimension d on a n-dimensional space, where d is the number of wildcard symbols on the classifier string, and n is the length of the classifier or instance strings.

An alternative representation is sometimes used in certain types of Artificial Immune System (AIS) [9] and also in some forms of LCS [2]. Here, classifiers are represented as a fully-specified binary string, and by a number, corresponding to a Hamming-distance within which instances are considered to be matched by the classifier. Hence, such a representation defines classifiers as hyperspheres of radius r, where r is the Hamming-distance within which instances are considered to be matched by the classifier. An instance that differs at r or less bits from a classifier of radius r is matched by that classifier. Hyperspheres are a form of the "partial matching" introduced by Booker[3].

Knowing that the representation used in a learning algorithm results in learning bias within that algorithm, we should already suspect that these two different representations, hyperplanes and hyperspheres, are likely to have different characteristics, leading to different performance. It is interesting at this point to investigate these characteristics in some detail.

The first point to note is that hyperplanes and hyperspheres are not equivalent types of representation in the following significant sense: the generalisation of hyperplanes is conditioned on individual dimensions while that of hyperspheres is not. That is, a hyperplane selects *which* dimensions it generalises over (using hashes) while a hypersphere merely specifies *how many* dimensions to generalise over (using a radius).

This indifference as to which dimensions match makes it difficult to express many concepts with hyperspheres. One option is to use many very specific spheres. Another is to use default hierarchies. We anticipate that there may be a greater need for default hierarchies when using spheres than using planes (see the appendix for one example), although we have yet to investigate this in full.

Another observation is that spheres grow more quickly than planes as we generalise them. That is, increasing the radius of a sphere by one results in it matching more new states than adding one hash to a plane; planes are a finer-grained representation. (This too suggests that default hierarchies may be more useful with spheres than planes.)

The size (number of input instances matched) of a plane is given by

$$2^d,$$

where $0 \leq d \leq n$ is the dimension of the hyperplane. Note that for a given d this expression is independent of n, the problem size. In contrast, hypersphere size increases monotonically with n as

$$\sum_{k=1}^{r} \binom{n}{k},$$

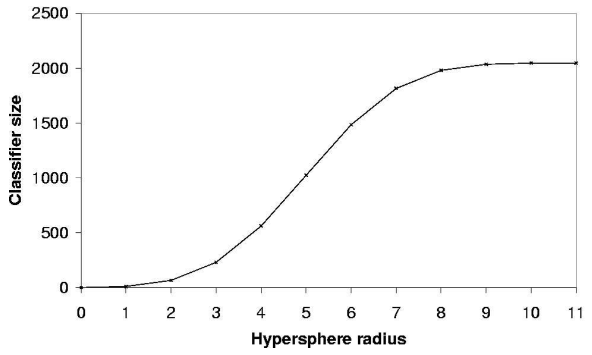

Figure 1: Hypersphere sizes on 11-bit functions

where $0 \leq r \leq n$ is the radius of the hypersphere. For illustration, the sizes of hyperplanes and hyperspheres of varying radius and dimension on an 11-bit boolean function are shown in figures 1 and 2 respectively.

It is also interesting to investigate the space of possible classifiers of both types. The number of possible hyperplanes is a function of both hyperplane dimensionality d, and problem size n, being given by

$$\binom{n}{d} 2^{(n-d)}.$$

The number of hyperplanes of different dimension on an 11-bit boolean function is illustrated in figure 3.

In contrast, the number of hyperspheres is constant for any specified radius, depending only on problem size n, and assuming all possible radii of hypersphere can be specified is given by

$$(n+1)2^n.$$

As problem size n increases, the search space for hyperplanes will become much larger than that for hyperspheres (see the appendix for further details).

Having examined the differences in the characteristics of the two different representations, let us now examine their relative performance on a well-known problem, the boolean multiplexer.

An n-bit boolean multiplexer is an addressing problem with n-bit problem instances, made up of k address bits, which specify an index into the following 2^k data bits, where $n = k + 2^k$. The class of a particular instance is given by

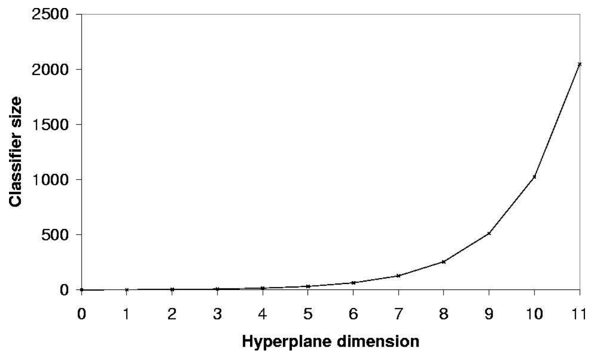

Figure 2: Hyperplane sizes on 11-bit functions

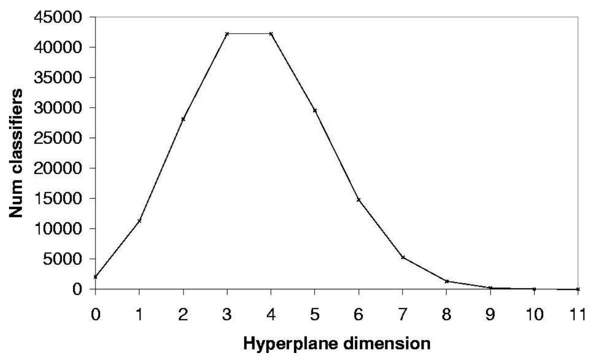

Figure 3: Number of hyperplanes on 11-bit functions

the value of the data bit indexed by the address bits. The smaller multiplexer problems, such as the 6-bit and 11-bit multiplexer, are sufficiently small that we can enumerate all possible classifiers for those problems, both hyperplane-based and hypersphere-based. By evaluating the instances covered by any given classifier and assigning the majority class of those covered instances to that classifier, we may evaluate the distribution of classifier accuracies for hyperplane and hypersphere classifiers. Note that such a distribution of accuracies will be in the interval $[50, 100]\%$, as we are assigning the majority class to the classifiers. The accuracy distributions for hyperplane and hypersphere classifiers of a particular size are illustrated for the 11-bit multiplexer in figures 4 and 5 respectively. For a fair comparison the dimensionality of the hyperplanes and the radius of the hyperspheres have been set such that the classifiers are of similar size under both representations ($2^4 = 16$ for hyperplanes, and $\binom{11}{1} + \binom{11}{0} = 12$ for hyperspheres).

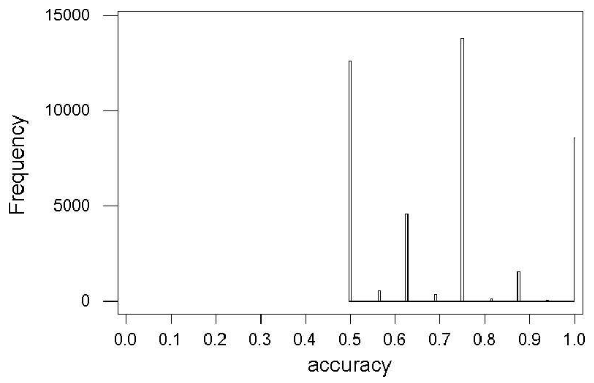

Figure 4: Hyperplane (dimension 4) accuracy distribution on 11-bit multiplexer

Note that, despite the much larger search space for hyperplanes, 1/4 of all possible 4-dimensional classifiers are 100% accurate. In contrast, 100% accurate hyperspheres do not occur at all in the accuracy distribution presented in figure 5. In fact, it can easily be proved that for the boolean multiplexer no hypersphere classifier of any generality (i.e. with radius > 0) can be 100% accurate (see the appendix for details).

Let us also briefly consider at this point hyperplanes of lower dimension. Figure 6 shows the accuracy distribution for hyperplane classifiers of dimension 2, from which it can be seen that 100% accurate classifiers are the mode. This suggests a possible explanation for earlier observations of the effectiveness of randomly generated classifiers on the boolean multiplexer [11].

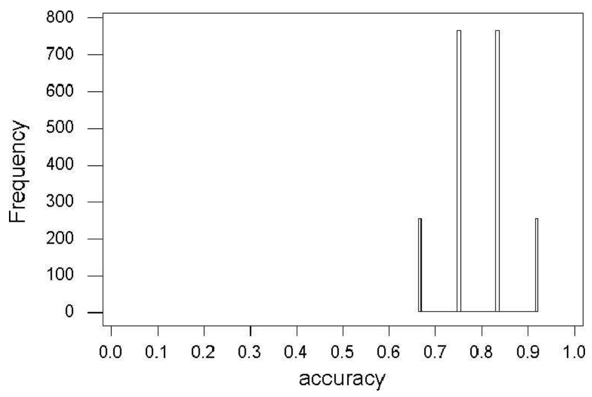

Figure 5: Hypersphere (radius 1) accuracy distribution on 11-bit multiplexer

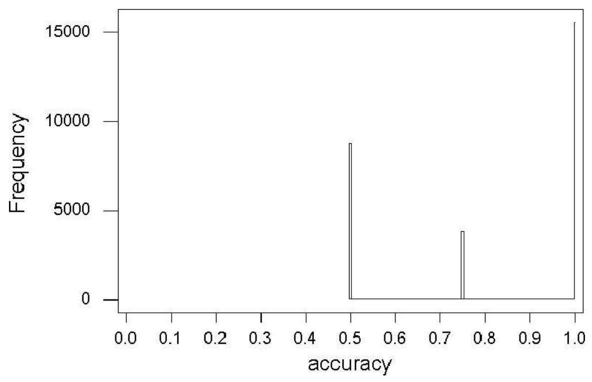

Figure 6: Hyperplane (dimension 2) accuracy distribution on 11-bit multiplexer

Thus, despite the substantially larger search space for hyperplanes compared with hyperspheres noted above, it seems that finding a set of accurate hyperplane classifiers for a boolean multiplexer is in fact likely to be much easier than finding a set of accurate hypersphere classifiers. This is unsurprising, given that the basic definition of the boolean multiplexer is effectively given in terms of hyperplanes where the only specified bits are the address bits and the one indicated data bit. As such, multiplexer problems are a staple of the LCS literature (for a comprehensive list see [11]). However, we should also be able to find a boolean function in which hyperspheres significantly outperform hyperplanes, by describing such a function in terms of hyperspheres. It would then be interesting to evaluate the performance of a classification algorithm on these different problems, when equipped with different classifier representations. To achieve this, we first need to extend a classification algorithm to support such different representations.

3. HYPERSPHERE CLASSIFIERS IN XCS

We extended XCS [14] to support hypersphere classifiers

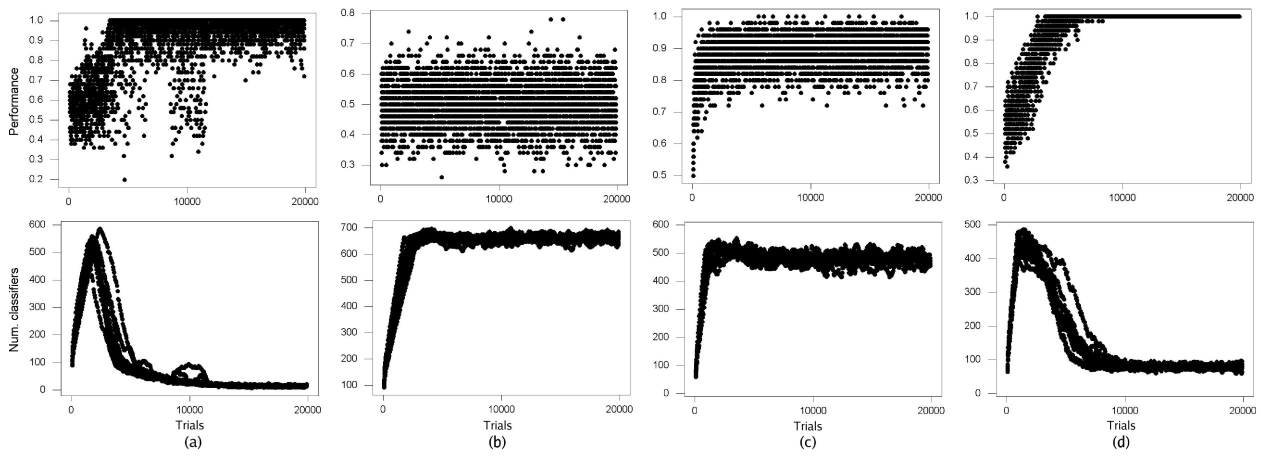

Figure 7: Control results for XCSphere initialised only with hyperspheres on (a) sphere function, (b) multiplexer; and for XCS on (c) hypersphere function, (d) multiplexer. Proportion of correctly classified instances, and classifier population size, are plotted against generation number

in addition to the traditional hyperplane classifiers. To achieve this, we took an existing implementation of XCS [4], and modified it appropriately[1]. The extensions are for the most part trivial. We must introduce a new classifier class, defined by a binary string and a radius in the interval $[0, n]$, where n is the problem size. We must also define a new matching function for this classifier class, based on Hamming distance. We must then introduce new constants to govern the mean and variance of the distribution from which radii for new hyperspheres are assigned, the probability of mutating the radius of a hypersphere when applying the genetic algorithm to it, and the variance of the distribution from which changes to a hypersphere radius during mutation are drawn. XCS' crossover operator is applied to hypersphere classifiers in the same way as it is to the traditional hyperplane classifiers. However, the crossover operator cannot be applied to two classifiers of different type. Hence, the classifiers using different representations may be thought of as different species that cannot mate with each other.

The only problem of any consequence that must be solved in introducing hypersphere classifiers into XCS is designing an efficient algorithm for checking when one hypersphere is more general than another. This can be achieved in constant time with the simple function

$$g(\delta, r_l, r_m, n) := \begin{cases} \text{false} & \text{if } r_l = r_m \\ \min(r_l + \delta, n) \leq r_m & \text{otherwise} \end{cases} \quad (1)$$

where δ is the distance between the centres of the two hyperspheres under consideration, r_l is the radius of the hypersphere believed to be less general, r_m is the radius of the hypersphere believed to be more general, and n is the length of problem instances. If the inequality is satisfied, then the more general hypersphere is indeed more general, *except in the case where the radii of both hyperspheres are the same*. This exception is required as one of the classifiers needs to be genuinely more general than the other; in other words it must cover more instances. The inequality in equation 1 above would be satisfied if the two hyperspheres had the same radius and were centred on the same instance, yet these two classifiers would be equally general. However, this case will not occur if we are using macroclassifiers [14], as the identical classifiers will be represented by a single macroclassifier.

All that remains to do to complete the extension is to introduce one final constant, which controls the ratio of hypersphere classifiers to hyperplane classifiers with which XCS is initialised.

4. RESULTS

We are interested in whether, for a given problem, a classification algorithm can learn to use the most appropriate representation from among several it has available to it. We shall investigate this using our extended version of XCS, named XCSphere and introduced in the previous section, as applied to two boolean function problems. The first of these is the boolean multiplexer which, as already discussed, is particularly suited to representation using hyperplane classifiers. The second problem we shall design such that it is more suited to representation by hypersphere classifiers.

The simplest way to define a boolean function such that it is easily represented by hyperspheres is as follows. Assume a boolean function with an odd number of bits n, and assign classes to instances of the function according to their Hamming distance from one of two instance strings: the instance string consisting only of 0s, and the instance string consisting only of 1s. Those instances that are within Hamming distance $(n-1)/2$ of the 0s string shall be assigned class 0, and those that are within Hamming distance $(n-1)/2$ of the 1s string shall be assigned class 1. This function has the advantages of being unambiguously describable by hyperspheres, and requiring generalisation (i.e. hyperspheres of radius > 0) in order to be described efficiently. It is related to the count ones problem used with LCS in e.g. in [6].

In figure 7 we present control results for two different classification algorithms, the standard XCS[2], and our version

[1]the source code for the extended XCS implementation is downloadable from http://www.cs.bris.ac.uk/~marshall/

[2]for the standard XCS we used Martin Butz's implementation with the following settings: $N = 800$, $\alpha = 0.1$, $\beta = 0.2$,

extended to use hypersphere classifiers instead of hyperplanes (XCSphere[3]). We collected control results on performance (proportion of presented instances correctly classified) and classifier population size, using our two test functions, the hypersphere function and the multiplexer, with 100 replicates of each experiment. Note that the figures are scatter plots of 100 replicates rather than the single average curve normally plotted for XCS. As expected, XCSphere performs well on the hypersphere function (7a), but fares less well on the multiplexer (7b). Also as expected, the traditional XCS performs well on the multiplexer (7d), but struggles on the hypersphere function (7c).

In the experiments where the appropriate representation is being used for a problem (7a and 7d), performance rapidly converges on the maximum level in many if not all replicates. Simultaneously in these experiments, after an initial exploratory proliferation, the number of classifiers being maintained by the algorithm converges towards the minimum number needed to represent the problem (2 for the hypersphere function, $2^4 = 16$ for the multiplexer), although additional classifiers are maintained due to XCS' balance between exploration and exploitation.

In contrast, in the experiments where the inappropriate representation is being used for a problem (7b and 7c), the performance level reached is lower and much more variable across replicates, with no replicate converging to a stable maximum performance level. Simultaneously, the number of classifiers explodes as the unsuitability of the representation forces the classification algorithm to use more classifiers to try and cover the instance space. In the worst case, the classification algorithm will be forced to enumerate the instance space and assign a fully specific classifier to each instance within it (a total of $2^{11} = 2048$ classifiers for the 11-bit functions studied here).

We can examine statistically the benefit of choosing the appropriate representation on the problems we have studied here. We shall compare the classification accuracy achieved after a learning period of 20,000 trials by both hyperplane and hypersphere classifiers on each of the two problems, with 100 replicates of each.

As the results in figure 7 suggest, the hypersphere-based XCS does significantly better on the hypersphere boolean function than does the hyperplane-based XCS ($P < 0.0005$, $U = 13581.5$, $N_A = N_B = 100$; Mann-Whitney U-test on performance). Similarly, the hyperplane-based XCS does significantly better on the multiplexer than the hypersphere-based XCS ($P < 0.0005$, $U = 15050$, $N_A = N_B = 100$; Mann-Whitney U-test on performance).

Having established that neither hyperplane classifiers nor hypersphere classifiers are generally superior on our two boolean test functions, let us now examine the effects of allowing XCSphere to choose between representations during learning. This is achieved by setting the constant that regulates the hyperplane/hypersphere classifier split at XCSphere's initialisation to be 0.5. Thus, when initialised, XCSphere will have equally sized populations of hypersphere and hyperplane based classifiers. During learning, these representations will compete with each other to be maintained in the classifier population. Results from applying this hybrid XCSphere to the two boolean functions are presented in figure 8.

$\delta = 0.1$, $\nu = 5$, $\theta_{GA} = 25$, $\varepsilon_0 = 10$, $\theta_{del} = 20$, $\chi = 0.8$, $\mu = 0.04$, $P_{\#} = 0.5$, $predictionErrorReduced = 0.25$, $fitnessReduction = 0.1$, $\theta_{sub} = 20$, $p_I = 10.0$, $\varepsilon_I = 0$, $F_I = 0.01$. GA and ActionSet subsumption were enabled

[3]for XCSphere the constants used were as for the standard XCS, with the additional constants set as follows: $newRadiusMean = 1$ and $newRadiusStdDev = 1$ control the radii of hyperspheres generated without parents while $radiusMutationProb = 0.5$ and $radiusMutationStdDev = 2$ control the mutation of radii of offspring hyperspheres

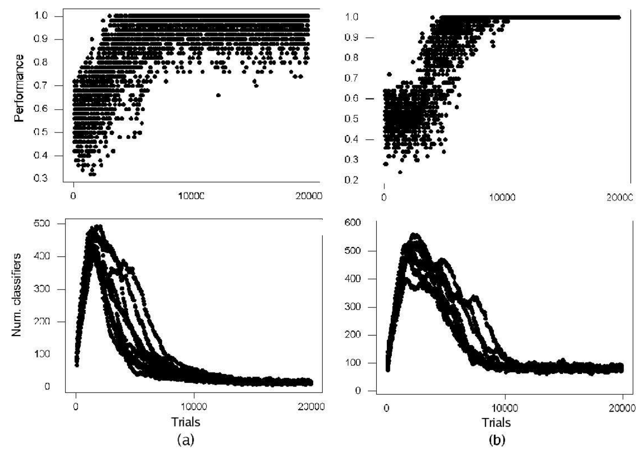

Figure 8: Results for XCSphere initialised with equal numbers of hyperplanes and hyperspheres on (a) hypersphere function, (b) multiplexer. Proportion of correctly classified instances, and classifier population size, are plotted against generation number

On examination the results for hybrid XCSphere, applied to the hypersphere function (8a) and the multiplexer (8b), appear encouraging. In both cases XCSphere has converged to 100% classification accuracy to a greater or lesser extent by the end of the learning period.

We can evaluate the performance of hybrid XCSphere more quantitatively, by statistically comparing its classification accuracy at the end of the learning period on each of the problems against the version of XCS using only the most suitable representation. We hypothesise that there is no significant difference between the accuracy achieved by the hybrid XCSphere and by the most appropriate single-representation XCS in each case. This is indeed shown to be the case for performance on the hypersphere function at the end of the learning period, for hybrid XCSphere and XCSphere using only hyperspheres ($P = 0.1508$, $U = 9488$, $N_A = N_B = 100$; Mann-Whitney U-test). The difference in performance on the multiplexer between the hybrid XCSphere and the hyperplane-based XCS is also clearly non-significant, to such an extent that we cannot usefully apply a Mann-Whitney U test; in the hyperplanes-based XCS 100% of replicates converged to 100% accuracy within the learning period, while for the hybrid XCSphere the figure was 99%, with the one replicate that did not converge achieving 99.6% accuracy at the end of the learning period.

At the same time, in both cases the number of classifiers in the population has initially exploded, and then converged on a number close to the optimal number of classifiers with which the problem can be described (if we are free to choose the appropriate representation). Tables 1 and 2 below show classifiers having fitness above 0.9 discovered by the hybrid XCSphere in a typical run, for each of the functions tested.

For the multiplexer, XCSphere discovers all 32 classifiers in %[O] – the minimal, accurate, non-overlapping solution [10]. Similarly, XCSphere discovers the 4 classifiers in %[O] for the hypersphere function. (Note that twice as many classifiers as are strictly needed are present, as for each classifier XCS maintains another one differing only in the action specified and in the prediction – a complete map of the reward function [14]).

To get a complete picture of the performance of hybrid XCSphere, we must also compare its time to converge (if at all[4]) on 100% classification accuracy with that of XCS and hyperspheres-only XCSphere. We might expect that the hybrid XCSphere will take longer to learn a problem than the appropriate dedicated-representation classifier system will. This is indeed shown to be the case for the one-tailed hypothesis that hybrid XCSphere is slower to converge than XCS on the multiplexer ($t = -4.41$, $P < 0.0005$, $N = 100$; 2-sample t-test). Mean trials to converge for traditional XCS is 6731, (std. dev. 2509), compared to 8335 (std. dev. 2634) for hybrid XCSphere. However, the one-tailed hypothesis that hybrid XCSphere is slower to converge than hyperspheres-only XCSphere on the hypersphere problem is not supported ($U = 9607$, $N_A = N_B = 100$; Mann-Whitney U-test), and in fact there is even weak evidence that hybrid XCSphere is the quicker to converge on the hypersphere problem ($P = 0.1125$, $U = 9607$, $N_A = N_B = 100$; Mann-Whitney U-test). However, we note that the sphere problem is very simple and perhaps not a good basis for such a comparison.

5. DISCUSSION

This paper has demonstrated the ability of XCS, given a choice of two alternative representations, to make use of the most appropriate one in learning a boolean function. In the experiments presented here a hybrid version of XCS is initialised with a population of random classifiers, half of which are hypersphere-based and half hyperplane-based. This hybrid XCSphere is then able to converge on the most efficient hyperplane-based set of classifiers for the multiplexer, or the most efficient hypersphere-based set of classifiers for a hypersphere-based function we designed. The statistical tests on performance of the hybrid XCS against dedicated versions restricted to use only the most appropriate representation for the problem tested again, showed no significant difference in classification accuracy achieved at the end of the learning period. However, significant differences *were* shown between applying the appropriate and inappropriate representations to a given problem within the single-representation version of XCS. XCS should be particularly good at exploiting the most suitable representation due to its strong competition between overlapping classifiers [10]. This ability to exploit the most suitable representations on the problems tested against has allowed XCSphere to outperform, in terms of classification accuracy, either of the pure-representation versions of XCS when evaluated against both boolean functions considered. This suggests a possible escape from the problems raised by representational bias. However, while on the hypersphere problem hybrid XCSphere was no slower to converge on 100% classification accuracy than hyperspheres-only XCSphere, and was even slightly faster, for the multiplexer XCSphere was significantly slower than traditional XCS.

[4]replicates failing to converge were assigned a convergence time equal to the total number of trials in the learning period

Table 1: Classifiers of fitness 0.9 or above discovered by XCSphere on the multiplexer during a typical run

condition	action
100####1###	0
001#0######	1
111#######0	0
0001#######	1
101#####1##	0
111#######1	1
110######1#	0
100####0###	1
011###1####	1
010##1#####	1
010##1#####	0
0000#######	1
101#####0##	0
111#######0	1
011###1####	0
010##0#####	0
0001#######	0
001#1######	1
101#####0##	1
011###0####	0
111#######1	0
110######1#	1
001#0######	0
011###0####	1
110######0#	1
010##0#####	1
110######0#	0
100####0###	0
101#####1##	1
100####1###	1
0000#######	0
001#1######	0

Table 2: Classifiers of fitness 0.9 or above discovered by XCSphere on the hypersphere function during a typical run

condition	radius	action
11111111111	5	0
11111111111	5	1
00000000000	5	1
00000000000	5	0

Other GBML work has used multiple representations to reduce the impact of representational bias. Llorà & Wilson's [12] approach is to construct heterogenous decision trees which may have different classes of internal node that partition the instance space in different ways. These hybrid decision trees are created in a Pittsburgh-style LCS, by allowing subtrees from trees using different representations to be recombined to produce a new hybrid decision tree.

Within our hybrid XCS implementation, classifiers of different representational type are manipulated separately by the genetic algorithm, so there can be no recombination of classifiers that make use of different representations. Thus classifiers of each representation may be thought of as belonging to two separate species, that interact together and compete for fitness within the same environment. Hence, our approach to mitigating the effects of representational bias can perhaps best be thought of as introducing an ecological aspect into LCS. This approach differentiates our work from that of Llorà & Wilson [12]. Our work is also distinguished by the fact that we make only minor modifications to the widely-known XCS algorithm to achieve our representational ecology; we exploit the existing strong pressure XCS has against overlapping rules due to its niche GA [10].

Bacardit, Goldberg and Butz also studied competing representations in a Pittsburgh LCS [1]. In this work rules with different default classifications competed and the system selected suitable default rules for the task at hand. One of our reviewers pointed out that this work found it necessary to encourage diversity in representations later in the run to avoid premature convergence to one representation, and that a similar effect might occur in XCS. This reviewer suggested allowing mutation from one representation to another might help in this respect and this potential problem and solution are certainly worth future investigation.

Other related work includes [5] in which XCSF (XCS with function approximation – classifiers compute outputs based on inputs) is modified to use hyperspheres and hyperellipses, with good results compared to the previously used hyperrectangles. A natural extension of that work and the current one would be to introduce an ecology of real-valued representations into XCSF.

Future work will probably focus on enabling rule hierarchies in our implementation of XCS. This is likely to be of particular importance for the application of hyperspheres to boolean functions. In this paper we have considered one boolean function which can be unambiguously described by hyperspheres. However, our initial investigations suggest that many boolean functions for which hyperspheres may be suitable can only be tackled if action selection is influenced by the degree of match between a rule and an input. That is, hyperspheres require hierarchies of default and exception rules to efficiently represent many functions. For example the parity function, which is very hard to describe using hyperplanes, may be more easily described by hyperspheres, *but only if the degree of match is incorporated into the action selection mechanism.* Indeed, for this reason the hypersphere function we studied was much simpler in terms of number of classifiers required for coverage, compared to the multiplexer. We also anticipate developing our formal understanding of the mechanisms required to support the use of different representations, extending the range of representations we consider (e.g. to include "r-chunks" representations [7]), and extending the problem set we test them against. In particular, the extended problem set will include problems which are best represented by a mixture of classifiers, as done by Llorà & Wilson [12].

6. ACKNOWLEDGEMENTS

Thanks are due to Bob Planqué for help with the scripts used in analysing the results, to Martin Butz whose XCS code we used and extended, and to the anonymous reviewers for insightful comments.

7. REFERENCES

[1] J. Bacardit, D. E. Goldberg, and M. V. Butz. Improving the performance of a Pittsburgh learning classifier system using a default rule. To appear in: Advances at the frontier of Learning Classifier Systems. Springer, 2006.

[2] J. Bassett. *A Study of Generalization Techniques in Evolutionary Rule Learning.* MSc thesis, George Mason University, Fairfax VA, USA, 2002.

[3] L. B. Booker. Improving the performance of genetic algorithms in classifier systems. In J. J. Grefenstette, editor, *Proc. of the 1st International Conference on Genetic Algorithms and their Applications (ICGA85)*, pages 80–92. Lawrence Erlbaum Associates, 1985.

[4] M. V. Butz. XCSJava 1.0: An Implementation of the XCS classifier system in Java . Technical Report 2000027, Illinois Genetic Algorithms Laboratory, 2000.

[5] M. V. Butz. Kernel-based, ellipsoidal conditions in the real-valued XCS classifier system. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2005)*, pages 1835–1842. ACM, 2005.

[6] 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*, 11:239–277, 2003.

[7] F. Esponda, S. Forrest, and P. Helman. A formal framework for positive and negative detection schemes. *IEEE Transactions on Man, Systems and Cybernetics Part B*, 34:357–373, 2004.

[8] E. Falkenauer. *Genetic Algorithms and Their Adaptation to Grouping and Clustering Problems.* Wiley, 1998.

[9] J. D. Farmer, N. Packard, and A. Perelson. The immune system, adaptation, and machine learning. *Physica D*, 22:187–204, 1986.

[10] T. Kovacs. *Strength or Accuracy: Credit Assignment in Learning Classifier Systems.* Springer, 2004.

[11] T. Kovacs and M. Kerber. High classification accuracy does not imply effective genetic search. In *Proc. of the Genetic and Evolutionary Computation Conference (GECCO-2004)*, pages 785–796. Springer, 2004.

[12] X. Llorà and S. W. Wilson. Mixed decision trees: Minimising knowledge representation bias in LCS. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004)*, pages 797–809. Springer, 2004.

[13] N. Radcliffe and P. Surry. Fundamental limitations on search algorithms: Evolutionary computing in perspective. In J. van Leeuwen, editor, *Computer Science Today: Recent Trends and Developments, LNCS 1000*, pages 275–291. Springer-Verlag, 1995.

[14] S. W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.

[15] D. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8:1341–1390, 1996.

[16] D. Wolpert and W. MacReady. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1995.

APPENDIX

THEOREM 1. *No hypersphere classifier for the multiplexer can achieve both generality and 100% accuracy*

PROOF. A general hypersphere classifier is centred on an instance, which is of a particular class. By definition of the hypersphere, this general classifier has radius ≥ 1 and thus also includes all other instances that differ at one point on the bit string from the central instance. So, the classifier will include the instance which is identical to the central instance in all but indexed bit, and hence is of a different class. Therefore, a hypersphere classifier of any generality always includes instances of both classes and hence cannot be 100% accurate $\square$

THEOREM 2. *The space of possible hypersphere classifiers for a boolean function is much smaller than the space of possible hyperplane classifiers for that function*

PROOF. The number of possible hypersphere centres is given by 2^n while the number of possible hyperplane classifiers is given by 3^n, where n is function length. Even allowing that all possible radii $0 \leq r \leq n$ of hyperspheres may be required to cover a boolean function, the space of possible hypersphere classifiers will still remain much smaller, for as n increases $(n+1)2^n \lll 3^n$ $\square$