Function Approximation with XCS: Hyperellipsoidal Conditions, Recursive Least Squares, and Compaction

Martin V. Butz*, Pier Luca Lanzi†, Stewart W. Wilson‡

*Department of Cognitive Psychology
University of Würzburg
Röntgenring 11, 97070 Würzburg, Germany
butz@psychologie.uni-wuerzburg.de

†Artificial Intelligence and Robotics Laboratory
Dipartimento di Elettronica e Informazione
Milano 20133, Italy
pierluca.lanzi@polimi.it

‡Prediction Dynamics
Concord, MA 01742, USA
wilson@prediction-dynamics.com

Abstract

An important strength of learning classifier systems (LCSs) lies in the combination of genetic optimization techniques with gradient-based approximation techniques. The chosen approximation technique develops locally optimal approximations, such as accurate classification estimates, Q-value predictions, or linear function approximations. The genetic optimization technique is designed to distribute these local approximations efficiently over the problem space. Together, the two components develop a distributed, locally optimized problem solution in the form of a population of expert rules, often called classifiers. In function approximation problems, the XCSF classifier system develops a problem solution in the form of overlapping, piecewise linear approximations. This paper shows that XCSF performance on function approximation problems additively benefits from (1) improved representations, (2) improved genetic operators, and (3) improved approximation techniques. Additionally, this paper introduces a novel closest classifier matching mechanism for the efficient compaction of XCS's final problem solution. The resulting compaction mechanism can boil the population size down by 90% on average, while decreasing prediction accuracy only marginally. Performance evaluations show that the additional mechanisms enable XCSF to reliably, accurately, and compactly approximate even seven dimensional functions. Performance comparisons with other, heuristic function approximation techniques show that XCSF yields competitive or even superior noise-robust performance.

Index Terms

Learning classifier systems, LCS, XCS, function approximation, hyperellipsoids, condensation, compaction, self organization, neural networks.

I. Introduction

Learning classifier systems (LCSs) are rule-based, evolutionary learning systems that are designed to efficiently combine gradient-based approximation techniques with evolutionary optimization techniques. The XCS classifier system, introduced by Wilson in 1995 [1], may be regarded as the most prominent learning classifier system (LCS) to-date. XCS is an accuracy-based LCS that it is designed to learn maximally accurate predictions for any given input and available action combination. Its function approximation form, XCSF [2], [3], develops overlapping, piecewise-linear function approximations.

It was shown that the combination of local gradient-based optimization and global, evolutionary-based optimization in XCS(F) yields a flexible, iterative learning algorithm. XCS was successfully applied to large binary classification tasks [4], [5], [1], [6], real-world datamining problems [7], [8], [9], [10], [11], challenging reinforcement learning problems [9], [12], and function approximation problems [3], [13], [9]. These successful applications confirm the flexibility of the XCS learning architecture. Moreover, theoretical analyses of XCS have confirmed that the system is guaranteed to evolve accurate problem solutions for a wide range of problems with high probability in polynomial time [14].

Most recently, extensions of XCSF have focused on improving condition structures as well as predictive capabilities. Condition structures have been modified to represent and evolve different hyperrectangular structures [15], [11], [3] as well as other condition structures including various spheroids [16], [9], [17], convex hulls [18], and tile codings [12]. Linear approximations have been extended to polynomial predictions [13]. Moreover, the predictive capabilities have been improved by replacing the original least-mean-square delta rule [19] with the pseudo-inverse method, recursive least squares (RLS) method [13], [20], or Kalman-filtering-based approximation techniques [21]. Additionally, the XCS framework has been successfully combined with pure NN-based approximation techniques [22], [23].

This paper aims at putting these advancements in perspective, comparing hyperrectangular with hyperellipsoidal condition structures on a variety of function approximation problems. We examine the improvements achieved due to both the modified condition structure and the improved linear approximation technique, that is, RLS. The introduced improvements confirm that XCS performance can be additively optimized (1) in its space partitioning representation, (2) in the evolutionary operators, and (3) in the linear approximation technique.

Moreover, this paper addresses problem solution compaction. Since XCSF relies on a population-based learning technique (the evolutionary learning component), problem solutions are represented redundantly by multiple, similar, strongly overlapping classifiers. Thus, an important part of LCS-based approximation is effective population compaction [24], [25]. To develop a final, complete, but compact problem solution, we introduce a novel compaction approach, which relies on a competitive matching mechanism, termed *closest classifier matching*. The results show that compaction works highly effective. During compaction, the population size often decreases by over 90% while hardly affecting function approximation accuracy.

Finally, this paper compares XCSF performance with other function approximation techniques available in the literature. We show that XCSF clearly outperforms pure clustering mechanisms, exemplified in the Neural GAS architecture [26], [27]. We also show that XCSF evolves solutions similar to those generated by a constructive, incremental learning approach [28] as well as similar to an incremental learning linear model tree algorithm [29]. Performance is competitive showing successful generalization capabilities, accurate approximations, and noise robustness. In comparison to the other algorithms, though, XCSF is the most flexible learning algorithm, applicable to a large variety of problems.

This paper is structured as follows. First, we give a short introduction to the XCSF system. Next, we show how hyperellipsoidal condition structures can improve system performance. Following that, we show the more robust RLS technique for linear approximation. Finally, we introduce the compaction mechanisms. A performance suite on functions of up to seven dimensions confirms the strength and robustness of the introduced mechanisms. Finally, we compare XCSF with the Neural GAS clustering algorithm and two statistical, incremental learning approaches. Summary and conclusions put the results in a broader perspective.

II. XCSF Overview

XCS is a typical Michigan-style learning classifier system (LCS) [30], [31]. The following introduction of XCS describes the enhanced XCS system for function approximation, often termed XCSF [11], [3]. For detailed information on XCS the interested reader is referred to the algorithmic description of XCS [32].

A. Representation

XCSF is a function approximation system that evolves overlapping, typically piecewise linear function approximations. Given at time t an input vector $\vec{x_t} = (x_1, ..., x_n) \in \mathcal{S} \subseteq \Re^n$, XCSF determines its function value prediction P and receives as feedback the actual function value y_t. Using this information, XCSF iteratively evolves its solution representation within a population of *classifiers* (condition-prediction rules). Each classifier specifies in its condition part its applicability, and in its prediction part its function value prediction—typically a linear prediction. Thus XCSF is a locally weighted learning approach [33], but one in which the local rule structures (classifiers) evolve by the means of a genetic algorithm. The classifiers partition the input space into overlapping, piecewise linear prediction surfaces. The resulting smoothed surface forms the function approximation surface.

More formally, a classifier in XCSF consists of a condition C, a prediction R, a prediction error ε, and a fitness value F. (1) The condition part C specifies a hyperrectangle by the means of interval encoding, that is, $C = (\vec{l}, \vec{u}) = ((l_1, l_2, ..., l_n)^T, (u_1, u_2, ..., u_n)^T)$, where ^{T} denotes the transpose ($\vec{l}$ and $\vec{u}$ are column vectors). (2) The prediction R specifies a linear prediction of the input vector x_t in the form of a weight vector, that is, $R = \vec{w} = (w_0, w_1, ..., w_n)^T$ where w_0 is the offset weight. The prediction is determined by the inner product $(\vec{x^*} - \vec{l^*})^T \vec{w}$ where vectors $\vec{x^*}$ and $\vec{l^*}$ are vectors $\vec{x}$ and $\vec{l}$, enhanced with a leading one and zero, respectively. (3)

The prediction error ε estimates the mean absolute deviation of the reward predictions. (4) The fitness F specifies the relative predictive accuracy of the classifier. While the condition parts evolve by the means of a genetic algorithm, the other components are iteratively approximated by the least-mean-square update technique [19].

B. Gradient-Based Rule Updates

Each learning iteration, XCSF generates a match set $[M]$ that contains all classifiers whose conditions match the input vector $\vec{x_t}$. The match set is used to generate the function value prediction. Given input vector $\vec{x_t}$, each classifier cl forms the prediction $cl.P_t = (\vec{x_t^*} - cl.\vec{l^*})^T cl.\vec{w}$. The fitness-weighted average of the predictions of all matching classifiers denotes the function value prediction, that is, $P_t = \sum_{c \in [M]} c.P_t c.F / \sum_{c \in [M]} c.F$.

The error signal to update reward prediction, prediction error, and fitness is the error between a classifier prediction and the actual value: $y_t - (\vec{x_t^*} - \vec{l^*})^T \vec{w}$. Each classifier in $[M]$ is updated according to its error signal using the delta update rule:

$$\vec{w} \leftarrow \vec{w} + \eta(y_t - (\vec{x_t^*} - \vec{l^*})^T \vec{w})(\vec{x_t^*} - \vec{l^*}), \tag{1}$$

where η denotes the learning rate.[1] The reward prediction error approximates the mean absolute deviation of its prediction by the following delta rule:

$$\varepsilon \leftarrow \varepsilon + \eta(|y_t - (\vec{x_t^*} - \vec{l^*})^T \vec{w}| - \varepsilon) \tag{2}$$

Classifier accuracy is determined by the scaled inverse of the error, whereby classifiers with mean absolute error lower than ϵ_0 are considered completely accurate. The fitness value is derived from the relative classifier accuracy in $[M]$ [32]. Thus, fitness reflects the current predictive quality of a classifier in comparison to all overlapping classifiers. The evolutionary algorithm selects classifiers dependent on their current fitness values. After rule updates and possible GA invocation in the current match set, the next iteration begins.

C. Rule Structure Evolution

XCSF is initialized with an empty population. Initial classifiers are generated by a covering mechanism that creates a matching condition given a problem instance $\vec{x}$ for which no classifier matches. The resulting interval size lies between 0 and $2r_0$ uniformly random in each dimension, where r_0 is an additional XCSF parameter that determines the width of initial classifiers.

Most classifiers (i.e., classifiers not generated by covering) are generated by the evolutionary component for which XCS applies a steady-state, niche GA. A GA is invoked if the average time since the last GA application upon the classifiers in $[M]$ exceeds a threshold θ_{GA}. The GA selects two parent classifiers from the current match

[1]Wilson [3] used a modified delta rule, instead, to stabilize the update mechanism. Comparative runs did not show any performance differences in the experiments reported in this paper.

set $[M]$ using set-size-relative tournament selection based on the classifier's fitness estimates [4]. Two offspring are generated from the selected parents. Each attribute of the offspring conditions is mutated with probability μ. In the hyperrectangular representation, mutation alters the lower or upper boundary by increasing/decreasing its stretch, maximally doubling/halving the covered interval, respectively. For recombination, we utilize uniform crossover, in which any corresponding values in the two classifier conditions are exchanged with probability 0.5.

Before the offspring are inserted in the population, two classifiers may be deleted to keep a fixed population size N. Classifiers are deleted from $[P]$ with probability proportional to an estimate of the size of the match sets that the classifiers occur in. If the classifier is sufficiently experienced and its fitness F is significantly lower than the average fitness of classifiers in $[P]$, its deletion probability is further increased [34].

D. XCSF Learning Intuition

As can be seen, XCSF relies on the following two major learning components: (1) The gradient-based component generates linear approximations and estimates classifier utility based on the relative accuracy of the generated linear approximations. (2) The evolutionary component relies on these utility estimates (represented in classifier fitness) to evolve better classifiers by reproducing more accurate classifiers and deleting less accurate ones (on average).

The interplay between the two learning components is crucial for the development of the evolution of an effective, global solution. The gradient component needs to produce classifier utility guesses as fast as possible in order for XCSF to propagate more accurate classifiers effectively. Faster and more reliable estimates avoid misleading signals for the evolutionary component. On the other hand, the evolutionary component needs to evolve better classifier structures as fast as possible for the task at hand. This gives room for representational and operator improvements on the GA side. In Section III we investigate the interplay of these two mechanisms. We show how important an effective representation of conditions, plus fast and accurate parameter estimates are for quick, accurate, and reliable learning.

So far we have seen that XCSF is designed to evolve accurate rules. Generalization is achieved by a continuous generalization pressure [5], [1] due to classifier reproduction in $[M]$ and deletion from $[P]$. Since more-general classifiers match on average more often than the average classifier in $[P]$, they are reproduced more often. Moreover, reproduction in $[M]$ has a niching effect ensuring that only related classifier structures are recombined. Finally, the mechanism also ensures complete coverage of the encountered problem space since overrepresented subspaces undergo more deletions while reproduction is occurrence-based [35].

In sum, the evolutionary mechanism is designed to evolve partitions in which linear approximations are maximally accurate. The gradient descent-based algorithm is responsible for estimating the suitability of the current partitions. Thus, XCS applies a distributed, local search mechanism combining evolutionary techniques with gradient learning techniques to find a global problem solution. As a whole, XCS strives to evolve complete, maximally accurate, and maximally general function approximations represented in its population of classifiers.

Due to this combination of gradient-based and evolutionary-based learning techniques, XCSF is particularly applicable in problem domains in which gradient techniques alone are not guaranteed to converge to an optimal problem solution, or in which problem partitions are required that cannot be shaped by gradient information. In XCS, gradient information is only necessary to yield locally optimal approximations. The development of a suitable space partitioning, which yields a globally optimal solution in conjunction with the local approximations, is the responsibility of the evolutionary learning component.

From a machine learning perspective, XCSF can be compared to iterative clustering mechanisms. In principle, it works similarly to an expectation maximization (EM) algorithm [36]. The expectation step is accomplished by the matching technique in combination with classifier predictions. The maximization step is realized by the genetic algorithm, which depends on accurate parameter estimates. Hereby, not only a pre-defined cluster distance measure is maximized but also the accuracy of classifier predictions. This makes XCSF an iterative clustering mechanism that clusters the input space for the generation of maximally accurate predictions. Performance comparisons with the Neural GAS algorithm [26], [27] in Section V-D.3 confirm that XCSF is able to generate much more accurate and general function approximation representations.

Before we compare XCSF to other function approximation techniques, though, we investigate performance on a set of two dimensional functions and subsequently show how performance can be improved in approximation accuracy and in solution compactness.

E. Performance on 2-D Functions

We now show how XCSF evolves accurate function approximations in exemplary 2-D functions. In the subsequent sections we then show how learning in XCSF can be improved by generating hyperellipsoidal condition structures, enabling efficient search through these structures, and improving the gradient-based approximation mechanism.

We first test unenhanced XCSF on the following functions to illustrate its general approximation performance:

$$f_1(x,y) = \sin(2\pi(x)) + \sin(2\pi(y)); \;\; f_2(x,y) = \sin(2\pi(x+y)); \;\; f_3(x,y) = \sin(4\pi(x+y)). \tag{3}$$

All three functions are continuous and continuously differentiable. The domain of all functions in this paper is $\mathcal{S} = [0,1)^n$. Figure 1 shows the three functions. Function f_1 is a simple sine function, in which each dimension is independent of the others. In functions f_2 and f_3, the dimensions are interdependent. The sine wave lies obliquely in the two dimensions. Function f_3 makes the problem harder since four full sine waves lie in the problem domain.

It is interesting to note that Function f_1 actually resembles a checkerboard problem [37], with variable side lengths in the grid. That is, each dimension contributes to the solution independently. Thus, to reach a certain accuracy, the sine function needs to be suitably partitioned in each dimension, dependent on the curvature of the sine wave. These partitions are independent in each dimension resulting in the checkerboard layout. In functions f_2 and f_3, the partitionings rather resemble uniform, oblique subspaces. Figure 1 indicates curvature-dependent space

partitions. These partitions can be expected to be approximated with any piecewise linear function approximation mechanism - albeit with varying granularity and exactness.

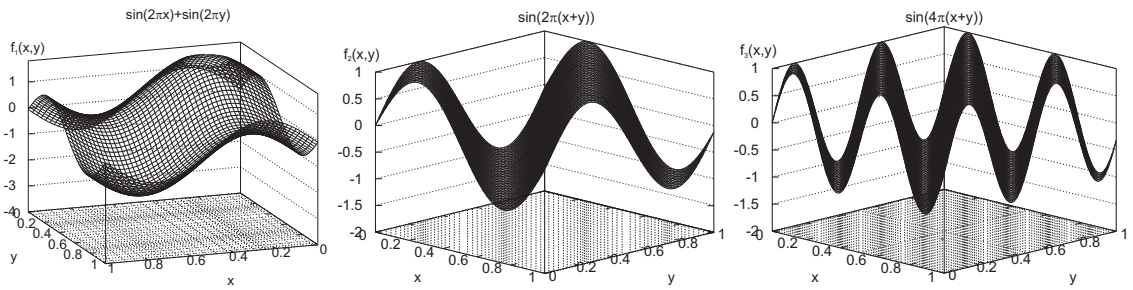


Fig. 1. The axis-parallel sinusoidal function f_1 as well as the axis-diagonal sinusoidal function f_2 appear similarly difficult, whereas the increased curvature in the axis-diagonal sinusoidal function f_3 results in a less accurate approximation, a longer learning effort, and a larger number of distinct classifiers.

Predictive performance of XCSF is shown in Figure 2.[2] The runs start with a fairly general population allowing the genetic pressure and mutation to introduce more specialized offspring. The parameter values are nearly identical to the values chosen in Wilson's work [3]. Function f_1 is learned best, resulting in a low error of about $.02$—slightly above the error threshold $\varepsilon_0 = .01$. The axis-diagonal function f_2 is initially slightly easier to approximate, due to its smaller y-value range. While learning proceeds, the population size rises to a higher level and the prediction again just reaches $.02$. When we double the number of sine waves (function f_3), approximation capabilities strongly break down: accuracy performance does not even get close to the targeted $.01$ accuracy level. Due to the obliqueness of the function, the hyperrectangular conditions make it hard to evolve an effective space partitioning. Also, population sizes do not decrease, indicating that no appropriate generalizations in the space partitions are found.

The learning rate β slightly influences approximation performance further. In the case of larger β values, learning is slightly delayed, most likely due to the higher variance in initial classifier estimates. Later, though, more stable performance can be observed. The larger number of macro classifiers indicates that a larger β value causes more variability in the population.

The results indicate that the hyperrectangular structure does not appear to be suitable for the approximation of smooth functions. Especially when the function lies obliquely in the two dimensions, performance breaks down quickly. Due to the corners of the hyperrectangles, strange overlap effects can be expected to distort the function approximation capability.

[2]All experiments herein are averaged over 20 experiments. If not stated differently, parameters were set as follows: $N = 6400$, $\beta = .1$, $\eta = .5$, $\alpha = 1$, $\varepsilon_0 = .01$, $\nu = 5$, $\theta_{GA} = 50$, $\chi = 1.0$, $\mu = .05$, $r_0 = 1$, $\theta_{del} = 20$, $\delta = 0.1$, $\theta_{sub} = 20$. GA subsumption was applied. Uniform crossover was applied. The error bars and numerical $\pm$ values indicate the respective unbiased standard deviation values.

III. Improving Conditions and Predictions

To eliminate unsuitable overlap effects and consequently evolve smoother approximation surfaces, we now enable the evolution of ellipsoidal condition structures showing their effect on function approximation performance. The resulting general hyperellipsoids do not have the unsuitably overlapping corners of the hyperrectangles, nor do they need to have an axis-parallel orientation. Additionally, we show that the predictions can be accurately estimated faster by using the recursive least squares (RLS) approximation.

A. Axis-Parallel Hyperellipsoids

First, we enable the evolution of axis-parallel hyperellipsoidal structures. Conditions are now represented by

$$C = (\vec{m}, \vec{\sigma}) = ((m_1, ..., m_n)^T, (\sigma_1, ..., \sigma_n)^T), \tag{4}$$

where the column vectors $\vec{m}$ and $\vec{\sigma}$ indicate the center of the ellipsoid and the deviations in the n dimensions, respectively. A Gaussian kernel function is used to determine the current activity of a classifier:

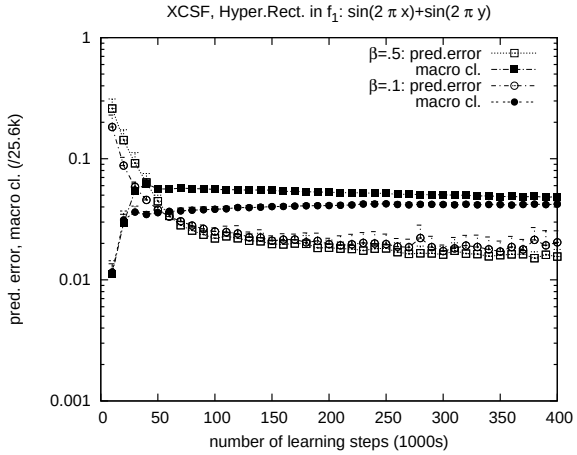
$$cl.act = \exp\left(-\sum_{i=1}^{n} \frac{(x_i - m_i)^2}{2\sigma_i^2}\right), \tag{5}$$

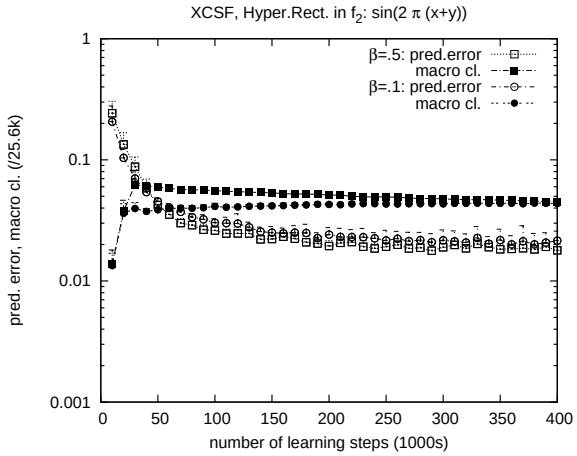
effectively dividing in each dimension the squared distance from the center by twice the variance in that dimension. A classifier is considered part of the current match set if its activity $cl.act$ lies above the threshold θ_m, which is set to .7 throughout the experiments (which corresponds to a radius of $.845$ within which the classifier matches, given all $\sigma_i = 1$). To form classifier predictions, the zero-enhanced lower bound of the condition $\vec{l^*}$ of Equation 1 is replaced with the similarly enhanced center $\vec{m^*}$.

The following other parts of the learning mechanism of XCSF are affected by the changed condition structure representation: (1) the covering mechanism, (2) mutation and crossover, and (3) the subsumption mechanism. Covering sets the center of the condition ($\vec{m}$) to the current instance value ($\vec{x_t}$). Entries of the deviation vector $\vec{\sigma}$ are each chosen independently, uniformly randomly between zero and r_0 (excluding zero). During mutation, each attribute in the condition part is mutated with probability μ. If an attribute of the center is mutated, the new value m_i' is set to a value uniformly randomly chosen in the interval the classifier applies in, that is, $|m_i - m_i'| \leq \sigma_i \sqrt{-2 \ln \theta_m}$. The standard deviation values σ_i are either increased or decreased (equally likely), maximally doubling or halving the values. If any σ_i is larger than the deviation necessary to contain the whole problem dimension, it is set to that value, that is:

$$\forall i : \sigma_i \leq \frac{u_i' - l_i'}{\sqrt{-2 \ln \theta_m}}, \tag{6}$$

where u_i' and l_i' denote the maximum and minimum value of problem dimension i, respectively. Uniform crossover treats all values independently exchanging values with a 50% probability. During subsumption, a classifier is considered more general, if it completely contains the other classifier in all n dimensions considered separately.





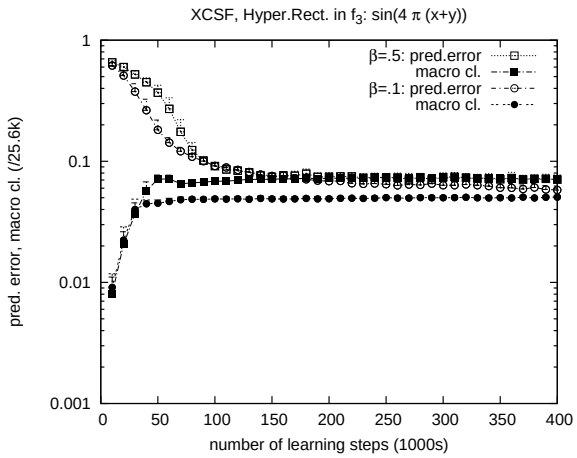
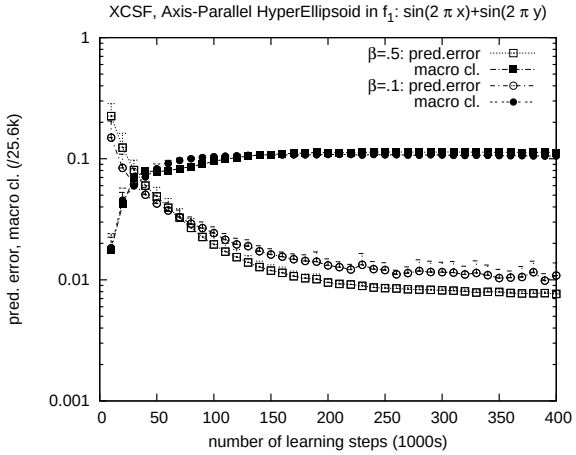
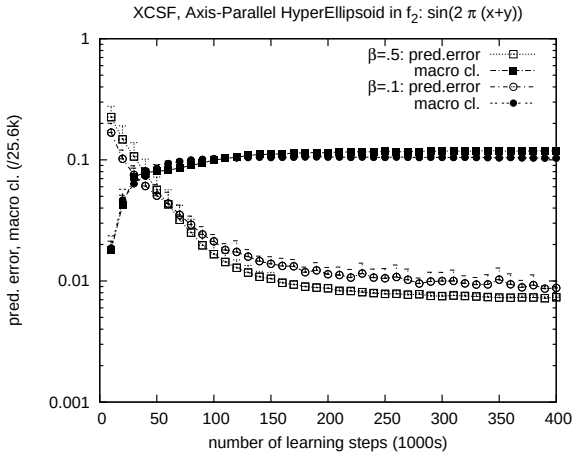


Fig. 2. XCSF is able to learn the three test functions with increasingly less success. The axis-parallel sinusoidal function f_1 as well as the axis-diagonal sinusoidal function f_2 appear similarly difficult whereas the increased curvature in the axis-diagonal sinusoidal function f_3 results in a less accurate approximation, a longer learning effort, and a larger number of distinct classifiers.





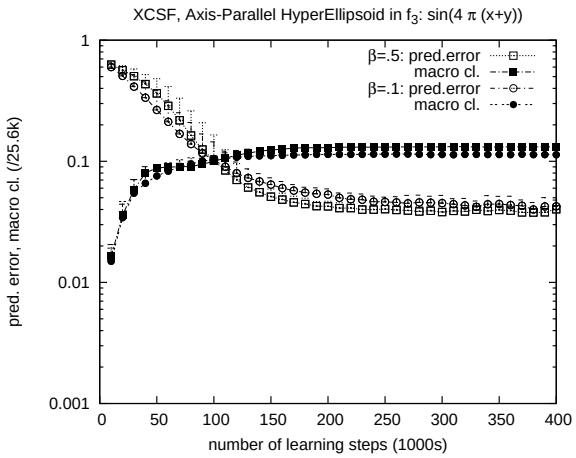


Fig. 3. Hyperellipsoidal conditions alleviate the problem of unsuitable overlaps due to the original hyper-rectangular representation. Higher accuracy is reached faster by XCSF with axis-parallel, hyperellipsoidal condition structures. Nonetheless, in the oblique, strongly curved function f_3, the targeted accuracy level of .01 is still not reached.

Performance on the test functions f_1, f_2, f_3 (Figure 3) shows that the hyperellipsoidal structures clearly outperform the hyperrectangular structures (cf. Figure 2). In the axis-parallel sine function f_1, performance reaches the targeted accuracy level of $.01$. Also in the axis-diagonal functions f_2 and f_3 performance reaches higher accuracy. However, in the harder oblique function f_3, the targeted maximum error of $.01$ still cannot be reached.

Despite the improvements, performance still remains unsatisfactory in function f_3. One big problem of the axis-parallel hyperellipsoidal encoding is that the orientation of the hyperellipsoids cannot account for the obliqueness in the function. Thus, we now proceed and enable the rotation of the developing hyperellipsoid in the input space.

B. General Hyperellipsoids

To enable rotation of the hyperellipsoids, we endow the condition structure with a full transformation matrix that enables stretching and rotation of the evolving hyperellipsoidal structures. A condition is now defined as:

$$C = (\vec{m}, \Sigma) = ((m_1, m_2, ..., m_n)^T, \sigma_{1,1}, \sigma_{1,2}..., \sigma_{n,n-1}\sigma_{n,n}), \tag{7}$$

where $\vec{m}$ denotes the center of the hyperellipsoid and matrix Σ the transformation matrix of the condition. In this way, each condition effectively defines its own space transformation encoding separate Mahalanobis distances [33] in each classifier.

The consequent activity of a classifier is now defined as:

$$cl.act = \exp\left(-\frac{(\vec{x}-\vec{m})^T\Sigma^T\Sigma(\vec{x}-\vec{m})}{2}\right), \tag{8}$$

effectively multiplying the full transformation matrix with the vector difference $\vec{x}-\vec{m}$. The general hyperellipsoid coincides with the previous axis-parallel hyperellipsoid if values are only encoded on the diagonal of the transformation matrix. As before, a classifier matches a given input if its current activity lies above threshold θ_m.

In covering, the center of the hyperellipsoid is set to the current value. Only the diagonal entries in the matrix are initialized to the squared inverse of the uniformly randomly chosen number between zero and r_0. The inverse is chosen to mimic the initial size of the axis-parallel hyperellipsoids. All other matrix entries are set to zero.

Mutation is similarly adjusted in that each matrix entry is mutated separately, maximally decreasing (increasing) the value by 50%. If the value is still zero, it is initialized to a randomly chosen value as in covering for the diagonal matrix entries considering parameter μ_0. The values of the matrix entries are unrestricted. Uniform crossover is applied to all $n + n^2$ condition part values.

To decide if a classifier is contained by another classifier during subsumption, we use an approximation. A classifier is considered more general than a second classifier if its condition part contains the point on the outside surface of the other ellipsoid that lies beyond the midpoint of the other classifier. Figure 4 illustrates several cases.

Performance of XCSF with general hyperellipsoidal conditions in functions f_1, f_2, f_3 is shown in Figure 5. Function approximation is improved in the cases of functions f_2 and f_3 since it is possible to rotate the general

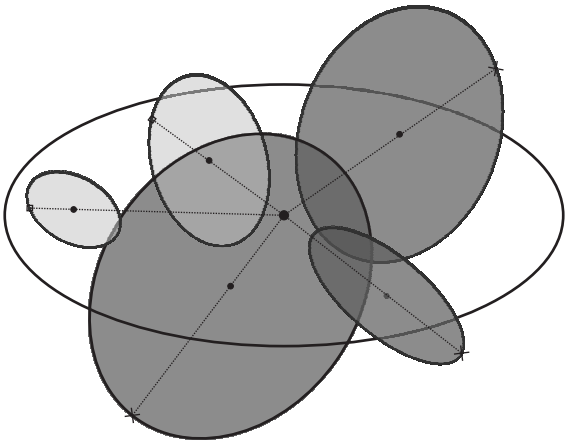


Fig. 4. During subsumption, an ellipsoid A is considered more general than ellipsoid B, if A contains the point at which the elongation from the center of A through the center of B intersects the surface of B. Elongations are indicated by dotted lines. Ellipsoids that can be subsumed by the big white ellipsoid are in light Grey, others in dark Grey.

hyperellipsoid in the input space. Thus, there are less disruptive overlaps and the classifier orientation can suitably rotate to enable even more accurate approximations. However, learning still takes a considerable amount of time. The next sections show the effects of further improved condition approximation and parameter estimation.

C. Explicit Representation of Ellipsoidal Orientation

So far, hyperellipsoids are represented by a transformation matrix Σ, which implicitly encodes stretch and angular orientation of the represented hyperellipsoid. This leads to a redundant encoding of the actual hyperellipsoidal structure. Such redundant encodings have been shown to be beneficial sometimes [38], since the encoding can open up additional paths through the problem space the evolutionary process can exploit. However, as shown in Figure 5, the evolutionary process is still rather slow. The redundant encoding seems to slow down evolutionary progress since (1) mutations of entries in the transformation matrix may cause strong and often misleading changes in the hyperellipsoidal structure, (2) crossover may be disruptive possibly generating two unsuitable ellipsoidal structures out of two currently useful ones, and (3) unnecessary diversity can hinder effective evolutionary progress since reproductive opportunities of successful classifier structures may decrease [4]. Thus, we now enable the more explicit evolution of stretch and angular orientation of the hyperellipsoidal structure in order to speed up the evolutionary progress.

In the case of the axis-parallel representation, we already had an explicit stretch representation of the ellipsoid in each dimension. Thus, as in the previous case, we now again represent stretch with vector $\vec{\sigma} = (\sigma_1, ..., \sigma_n)^T$. To represent rotations properly, we need to represent $\binom{n}{2}$ angles to define unique angular rotations with respect to each axis. Thus, we replace the previous covariance matrix representation of the condition with the stretch vector $\vec{\sigma}$ and a

vector of transformation angles $\vec{\gamma}$ of size $\binom{n}{2}$. The necessary rotation matrices and consequent transformation matrix is then derivable from the angular vector. In 2-D and 3-D, the rotation matrix can be derived directly from the Euler angles [39]. In higher dimensions the rotation matrix can be determined by multiplying together the $\binom{n}{2}$ rotations with respect to each of the hyperplanes. We consequently enable XCSF to explicitly evolve the hyperellipsoidal representation instead of evolving the orientations indirectly within the transformation matrix representation.

The angles are initialized to zero upon covering, essentially starting the learning progress with axis-parallel hyperellipsoidal structures. Mutation alters the angles by a uniform random number from $[-\pi\mu_0, \pi\mu_0]$. The angles are constrained to lie in $(-2\pi, 2\pi]$. The representation still allows redundant encodings: For example, in two dimensions an ellipse with stretch $m_1 = 2$ and $m_2 = .1$ and angle 0 (or 2π) is equivalent to another ellipse with stretch $m_1 = .1$ and $m_2 = 2$ and angle $\pi/2$ (or $-3\pi/2$). Crossover applies to each angular value separately applying uniform crossover.

Figure 6 shows the performance of XCS with rotating hyperellipsoids in the three test functions. Performance improvements occur in the diagonal sinusoidal function f_2 and are even more pronounced in f_3, compared to the runs without explicit rotation (cf. Figure 5). In the axis-parallel sinusoidal function, performance is not affected since rotations are unnecessary in this case. Nonetheless, dependent on the learning rate β applied, accuracy remains rather noisy indicating that the prediction value approximations could be further improved. Thus, we now add the recently introduced RLS approximation technique to XCS parameter estimation.

D. More Accurate Approximations using RLS

Recently, the delta rule update of the prediction part (Equation 1) was replaced by the pseudoinverse method [13] and RLS [40]. RLS is known to yield fast and stable parameter approximations due to the utilization of second-order gradient information [41]. The resulting approximations were shown to yield more suitable linear approximations in XCSF [13] while decreasing the parameter estimation variance [21].

To implement (linear) RLS in XCSF classifiers, a matrix V (of size $(n+1) \times (n+1)$) needs to be added to each classifier. The update of XCSF with RLS is done as follows. Given the current input $\vec{x}$ and the target value y, RLS updates the weight vector $\vec{w}$ by

$$\vec{w} \quad \leftarrow \quad \vec{w} + \vec{k}[(y_t - (\vec{x^*} - \vec{m^*}))^T \vec{w}],$$

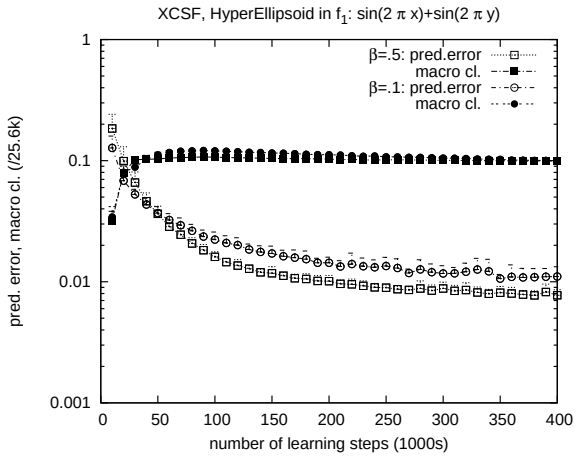
where, $\vec{k}$ is the *gain vector* computed as

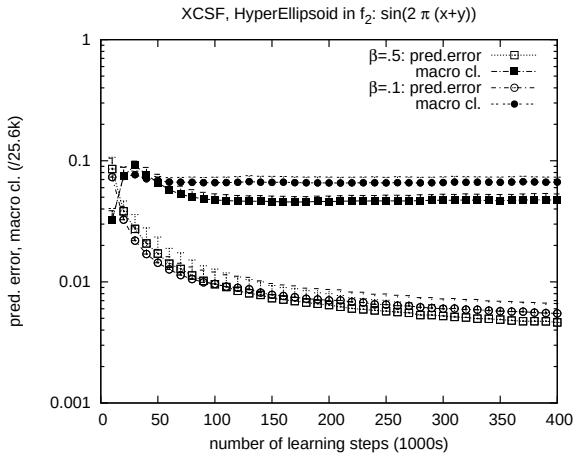
$$\vec{k} \quad = \quad \frac{V^T(\vec{x^*} - \vec{m^*})}{\lambda + (\vec{x^*} - \vec{m^*})^T V^T (\vec{x^*} - \vec{m^*})}, \tag{9}$$

while matrix V is updated recursively by,

$$V^T \quad = \quad \lambda^{-1} \left[I - \vec{k}(\vec{x^*} - \vec{m^*})^T \right] V^T. \tag{10}$$

Parameter λ denotes the forget rate for RLS, where $\lambda = 1$ denotes infinite memory. A value less than one leads to "forgetting" of values in the (distant) past consequently enabling continuous adaptivity but also potentially





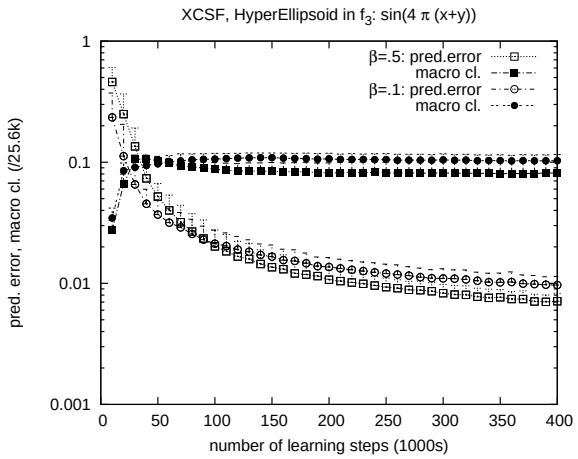
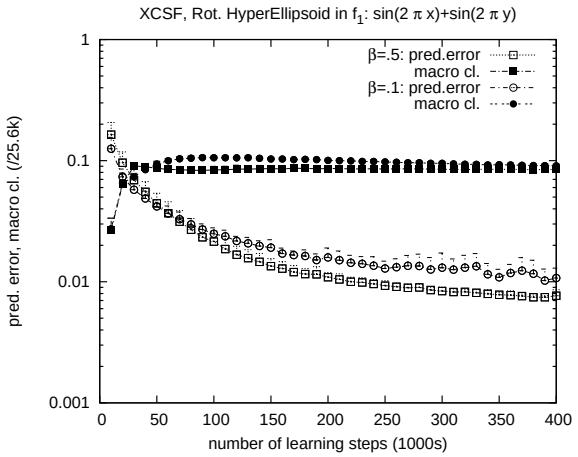
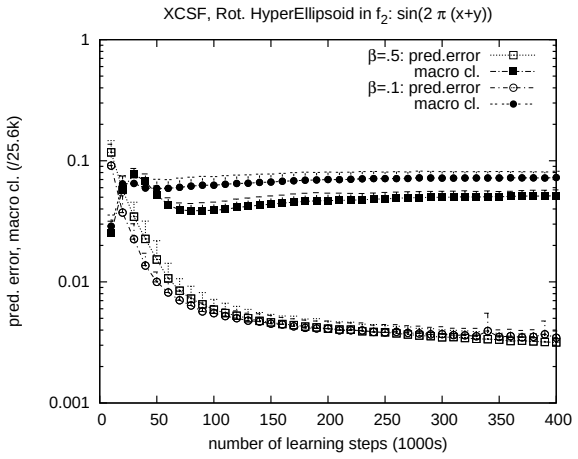


Fig. 5. General hyperellipsoidal conditions further improve performance compared to (restricted) hyperellipsoidal conditions. Also the number of distinct classifiers slightly decreases. Although all runs reached an accuracy level below .01 after 400k learning steps, learning progress still seems rather slow.





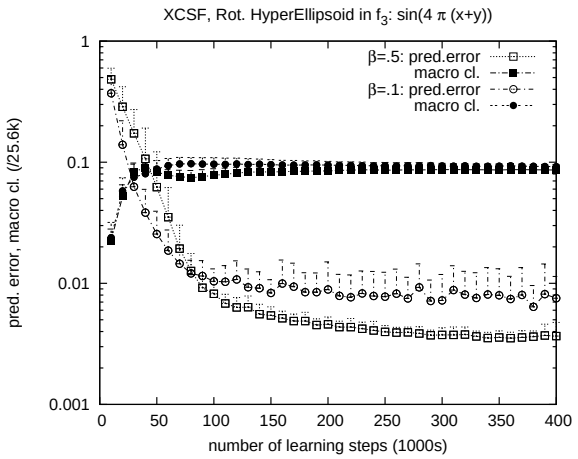


Fig. 6. Quickening the evolutionary process by enabling explicit ellipsoidal rotations speeds up learning progress and final accuracy in functions f_2 and f_3. Performance is still rather noisy, though. Population size does not appear to change compared to runs without explicit rotation.

introducing unintended instabilities [41]. Matrix V (avoiding the often used inverse computation) can be initialized by the scaled identity matrix, as suggested elsewhere [21], [41]:

$$V = \delta_{rls} I, \tag{11}$$

where I is the identity matrix of dimension $n+1$ and $\delta_{rls} > 0$, where larger δ_{rls} values introduce less bias to the initial weight vector $\vec{w}$ (if not stated differently, $\lambda = 1$, $\delta_{rls} = 1000$ in the experiments reported below). Note that RLS is a special case of Kalman filtering for the case of a fixed target state in which no control signal is applied to the state variables (that is, the weight vector $\vec{w}$).

Figure 7 shows the resulting performance on the three test functions. Performance accuracy generally improves in all three functions compared to runs without RLS (cf. Figure 5). This indicates that more accurate predictions are formed independent of the orientation of the function. Also learning speed increases suggesting that RLS is able to deliver suitable prediction estimates faster than the direct gradient-based update mechanism. Figure 7 also compares different initialization values of matrix V during classifier generation (during covering or GA) and different δ_{rls} values. It can be seen that a forget rate of $\lambda = .99$ is advantageous if the matrix diagonal is initialized with $\delta_{rls} = 1$. The small δ_{rls} biases the weight updates towards the initial weight setting. This can be alleviated by the forget rate. However, it can also be prevented by initializing the matrix diagonal with larger values $\delta_{rls} = 1000$. In this case, the forget rate λ has no additional positive influence but actually can cause temporary instabilities (Figure 7 right bottom graph). Nonetheless, in dynamic problems in which the function or concept values change over time, a forget rate λ below 1 might still be advantageous.

IV. Rule-set Compaction

Despite the accurate and reliable performance of XCSF with RLS approximation and rotating hyperellipsoidal representations, the final population sizes of XCSF suggest that the function is overrepresented with highly overlapping classifiers. Thus, we now introduce a new compaction mechanism to XCSF. The mechanism can compact the population of XCSF by over 90% while influencing accuracy of performance only marginally.

A challenge for any form of compaction is that the resulting population may not cover the complete problem space any longer. The determination of uncovered subspaces, however, is a computationally intensive problem. Our compaction mechanism avoids this problem by switching on compaction onset to a *closest classifier matching* (CCM) mechanism, in which a fixed number of closest classifiers match. During compaction, the GA does not apply mutation or crossover any longer, as suggested in Wilson's original condensation approach [1]. Additionally, a greedy algorithm may be applied on compaction onset, which deletes redundant, inaccurate, overlapping classifiers.

A. Closest Classifier Matching

Most classifier systems to date have experimented only with LCS populations in which each classifier matches in a restricted subspace of the search space that is determined in the classifier condition. Booker [42] defined a

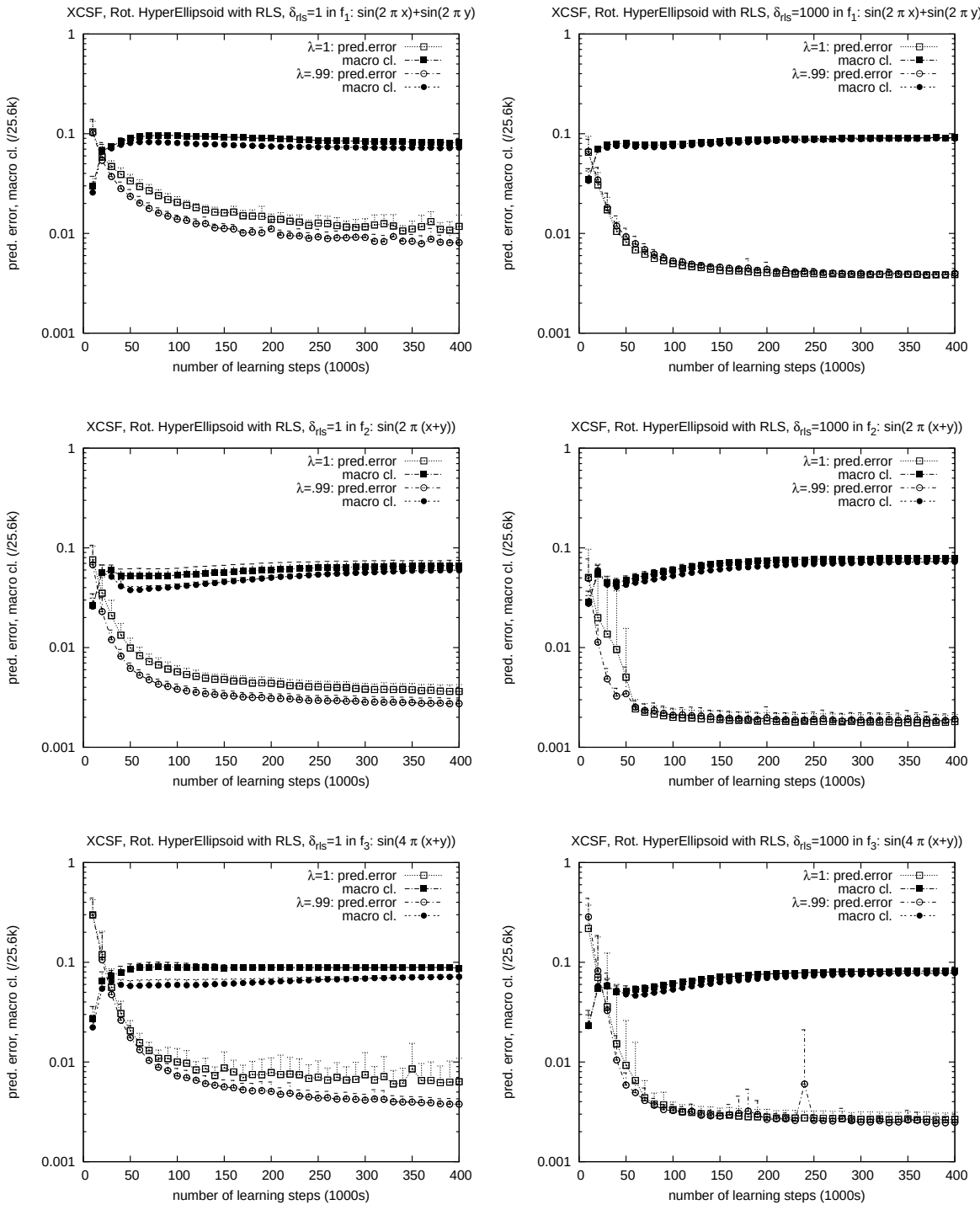


Fig. 7. Learning progress improves further when RLS is applied to optimize the linear prediction of classifiers in all three test functions. The usage of a forget factor $\lambda = .99$ can abolish the bias due the chosen initial weight vector initialization (left-hand side). However, larger initial values in the diagonal of matrix V ($\delta_{rls} = 1000$) abolish the bias as well and enable even more accurate parameter estimations. Hereby, the runs with $\lambda = 1$ (right-hand side) yield the most stable performance.

matching mechanism where a classifier matched a binary input string even when some bits did not actually match. Fuzzy classifier systems have been investigated [43], where classifiers match to self-defined degrees. However, none of these approaches set the classifiers in relation to each other, that is, there was no competition during matching but matching was determined individually for each classifier independent of other classifiers in the population.

Later, Booker [44] designed a matching mechanism that contained a minimum number of matching classifiers. If the number of matching classifiers did not reach that number, additional closest classifiers matched. Our CCM approach takes a similar road, matching the *closest* Θ_M (micro-) classifiers. In this way, it is guaranteed that Θ_M classifiers match the current input, so that the match set size is also Θ_M. Closest is defined in the condition structures of a classifiers, that is, by the activity determination in each classifier, defined in equations (5) and (8) for axis-parallel hyperellipsoids and general hyperellipsoids, respectively.

Thus, given current input vector $\vec{x_t}$, first the activity of each classifier is determined. Next, the Θ_M classifiers with the highest activity are added to the current match set. Note that this can be done in linear time on average so that only a constant amount of additional computation time is needed on average for this step (since determining classifier activity also takes linear time). The consequence is a mosaic-like matching of classifiers with overlapping tiles, which are determined by the distribution of classifiers over the problem space as well as the distance measures of the classifiers.

B. Greedy Compaction Algorithm

Several previous compaction approaches have used heuristics to compact the population quickly and effectively. However, all of these approaches had to take special care to prevent uncovered input regions [24], [25]. Our algorithm can ignore this problem due to the CCM approach and can consequently act more greedily.

The greedy algorithm works as follows:

```
Algorithm Compact XCSF:
1 Iteratively consider all experienced classifiers cl (exp > θ_sub) in [P]
     prioritized on the minimum error
2 Form match sets [M] containing all classifiers that match the center of cl
3 Set numerosity of cl to |[M]| and delete all other classifiers in [M]
```

The motivation of this compaction algorithm is to compact the population while maintaining the general classifier distribution over the problem space. The algorithm iteratively considers the next experienced, least error classifier cl in the population. Next, a match set $[M]$ is formed that contains all classifiers that match the center of the selected classifier, consequently considering all center-overlapping classifiers. All matching classifiers are deleted and their numerosity is transferred to the initially selected classifier cl, since the initially selected classifier is the experienced, least error classifier in that subspace. The algorithm assures that hard problem subspaces, those with strong curvature in the function, remain covered by more classifiers than regions that can be approximated easily.

The runtime of the algorithm is quadratic in population size, since the selection of the next least error classifier in the population and the respective match set formations take time linear in population size. Seeing that the number of learning iterations as well as population size grow polynomially in problem complexity [45], [9], this complexity is not a bottleneck in the learning process, but can rather be neglected, compared to learning time.

Since the shape of the subspace a classifier matches in changes due to the application of CCM upon compaction, XCSF continues adjusting its parameter values to account for the altered subspace that each classifier now matches in. XCSF also continues to apply the GA to balance space coverage. However, after compaction, neither mutation nor crossover are applied any longer, effectively applying the condensation mechanism of Wilson's original work [1], but with CCM ensuring complete problem space coverage.

In sum, rule set compaction is applied after a certain amount of learning steps. First, the compaction algorithm may be applied. From then on, CCM is used with a match set size of Θ_M (set to $\Theta_M = 20$ throughout) and XCSF condensation applies, executing the GA without mutation and crossover operations.

C. Compaction Performance

We tested the compaction mechanism on multiple real-valued functions. The size Θ_M was set to twenty throughout the experiments. In the implementation, we do not enforce Θ_M exactly but have enough individual classifiers match, so that at least Θ_M (micro-) classifiers match. For example, if $\Theta_M = 20$ and the closest classifier has numerosity 16, and the second closest classifier has numerosity 11, both complete classifiers will participate in the match set.

We will now show performance of the compaction algorithm in the three functions considered throughout this paper. The subsequent section further evaluates the compaction algorithm in more challenging functions as well as in higher dimensions.

Figure 8 shows the performance of the compaction mechanism in the three test functions. After iteration $400k$ the compaction mechanism applies changing the matching procedure to CCM and applying condensation (no more mutation nor crossover during the GA application). Without the application of the greedy compaction algorithm (left-hand side graphs of Figure 8), the number of distinct classifiers in the population strongly and continuously decreases while accuracy is affected marginally. In fact, in function f_2 accuracy even increases indicating that the compaction mechanism eliminates inaccurate classifiers allowing for an even more accurate function approximation. Moreover, it can be seen that a forget factor $\lambda = .99$ is slightly advantageous compared to infinite memory ($\lambda = 1$) in the classifier predictions. When changing to CCM, the subspace a classifier matches changes and becomes dynamically dependent on the distribution of surrounding classifiers and their numerosity values. Thus, the optimal linear approximation for the altered subspace is likely to change, dependent on the function values in the altered subspace. Thus, an adaptation rate to this change can be advantageous. Alternatively, also the covariance matrix V may be adjusted by increasing the diagonal entries (initialized by δ_{rls}), which is however not further investigated in this paper.

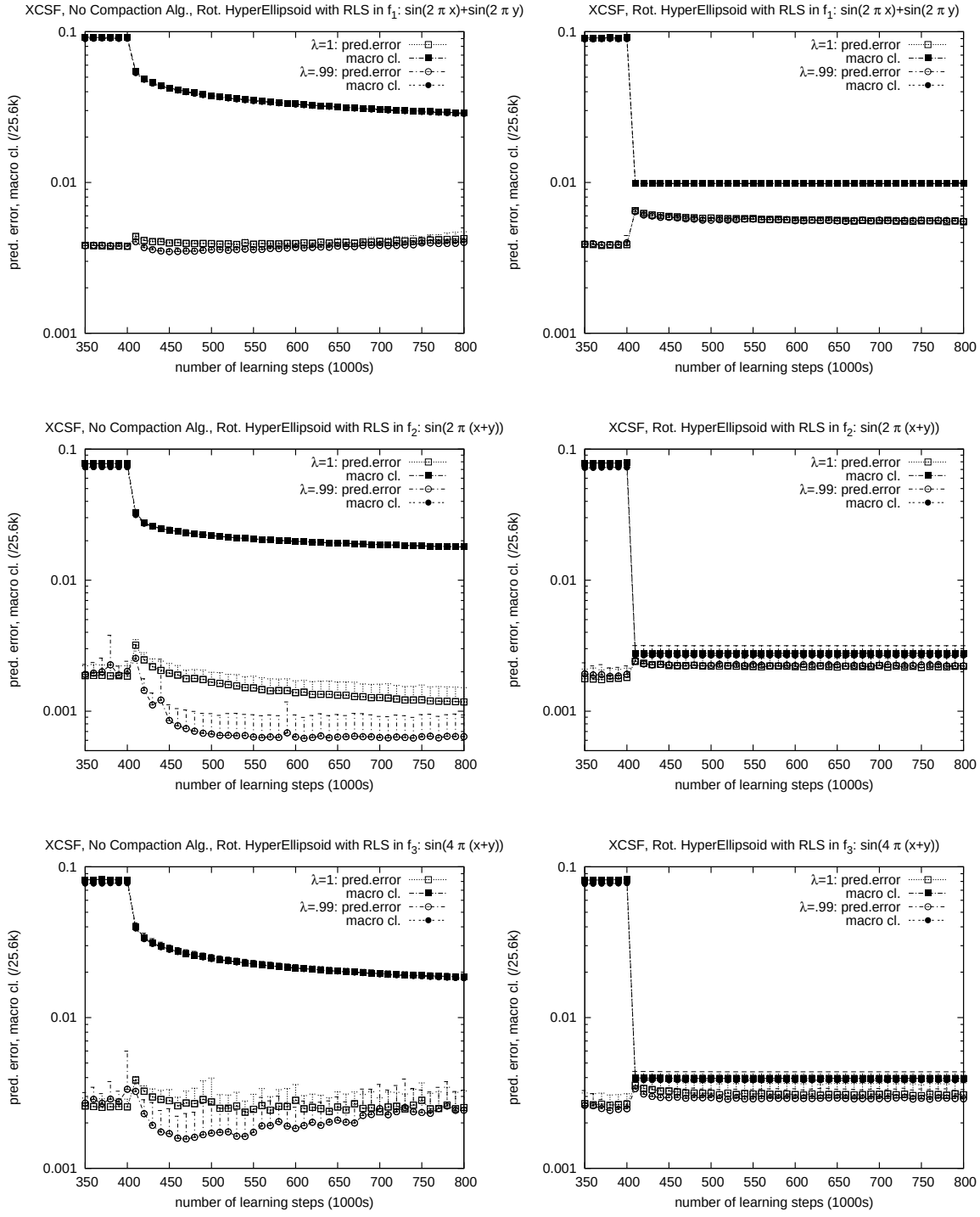


Fig. 8. The compaction mechanism with closest classifier matching and condensation decreases the number of macro classifiers while marginally affecting accuracy (left-hand side). Hereby, the forget factor $\lambda = .99$ shows to respond to the changing approximation subspace due to closest classifier matching faster than without forgetting, especially in functions f_2 and f_3. The application of the greedy algorithm upon the application of compaction decreases the number of macro classifiers even more strongly, still hardly affecting accuracy (right-hand side).

With the application of the greedy compaction algorithm, the number of distinct classifiers shrinks even more strongly (right-hand side graphs of Figure 8). Accuracy slightly drops since the matching algorithm and classifier distribution is changed, which is slightly compensated due to function value estimation adjustments. With the greedy compaction algorithm, the forget factor λ of the RLS algorithm does not show any additional influence on performance accuracy in the three functions. While the error values were $.0039 \pm .0002$, $.0018 \pm .0003$, and $.0026 \pm .0004$ for the three functions, respectively, before compaction was applied (setting $\lambda = 1$), they were at $.0056 \pm .0002$, $.0022 \pm .0004$, and $.0030 \pm .0006$, after $200k$ further iterations. Thus, accuracy was slightly affected. On the other hand, population sizes decreased highly significantly: Before compaction, the sizes of distinct (macro-) classifiers were 2344 ± 52, 2011 ± 80, and 2106 ± 46, respectively, and settled at 251 ± 7, 71 ± 10, and 101 ± 11 after compaction—a drop of more than 90% on average. It is also apparent that the higher regularity in functions f_2 and f_3 is detected appropriately, resulting in more compact function representations than in the case of f_1, in which less regularity can be exploited. The results confirm the robustness of the greedy compaction algorithm as well as the CCM mechanism maintaining accurate approximations while strongly decreasing population sizes.

V. Performance Evaluations

We now evaluate XCSF's performance on higher dimensional function problems as well as in other function domains. In each case, we analyze achieved accuracy and final population compaction. Finally, we compare XCSF's performance with statistics-based approximation approaches [28], [29], compared to which XCSF shows competitive performance, as well as with the self-organizing neural network approach Neural GAS [26], [27], which XCSF outperforms easily. We also evaluate XCSF's generalization capabilities by restricting the input space to a subset of sampled function values.

A. Performance on 3D Functions

Before moving on to other functions, we test XCSF on f_1, f_2, and f_3 in the three dimensional setting. Performance is shown in Figure 9.

In the case of f_1, the function becomes significantly more difficult. As predicted, the checkerboard quality of f_1 discussed above enforces a grid-like partitioning of the search space for maximally suitable approximations. In conjunction with the piecewise linear approximation in each classifier, the function consequently becomes exponentially more difficult with each additional dimension. Thus, regardless with which setting, XCSF does not reach an accuracy of $.01$ anymore and no convergence is observable in the population.

Compaction affects performance slightly if no proper convergence was reached in all three functions. Especially when no explicit rotations are enabled but the full transformation matrix is evolved (left-hand side graphs of Figure 9), accuracy drops upon the application of the compaction mechanism. However, with RLS and rotating hyperellipsoids this effect is marginalized. In function f_3, for example, compaction decreases population size from

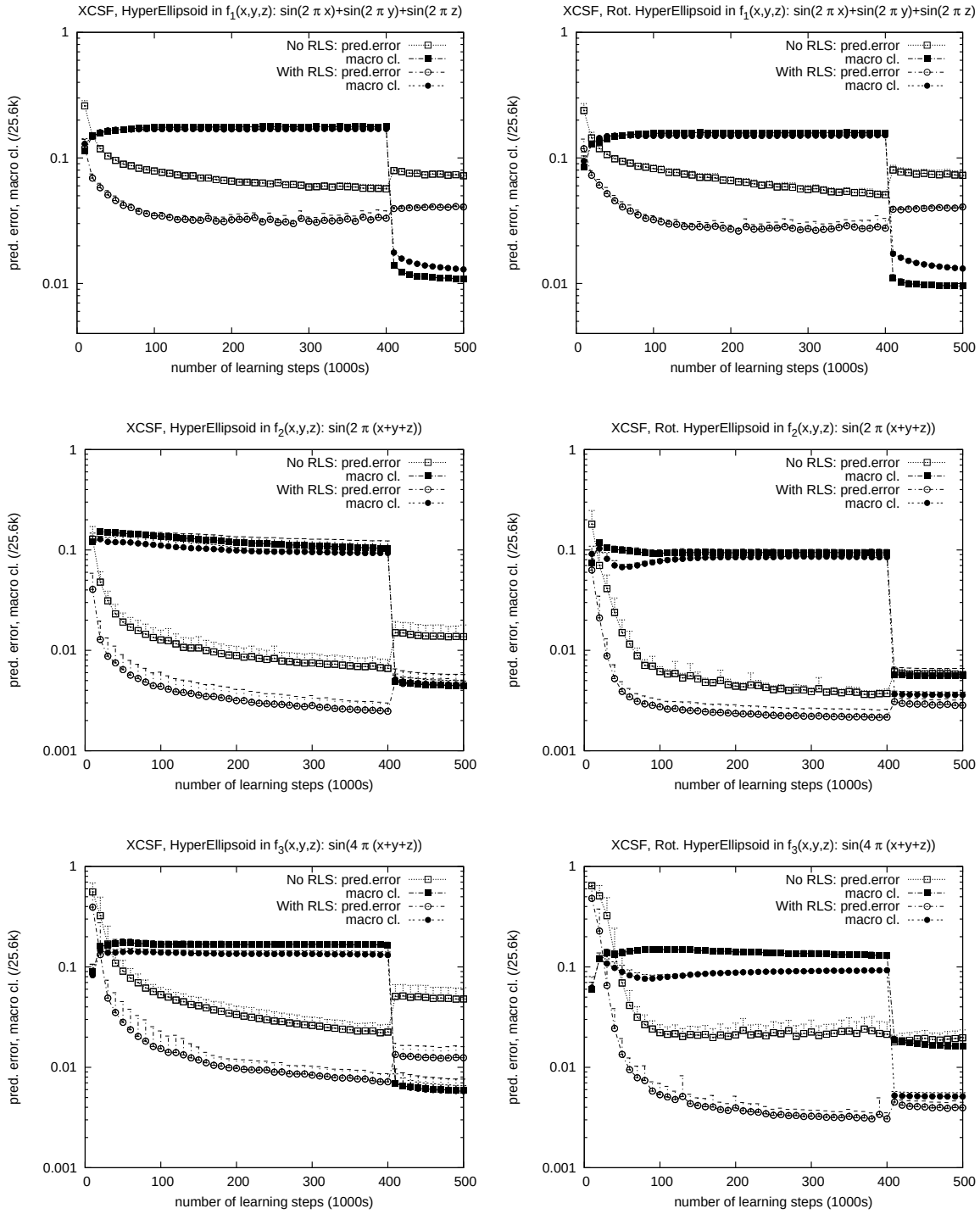


Fig. 9. The checkerboard quality of f_1 prevents XCSF from reaching a .01 error level in three dimensions. In functions f_2 and f_3, though, XCSF continues to learn accurately. As observed in the two dimensional case, RLS yields accuracy improvements in all three functions. The explicit rotation of hyperellipsoids (instead of evaluating the full covariance matrix) is advantageous in functions in which rotations are necessary (f_2 and f_3). Compaction strongly decreases population size while performance is only slightly affected, especially when the function was learned effectively.

2357 ± 70 to 131 ± 12 (measuring at learning step $390k$ and $500k$), affecting accuracy hardly at all ($.00392 \pm .00060$ vs. $.00396 \pm .00053$). This confirms that the evolutionary process of XCSF is able to identify the oblique function property and orient the evolving hyperellipsoids accordingly.

To further investigate the power of the approach, we tested XCSF with rotating hyperellipsoids in several other three dimensional functions:

$$
\begin{aligned}
f_4(x,y,z) &= \sin(8\pi(x+y+z)), \\
f_5(x,y,z) &= \sin(2\pi(x+y+z)) + \sin(4\pi(x+y+z)) + \sin(6\pi(x+y+z)) + \sin(8\pi(x+y+z)), \\
f_6(x,y,z) &= |\sin(2\pi(x+y+z))| + |\cos(2\pi(x+y+z))|, \\
f_7(x,y,z) &= \sin(2\pi(x+y+\sin(\pi z)))
\end{aligned}
$$

The first two functions make approximation even more difficult than in f_3 due to stronger curvature. Functions f_6 and f_7 are non-continuously differentiable and continuously changing in their obliqueness, respectively.

Performance on these four functions is shown in Figure 10. XCSF shows the typical learning pattern in f_4, which is more difficult than f_3, but generally of the same structural requirements (classifiers should be distributed approximately double as dense for equal accuracy). Performance improvement is more noisy and takes longer on average compared to performance on f_3. After compaction, performance on f_3 maintains an error level of $.0040 \pm .0005$ with only 131 ± 12 macro classifiers. On the other hand, in f_4, performance levels out at $.0054 \pm .0008$ with a macro classifier size of 256 ± 29, which clearly indicates the doubled complexity in the function.

Function f_5 is a superimposed sine function. Performance on this function is similar to f_4. Although the function has higher values in its second derivative, it allows initial faster learning due to the lower absolute value differences. Upon convergence, though, the error stays higher than in the f_4 runs ($.0065 \pm .0013$ with size 242 ± 22). In the non-differentiable case f_6, XCSF is still able to generalize maintaining high accuracy ($.0041 \pm .0006$ with size 162 ± 39). In f_7, the function does not lie perfectly obliquely in the problem space any-longer but obliqueness changes gradually. Consequently, the hyperellipsoidal orientations need to be locally optimized and the overlaps are not as clear-cut as before. Nonetheless, performance still reaches the $.01$ level with an error of $.0071 \pm .0016$ and a number of macro classifiers of 5100 ± 119 before compaction, and error $.0085 \pm .0011$ and number 407 ± 20 afterward. Compaction increases the approximation error most in function f_6, which appears to be due to the non-differentiability of the function at several locations. The compaction algorithm overgeneralizes at these points over- or under-estimating the slope of the function.

B. Higher Dimensions

After the evaluation of the general capabilities of XCSF with compaction, we now test the scalability of XCSF and the associated rotation mutation in higher dimensions.

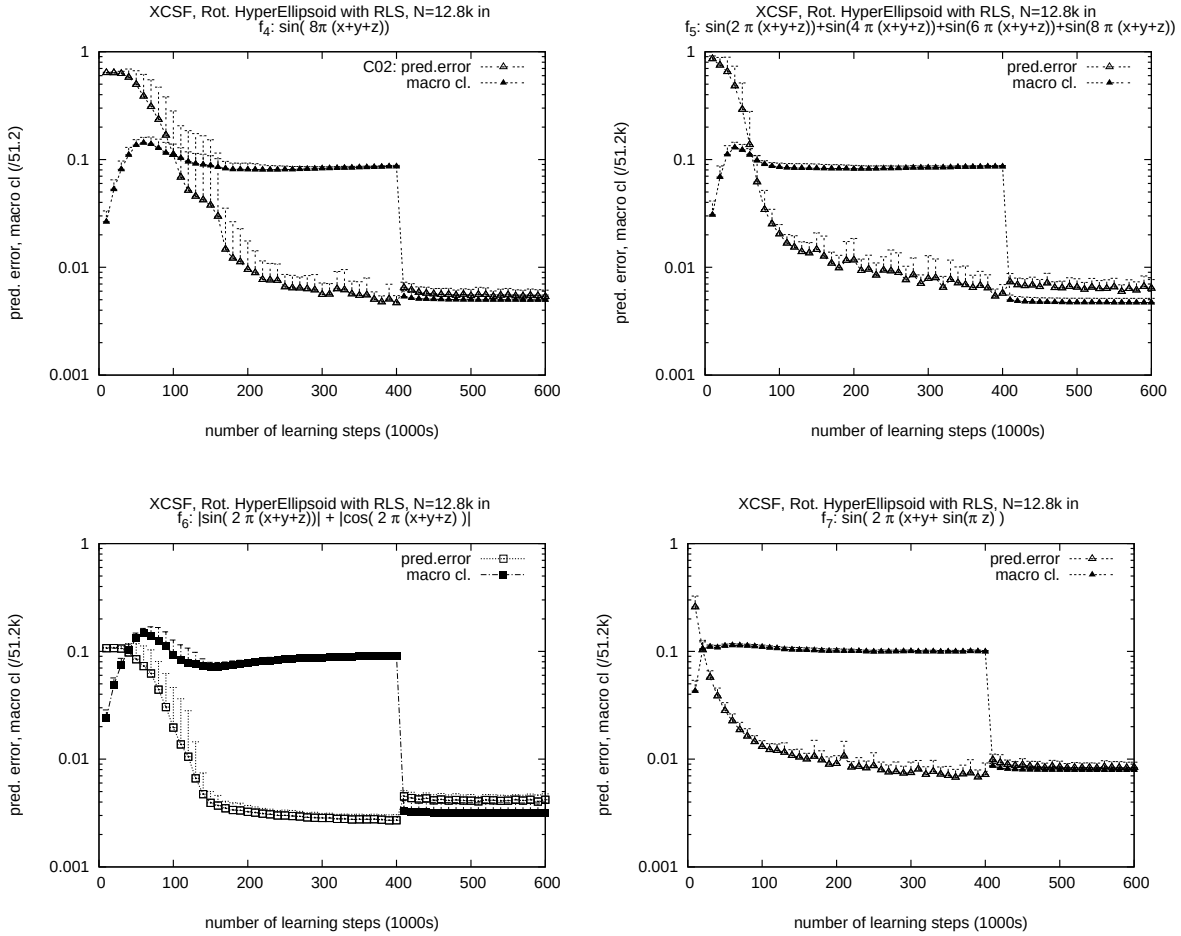


Fig. 10. Performance on other three dimensional functions confirms the robustness of XCSF learning and compaction mechanisms.

To evaluate the suitability of explicit rotations via mutation, we tested XCSF on f_2 and f_3 in four dimensions. Figure 11 shows that XCSF without explicitly rotating hyperellipsoids does not reach the accuracy level that XCSF with rotation reaches. Also the number of distinct macro classifiers in the population stays much higher, indicating a lower amount of convergence and higher diversity in the population. After compaction, the difference becomes even more pronounced: In function f_3, XCSF with rotation hardly looses accuracy ($.0046 \pm .0015$ before vs. $.0051 \pm .0010$ after compaction) while eliminating more than 90% of its classifiers (2504 ± 83 vs. 164 ± 13). Without rotation, XCSF exhibits higher variance and a loss of accuracy after compaction ($.0272 \pm .0169$ vs. $.0515 \pm .0207$). This confirms that the more direct mutation via rotations facilitates population convergence and the development of more suitable classifier hyperellipsoid orientations.

With a successful approximation of a four dimensional function in hand, we experimented with the scalability of our approach. Increasing the dimensions makes the problem progressively harder for XCSF. First, to cover the

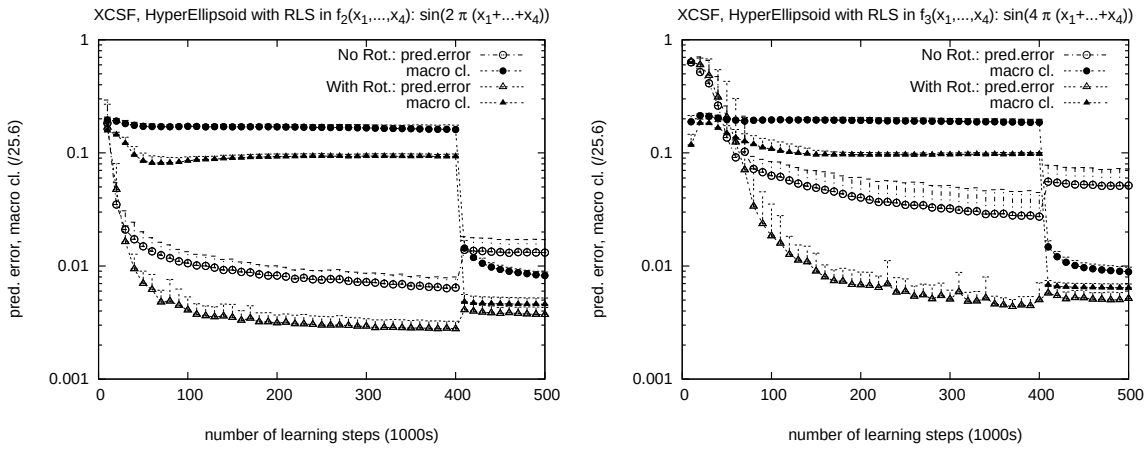


Fig. 11. Also in four dimensions, explicit rotation of hyperellipsoids via mutation facilitates learning in functions f_2 and f_3. If the hyperellipsoids are located suitably, compaction does not disrupt but generates an accurate, highly compact final solution.

whole problem space, the number of necessary classifiers increases exponentially with the dimensions given the average size covered in each dimension in a classifier remains constant. If the coverage of the initial classifiers (generated by covering) is increased, though, then important problem structures may not be detected due to the lack of fitness pressure [5], [9], [37]. Thus, in order to be able to solve problems in increasingly higher dimensional spaces, a good balance between full problem space coverage and classifier specificity needs to be found.

Figure 12 shows performance curves in five, six, and seven dimension versions of function f_3. Population sizes were set to $12.8k$, $25.6k$, and $51.2k$, respectively. To decrease the variability of the classifiers in the initial population, in the six (seven) dimensional runs during covering classifier intervals were initialized with a minimum value of $.4$ ($.5$), that is, $\sigma_{ii} = 6.25 = .4^{-2}$ ($\sigma_{ii} = 4 = .5^{-2}$), and a maximum value of $.5$ ($.6$), respectively. The necessary increase in maximum population size for the higher dimensions indicates the difficulty of the algorithm to detect interesting structures in the function while maintaining full problem space coverage. In the seven dimensional case, a classifier with an initial average spread of 10% in each dimension only covers $.1^7$ of the area, suggesting the need for 10^7 classifiers of that size to cover the whole problem space uniformly. XCSF manages to receive a signal with a much smaller population size and consequently larger classifiers.

To ensure the coverage of a subspace with 99% probability (see the *covering bound* [5]), given a population size of $51.2k$, each classifier should cover at least a subspace of size $1 - (1 - .99)^{1/51,200} = 8.99 * 10^{-5}$. In seven dimensions this corresponds to a square of $.264$ units in each dimension. The chosen maximum stretch of $\sigma_{ii} = .6^{-2} = 2.78$ corresponds to a distance of $\sqrt{-2 \ln \theta_m / l} / \sigma_{ii} = \sqrt{-2 \ln .7/7} / 2.78 = .1149$ in each dimension. Thus, the initialization slightly overspecializes. The specialized initialization enables the detection of the oblique orientation of the approximated function, though. The high variance in these higher dimensional cases shows the difficulty (cf. Figure 12): the detection and growth of the relevant problem structure starts to be extended in time

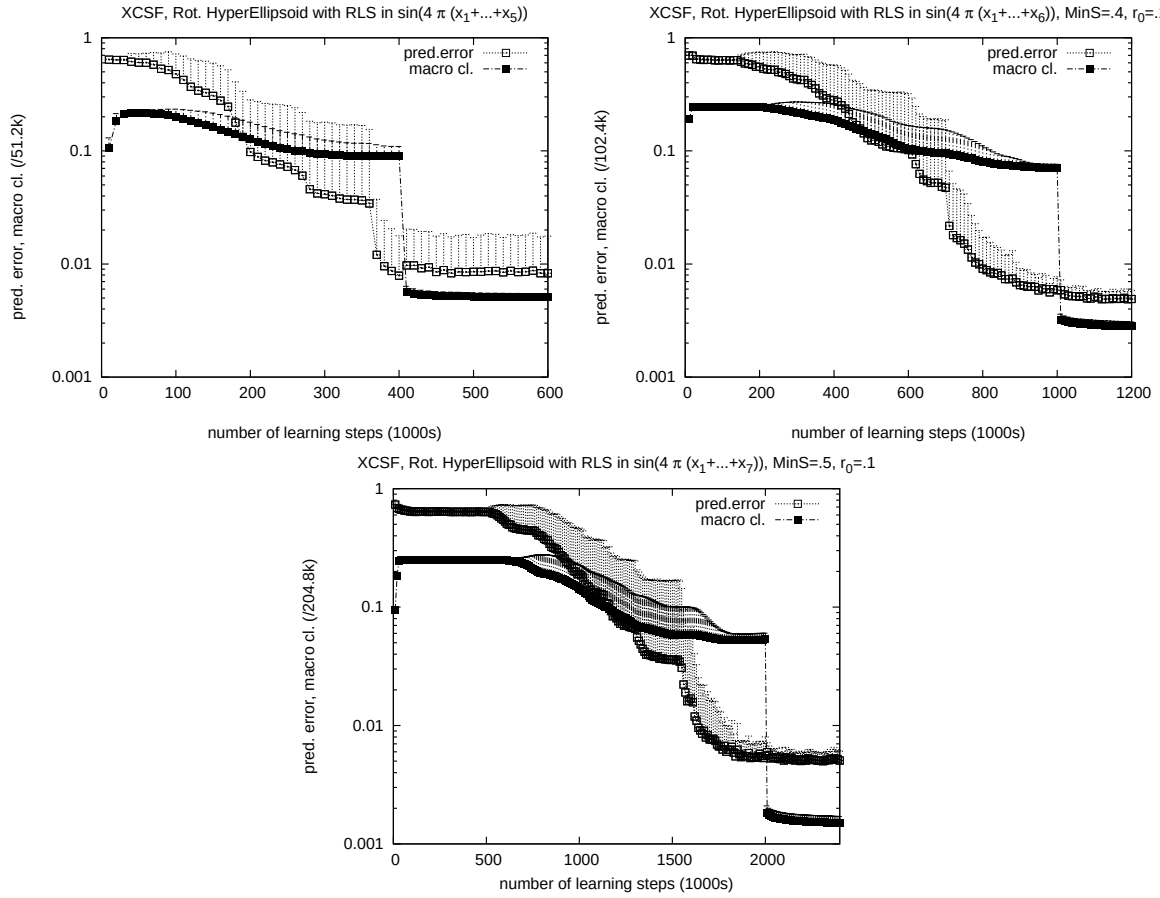


Fig. 12. XCSF can solve function f_3 in up to seven dimensions. Covering and reproductive opportunity challenges become increasingly pronounced due to the curse of dimensionality. Nonetheless, XCSF with compaction still generates a highly compact function approximation representation.

(indicated by the step-like convergence averaged over twenty problems): Each experiment begins to converge at a different point in time, resulting in the high variance during convergence. Nonetheless, XCSF is able to approximate even the seven dimensional version of the oblique function f_3 highly accurately, reaching an accuracy level of $.0051 \pm .00204$ with only 312 ± 42 distinct classifiers after compaction ($.0053 \pm .0020$ with 10898 ± 1414 classifiers before compaction).

C. Generalization Capabilities

Besides larger dimensions and other functions, it is interesting to evaluate the generalization capabilities of XCSF with and without compaction. Thus, we tested XCSF in functions f_1, f_2, and f_3 on a restricted training set of 500 uniformly randomly selected data points of the functions. The training set is continuously sampled in epochs of 500 learning iterations without replacement, as has been done elsewhere [28]. Performance, however, is tested on

a 41×41 grid distributed uniformly over the function domain. To allow comparisons with results of other systems available in the literature [28], performance now is measured by the normalized mean square error (nMSE), that is, the MSE divided by the sample variance of the $1,681$ test function values. Since restricted sampling emphasizes the importance of generalization, the fitness pressure was decreased by setting the tournament size proportion to $\tau = .1$.

Figure 13 shows that XCSF can accurately approximate the three functions with an nMSE of $.0009 \pm .0004$, $.00015 \pm .00013$, and $.0027 \pm .0034$, respectively, before compaction. This confirms the accurate generalization capabilities of XCSF. It can be seen that the application of a forget rate of $\lambda = .99$ does not improve accuracy but rather causes performance instabilities. These instabilities also apply during compaction. Compaction in general causes a slight accuracy decrease in this case, most likely due to under-sampled problem areas. After 100k iterations with CCM and condensation performance nMSE with $\lambda = 1$ and application of the greedy compaction algorithm (performance without compaction algorithm in brackets) was $.0024 \pm .0022$ ($.0015 \pm .0028$), $.0012 \pm .0024$ ($.0003 \pm .0009$), and $.0113 \pm .0126$ ($.0024 \pm .0025$) in the three functions, respectively. Nonetheless, population size again dropped in the runs with $\lambda = 1$ and greedy compaction algorithm (without compaction algorithm): from 1599 ± 68 to 135 ± 5 (1586 ± 76 to 323 ± 8), from 1524 ± 75 to 68 ± 9 (1576 ± 75 to 286 ± 9), and from 1401 ± 82 to 86 ± 11 (1418 ± 76 to 301 ± 10) macro classifiers in the three functions, respectively. Thus, without the greedy compaction algorithm, performance accuracy is only slightly affected while population size decreases by approximately 80%. With the compaction algorithm, performance degrades stronger but population size is decreased by over 90%.

D. Comparison with Other Approaches

As a final evaluation criterion, we chose to compare the performance of XCSF with several other non-evolutionary learning approaches.

1) Constructive Incremental Learning Approach: The constructive incremental learning approach learns a solution similar to the one evolved by XCSF using heuristics-based statistics [28]. The resulting greedy learning algorithm was tested on the following function:

$$f_8(x, y) = max\left\{e^{-10x^2}, e^{-50y^2}, 1.25e^{-5(x^2+y^2)}\right\} + N(0, .01), \tag{12}$$

where the last term denotes Gaussian noise with a standard deviation of $.01$. [28] reports performance values in the setting with a restricted training set of 500 points, drawn uniformly randomly from $[-1, 1]$ in both dimensions and sampled without replacement over many epochs. Performance is tested in a 41×41 grid, which is distributed uniformly over the function domain. A normalized mean squared error of $.02$ is reported. Besides the required approximation due to the restricted training set, the noise coefficient in the function requires sufficient noise tolerance of the algorithm.

We tested XCSF performance on function f_8 with unrestricted problem sampling and with a restricted set of 500 points, as in [28]. Again, the tournament size proportion was set to $\tau = .1$ in the restricted set runs. Performance

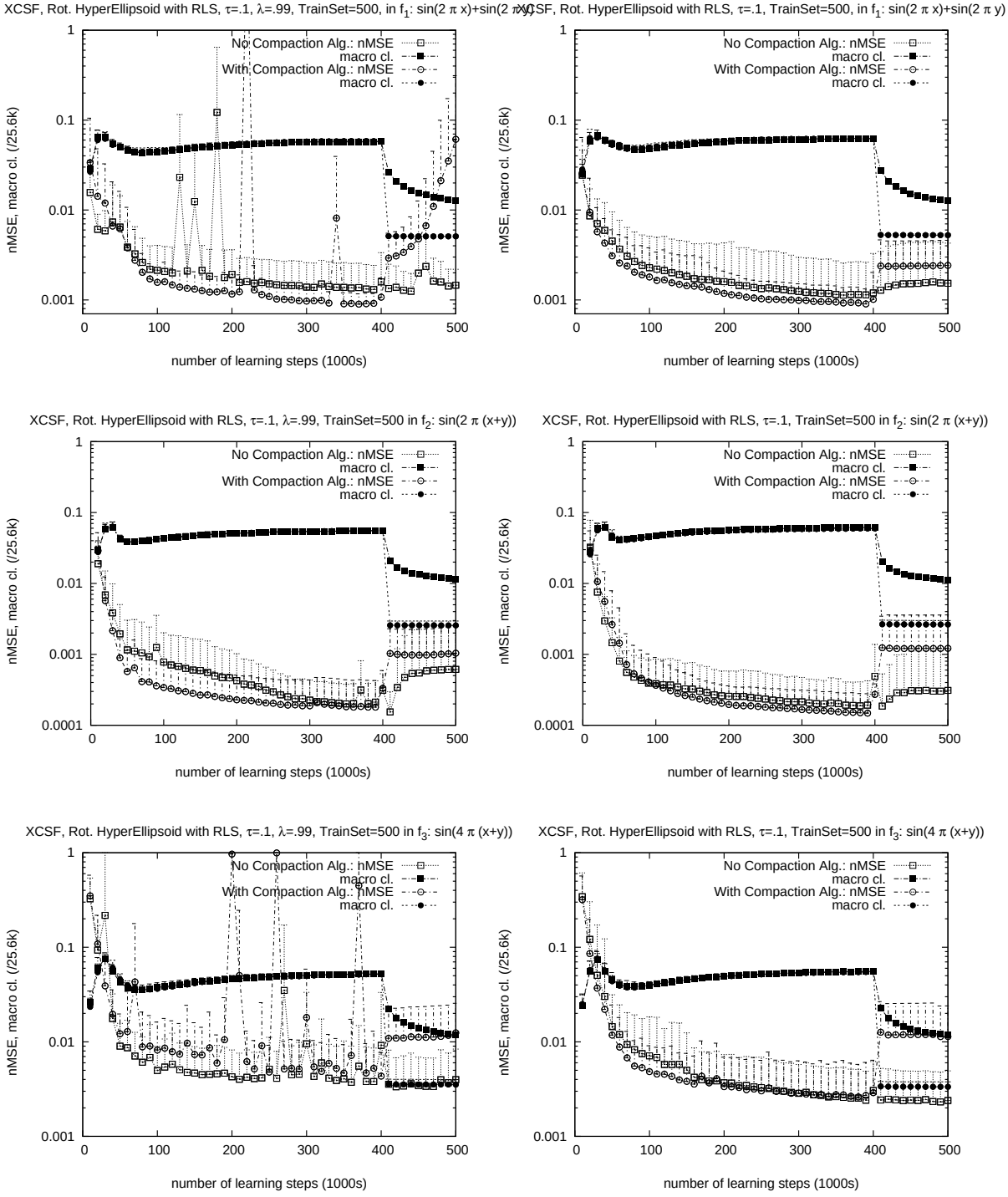


Fig. 13. Performance on f_1, f_2, and f_3 tested on restricted training sets of 500 uniformly randomly sampled points confirms the generalization capabilities of XCSF. A forget rate of $\lambda = .99$ is not beneficial for prediction accuracy stability (left-hand side) but can even result in performance degradation during compaction. Performance is more stable with an infinite memory setting ($\lambda = 1$, right-hand side), also during compaction. The application of the greedy compaction algorithm additional to CCM and condensation yields a more compact classifier set but also some additional loss in performance accuracy.

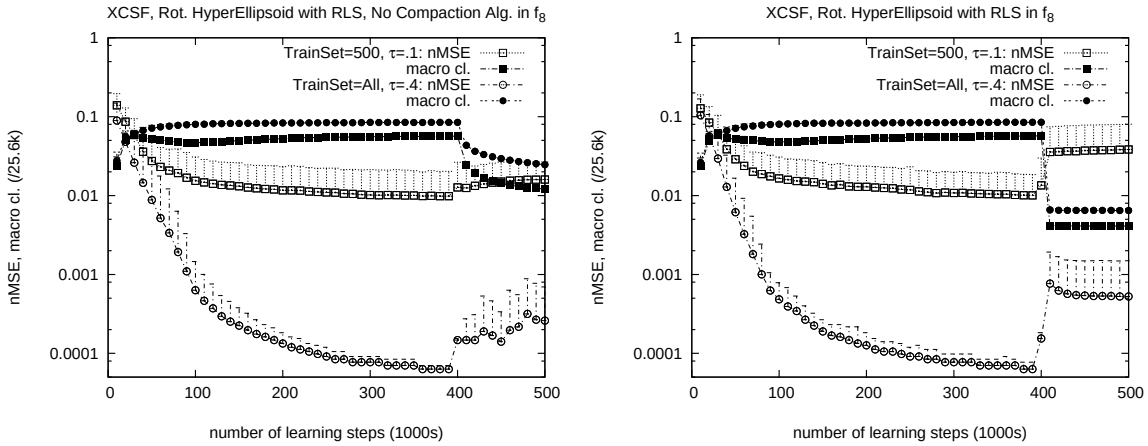


Fig. 14. The Schaal-Atkeson function is an interesting challenge for XCSF. When the whole problem space is sampled uniformly randomly, XCSF strongly outperforms the reported performance in [28]. The compaction algorithm appears to delete some partially useful classifiers, seeing that accuracy slightly decreases (right-hand side).

is reported in Figure 14, plotting the nMSE as done in [28]. When the whole problem space is sampled uniformly during learning, high accuracy is reached with an nMSE of $6.32 * 10^{-5} \pm .70 * 10^{-5}$, slightly degrading to $25.96 * 10^{-5} \pm 1.40 * 10^{-5}$ ($52.63 * 10^{-5} \pm 96.84 * 10^{-5}$) after the application of the compaction mechanism without (with) greedy compaction algorithm. This indicates that compaction causes slightly unsuitable generalization patterns on the approximation surface. The results, nonetheless, confirm that XCSF is able to reliably approximate the function despite the additional noise.

XCSF reaches a performance level of $.0098 \pm .0104$ (1448 ± 79 macro classifiers) before and $.0160 \pm .0108$ (310 ± 7) after 100k iterations with compaction mechanism but without greedy algorithm, when the set of sampled points is restricted to 500 continuously sampled data points (Figure 14 left-hand side, TrainSet=500). When the compaction algorithm was applied as well, accuracy of performance changed from $.0100 \pm .0084$ (1463 ± 74) to $.0384 \pm .0419$ (105 ± 7) (Figure 14 right-hand side, TrainSet=500). Due to slightly unsuitable generalizations, the additional decrease in the number of distinct classifiers led to a more severe error increase. Thus, in restrictedly sampled domains, the compaction algorithm can cause unsuitable generalizations. Nonetheless, using only CCM plus condensation, performance can be nearly maintained, while decreasing the number of distinct classifiers still by approximately 80%.

2) Incremental Linear Model Tree Algorithm: Function f_8 was also used in [29] with stronger Gaussian noise $N(0, .1)$ sampling from the complete problem space uniformly randomly, reporting normalized root mean square errors (nRMSE). Performance of XCSF is shown in Figure 15. When sampling the full problem, performance reaches a level of $.00140 \pm .00013$nMSE, which corresponds to $.0375$nRMSE. After condensation, approximation quality remains stable at $.00141 \pm .00028$nMSE ($= .0376$nRMSE) without the application of the compaction algorithm and

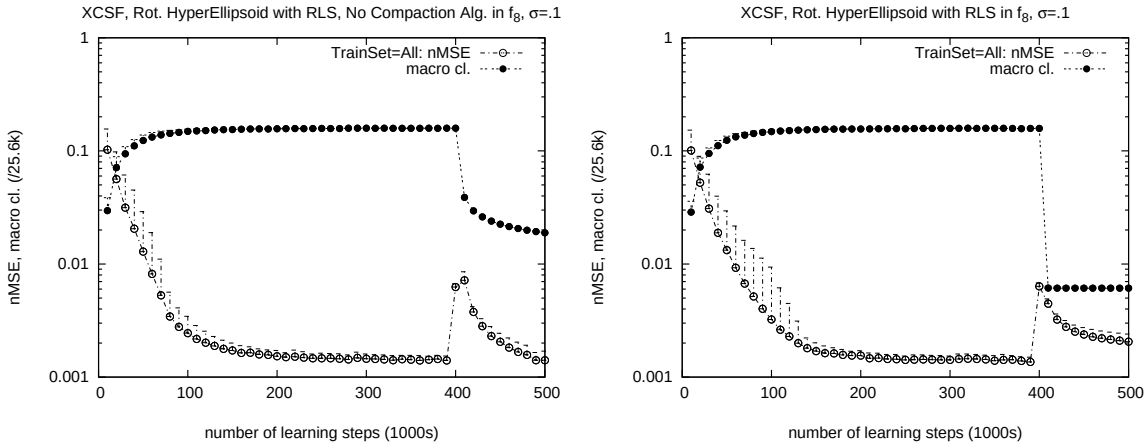


Fig. 15. With the higher noise level of $\sigma = .1$, XCSF is able to approximate Function f_8 yielding a higher accuracy than other learning approaches.

slightly decreases to $.00206 \pm .00034$nMSE ($= .0453$nRMSE) with compaction algorithm. Population size again dropped significantly in both cases: 4059 ± 58 classifiers boil down to 482 ± 8 without and 156 ± 6 with compaction algorithm.

Potts [29] reports a value of $.05$nRMSE for his incremental learning linear model tree algorithm with pruning (IMTI). He reports worse performances for Schaal and Atkeson's incremental learning approach ($.06$nRMSE) as well as for the ten nearest neighbor approach, which reaches a level of only $.08$nRMSE. The number of distinct models used are 68 ± 6 for IMTI and 92 ± 3 for Schaal and Atkeson's incremental learning approach [29]. Thus, XCSF is able to approximate the function more accurately requiring only a slightly higher number of models in its final population.

3) Neural GAS: As a final comparison, we chose the Neural GAS algorithm, which can also be applied to function approximation [26], [27]. Neural GAS is similar to XCSF in that it distributes its neurons based on occurrence frequency and approximates function values locally. However, the distance measure is not altered in the Neural GAS algorithm and the distribution is not dependent on the accuracy of the resulting function value approximation. Thus, Neural GAS performance can be considered a base line performance, which should be beaten by any algorithm that distributes its local approximations prediction error dependently, given such an additional distribution bias is useful in the considered approximation problem.

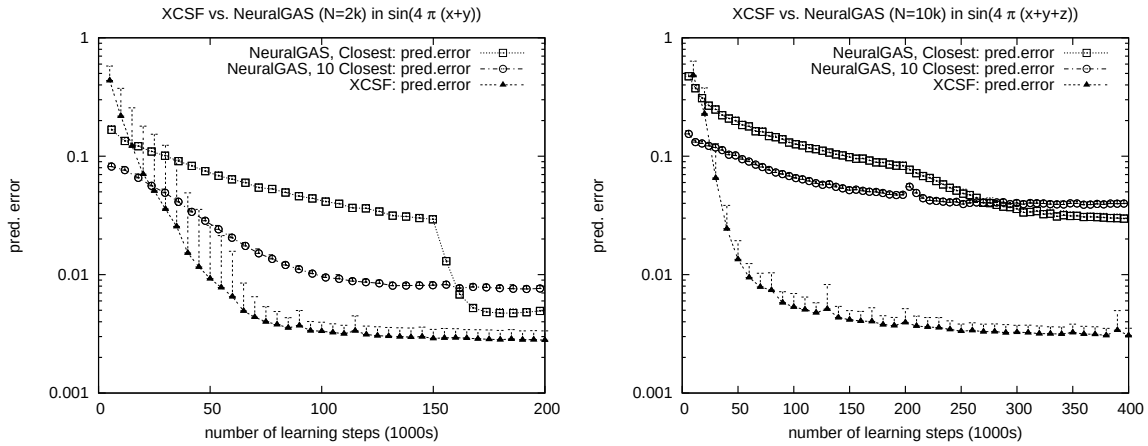


Fig. 16. XCSF clearly outperforms Neural GAS in function f_3 in the two and more pronouncedly in the three dimensional case.

Figure 16 compares performance of XCSF and Neural GAS on function f_3 in two and three dimensions.[3] In the two dimensional case, Neural GAS reaches an accuracy level of $.0049 \pm .00032$ ($.0075 \pm .00050$) with a neural population size of 2000 when the (ten) closest neuron(s) are used for the generation of the prediction and during RLS update. XCSF reaches an accuracy level of $.00280 \pm .00055$ with 1948 ± 78 different classifiers after $200k$ learning iterations. After compaction, XCSF only requires 101 ± 11 classifiers to maintain an accuracy of $.00315 \pm .00058$. This again confirms that XCSF does not blindly evolve classifiers, but distributes them suitably over the problem space.

The performance differences are even more pronounced in the three dimensional case. The Neural GAS algorithm reaches an error level of $.0304 \pm .00231$ ($.0396 \pm .00212$) after $400k$ learning steps with a neural population size of $10,000$ neurons and using the (ten) closest rule(s) for prediction generation and RLS update. XCSF, on the other hand, reaches an error level of $.00340 \pm .00157$ with a population size of 2363 ± 55 before and it maintains an error level of $.00396 \pm .00053$ with a population size of 131 ± 12 after compaction.

The results confirm that XCSF with rotating hyperellipsoids and local RLS approximation yields competitive performance in comparison to other state-of-the-art learning algorithms. In the case of uniform sampling as well as in the case of restricted sampling, XCSF reaches accuracy levels of the best incremental statistics-based learning algorithms published on the topic. In comparison to an unsupervised clustering algorithm with added linear approximation (Neural GAS), XCSF exhibits superior performance in accuracy and compactness of the

[3]Parameters of Neural GAS were set to $\epsilon_i = .5$, $\epsilon_f = .01$, $\lambda_i = 10$, $\lambda_f = .5$. Rank-based center adjustment was applied until step $150k$ ($200k$ in 3D), until which values λ and ϵ exponentially decreased according to [26]. RLS with $\lambda = .99$ and $\delta_{RLS} = 1000$ was applied ($\lambda = 1$ showed slightly worse results due to the continuous center adaptations in Neural GAS). Additionally, to ensure that maximally accurate approximations are generated by RLS, matrix V was reset after $150k$ steps by multiplying the diagonal elements of V with δ_{RLS}. Twenty independent runs were conducted.

representation. XCSF solves the considered problems without using any sophisticated statistics except the one directly derived out of its error measures. Space partitioning is not changed heuristically, but is evolved by the evolutionary component of XCSF. Thus, XCSF is a flexible learning method in which the evolutionary component can be easily applied to a multitude of other problems including, for example, reinforcement learning problems in which reward propagation is required [12], [1].

VI. Summary and Conclusions

This paper investigated XCS's function approximation capabilities. We showed that XCS's performance could be improved in three ways. (1) Faster and more accurate linear approximation with efficient RLS stabilized and improved performance. (2) The representation of the classifier condition improved function approximation, in this case preventing unsuitable classifier overlaps. (3) The operators in the evolutionary process were optimized to enable faster learning by a more directed evolutionary process. In sum, XCS performance can be improved by optimizing gradient-based approximation, classifier representation, and the evolutionary process.

Moreover, we introduced a new compaction mechanism to XCSF. The mechanism is based on *closest classifier matching* (CCM) plus condensation (no mutation nor crossover in the GA application). In CCM, a fixed number of closest classifiers match, where closest is defined by the distance measure of each classifier itself. CCM prevents the generation of holes in the function approximation surface during compaction. Meanwhile, condensation causes the propagation of well-shaped accurate classifiers and the deletion of overlapping inaccurate classifiers. The mechanism was able to decrease population sizes by often more than 80% hardly affecting performance accuracy. An additional greedy compaction algorithm, which iteratively deletes classifiers that overlap with low-error classifiers, was shown to be able to compact the population by often more than 90% on average—albeit with a slight accuracy decrease in non-differentiable or highly irregular functions.

Results showed that the improvements enabled XCSF to solve function approximation problems of up to seven dimensions with highly compact final representations. Moreover, XCSF was shown to be noise robust and able to generalize well to unseen problem instances. In general, it was highlighted that XCSF is a learning mechanism that clusters the problem space to ensure maximally accurate approximations in the experienced subspaces. It outperforms general clustering algorithms, such as Neural GAS [26], [27], in function approximation tasks, and it approximates and partially outperforms more directed, iterative function approximation mechanisms published elsewhere [29], [28].

In conclusion, XCSF was shown to be a flexible, easily adaptable learning system that is applicable to many types of predictive tasks, and particularly tasks that can be approximated with local, partially-overlapping gradient-based estimates. Future work will evaluate the XCSF enhancements in datamining tasks as well as in reinforcement learning problems. Moreover, the mechanism is planned to be integrated into a cognitive systems architecture, in which the predictions manipulate neural gates in a recurrent neural network structure.

Acknowledgments

This work was supported by the European commission contract no. FP6-511931. The authors would like to thank the three reviewers for their helpful comments and suggestions. We especially acknowledge the suggestion to alter the RLS matrix initialization, which improved XCSF performance even further.

REFERENCES

[1] S. W. Wilson, "Classifier fitness based on accuracy," *Evolutionary Computation*, vol. 3, no. 2, pp. 149–175, 1995.

[2] ——, "Function approximation with a classifier system," *Proceedings of the Third Genetic and Evolutionary Computation Conference (GECCO-2001)*, pp. 974–981, 2001.

[3] ——, "Classifiers that approximate functions," *Natural Computing*, vol. 1, pp. 211–234, 2002.

[4] 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*, vol. 11, pp. 239–277, 2003.

[5] M. V. Butz, T. Kovacs, P. L. Lanzi, and S. W. Wilson, "Toward a theory of generalization and learning in XCS," *IEEE Transactions on Evolutionary Computation*, vol. 8, pp. 28–46, 2004.

[6] S. W. Wilson, "Generalization in the XCS classifier system," *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pp. 665–674, 1998.

[7] E. Bernadó, X. Llorà, and J. M. Garrell, "XCS and GALE: A comparative study of two learning classifier systems and six other learning algorithms on classification tasks," in *Advances in Learning Classifier Systems (LNAI 2321)*, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds. Berlin Heidelberg: Springer-Verlag, 2002, pp. 115–132.

[8] E. Bernadó-Mansilla and J. M. Garrell-Guiu, "Accuracy-based learning classifier systems: Models, analysis, and applications to classification tasks," *Evolutionary Computation*, vol. 11, pp. 209–238, 2003.

[9] M. V. Butz, *Rule-Based Evolutionary Online Learning Systems: A Principled Approach to LCS Analysis and Design*. Berlin Heidelberg: Springer-Verlag, 2006.

[10] P. W. Dixon, D. W. Corne, and M. J. Oates, "A preliminary investigation of modified XCS as a generic data mining tool," in *Advances in learning classifier systems: Fourth international workshop, IWLCS 2001 (LNAI 2321)*, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds. Berlin Heidelberg: Springer-Verlag, 2002, pp. 133–150.

[11] S. W. Wilson, "Get real! XCS with continuous-valued inputs," in *Learning classifier systems: From foundations to applications (LNAI 1813)*, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds. Berlin Heidelberg: Springer-Verlag, 2000, pp. 209–219.

[12] P. L. Lanzi, D. Loiacono, S. W. Wilson, and D. E. Goldberg, "Classifier prediction based on tile coding," *GECCO 2006: Genetic and Evolutionary Computation Conference*, vol. 2, pp. 1497–1504, 2006.

[13] ——, "Extending XCSF beyond linear approximation," *GECCO 2005: Genetic and Evolutionary Computation Conference: Volume 2*, pp. 1827–1834, 2005.

[14] M. V. Butz, D. E. Goldberg, and P. L. Lanzi, "Computational complexity of the xcs classifier system," in *Foundations of Learning Classifier Systems*, ser. Studies in Fuzziness and Soft Computing, L. Bull and T. Kovacs, Eds. Berlin Heidelberg: Springer-Verlag, 2005, pp. 91–126.

[15] C. Stone and L. Bull, "For real! XCS with continuous-values inputs," *Evolutionary Computation*, vol. 11, pp. 299–336, 2003.

[16] M. V. Butz, "Kernel-based, ellipsoidal conditions in the real-valued XCS classifier system," *GECCO 2005: Genetic and Evolutionary Computation Conference: Volume 2*, pp. 1835–1842, 2005.

[17] J. A. R. Marshall and T. Kovacs, "A representational ecology for learning classifier systems," *GECCO 2006: Genetic and Evolutionary Computation Conference*, vol. 2, pp. 1529–1536, 2006.

[18] P. L. Lanzi and S. W. Wilson, "Using convex hulls to represent classifier conditions," *GECCO 2006: Genetic and Evolutionary Computation Conference*, vol. 2, pp. 1481–1488, 2006.

[19] B. Widrow and M. Hoff, "Adaptive switching circuits," *Western Electronic Show and Convention*, vol. 4, pp. 96–104, 1960.

[20] P. L. Lanzi, D. Loiacono, S. W. Wilson, and D. E. Goldberg, "Prediction update algorithms for XCSF: RLS, kalman filter and gain adaptation," *GECCO 2006: Genetic and Evolutionary Computation Conference*, vol. 2, pp. 1505–1512, 2006.

[21] J. Drugowitsch and A. M. Barry, "A formal framework and extensions for function approximation in learning classifier systems," Dept. of Computer Science, University of Bath, Bath, UK, Tech. Rep. CSBU2006-01, 2006, iSSN 1740-9497.

[22] L. Bull and T. O'Hara, "Accuracy-based neuro and neuro-fuzzy classifier systems," *Proceedings of the Fourth Genetic and Evolutionary Computation Conference (GECCO-2002)*, pp. 905–911, 2002.

[23] J. Hurst and L. Bull, "A neural learning classifier system with self-adaptive constructivism for mobile robot learning," *Artificial Life*, vol. 12, pp. 1–28, 2006.

[24] P. W. Dixon, D. W. Corne, and M. J. Oates, "A ruleset reduction algorithm for the xcs learning classifier system," in *Learning classifier system: Fifth international workshop, IWLCS 2002 (LNAI 2661)*, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds. Berlin Heidelberg: Springer-Verlag, 2003, pp. 20–29.

[25] S. W. Wilson, "Compact rulesets from XCSI," in *Advances in learning classifier systems: Fourth international workshop, IWLCS 2001 (LNAI 2321)*, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds. Berlin Heidelberg: Springer-Verlag, 2002, pp. 196–208.

[26] T. M. Martinetz, S. G. Berkovitsch, and K. J. Schulten, ""neural-gas" network for vector quantization and its application to time-series prediction," *IEEE Transactions on Neural Networks*, vol. 4, pp. 558–569, 1993.

[27] T. M. Martinetz and K. Schulten, "Topology representing networks," *Neural Networks*, vol. 7, pp. 507–522, 1994.

[28] S. Schaal and C. G. Atkeson, "Constructive incremental learning from only local information," *Neural Computation*, vol. 10, pp. 2047–2084, 1998.

[29] D. Potts, "Incremental learning of linear model trees," *Proceedings of the Twenty-First International Conference on Machine Learning (ICML-2004)*, pp. 663–670, 2004.

[30] L. B. Booker, D. E. Goldberg, and J. H. Holland, "Classifier systems and genetic algorithms," *Artificial Intelligence*, vol. 40, pp. 235–282, 1989.

[31] J. H. Holland and J. S. Reitman, "Cognitive systems based on adaptive algorithms," in *Pattern directed inference systems*, D. A. Waterman and F. Hayes-Roth, Eds. New York: Academic Press, 1978, pp. 313–329.

[32] M. V. Butz and S. W. Wilson, "An algorithmic description of XCS," in *Advances in learning classifier systems: Third international workshop, IWLCS 2000 (LNAI 1996)*, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds. Berlin Heidelberg: Springer-Verlag, 2001, pp. 253–272.

[33] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning," *Artificial Intelligence Review*, vol. 11, pp. 11–73, 1997.

[34] T. Kovacs, "Deletion schemes for classifier systems." *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, pp. 329–336, 1999.

[35] M. V. Butz, D. E. Goldberg, P. L. Lanzi, and K. Sastry, "Bounding the population size to ensure niche support in XCS," Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, IlliGAL report 2004033, 2004.

[36] T. M. Mitchell, *Machine Learning*. Boston, MA: McGraw-Hill, 1997.

[37] C. Stone and L. Bull, "An analysis of continuous-valued representations for learning classifier systems," in *Foundations of Learning Classifier Systems*, ser. Studies in Fuzziness and Soft Computing, L. Bull and T. Kovacs, Eds. Berlin Heidelberg: Springer-Verlag, 2005, pp. 127–175.

[38] M. Ebner, M. Shackleton, and R. Shipman, "How neutral networks influence evolvability," *Complexity*, vol. 7, no. 2, pp. 19–33, 2001.

[39] E. W., "Euler angles," From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/EulerAngles.html, 1999.

[40] P. L. Lanzi, D. Loiacono, S. W. Wilson, and D. E. Goldberg, "Generalization in the XCSF classifier system: Analysis, improvement, and extensions," Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, IlliGAL report 2005012, 2005.

[41] S. Haykin, *Adaptive filter theory*, 4th ed. Upper Saddle River, NJ: Prentice Hall, 2002.

[42] L. B. Booker, "Improving the performance of genetic algorithms in classifier systems," *Proceedings of an International Conference on Genetic Algorithms and their Applications*, pp. 80–92, 1985.

[43] A. Bonarini, C. Bonacina, and M. Matteucci, "Fuzzy and crisp representations of real-valued input for learning classifier systems," in *Learning classifier systems: From foundations to applications (LNAI 1813)*, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds. Berlin Heidelberg: Springer-Verlag, 2000, pp. 107–124.

[44] L. B. Booker, "Classifier systems that learn internal world models," *Machine Learning*, vol. 3, pp. 161–192, 1988.

[45] M. V. Butz, D. E. Goldberg, and P. L. Lanzi, “Bounding learning time in XCS,” *Proceedings of the Sixth Genetic and Evolutionary Computation Conference (GECCO-2004): Part II*, pp. 739–750, 2004.