CORE
Metadata, citation and similar papers at core.ac.uk
Provided by Elsevier - Publisher Connector

Theoretical Computer Science 425 (2012) 126–141

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Resource management and scalability of the XCSF learning classifier system

Patrick O. Stalph [a], Xavier Llorà [b], David E. Goldberg [b], Martin V. Butz [a,*]

[a] Department of Cognitive Psychology, University of Würzburg, 97070 Würzburg, Germany

[b] Illinois Genetic Algorithm Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA

ARTICLE INFO

Keywords:
Learning classifier system
XCS
Function approximation
Scalability
Resource management
Structure alignment

ABSTRACT

It has been shown many times that the evolutionary online learning XCS classifier system is a robustly generalizing reinforcement learning system, which also yields highly competitive results in data mining applications. The XCSF version of the system is a real-valued function approximation system, which learns piecewise overlapping local linear models to approximate an iteratively sampled function. While the theory on the binary domain side goes as far as showing that XCS can PAC learn a slightly restricted set of k-DNF problems, theory for XCSF is still rather sparse. This paper takes the theory from the XCS side and projects it onto the real-valued XCSF domain. For a set of functions, in which fitness guidance is given, we even show that XCSF scales optimally with respect to the population size, requiring only a constant overhead to ensure that the evolutionary process can locally optimize the evolving structures. Thus, we provide foundations concerning scalability and resource management for XCSF. Furthermore, we reveal dimensions of problem difficulty for XCSF — and local linear learners in general — showing how structural alignment, that is, alignment of XCSF's solution representation to the problem structure, can reduce the complexity of challenging problems by orders of magnitude.

© 2010 Elsevier B.V. All rights reserved.

1. Introduction

The application of Evolutionary Algorithms [1] to Machine Learning tasks led to the field of Genetics-based Machine Learning—Learning Classifier Systems (LCS) have become an integral part of this research field. John H. Holland envisioned LCS as cognitive systems that receive perceptions from the environment and perform actions depending on a set of rules [2]. These rules, also called classifiers, consist of a condition part that specifies the active context and a prediction part that approximates the (reward) function in this context. While the predictions are adapted by gradient descent techniques, a Genetic Algorithm (GA) evolves the condition structure.

The most prominent Learning Classifier System is XCS [3], and successful applications emphasize the robustness and generalization capabilities of that system: XCS has been applied to multistep problems [3,4], data mining tasks [5,6], as well as robot applications [7,8]. Furthermore, XCS can be used for multidimensional real-valued function approximation, then called XCSF [9,10]. The XCSF system learns piecewise overlapping local linear models to approximate an iteratively sampled function.

For XCS in the binary domain, theory on population size bounds as well as learning time bounds is available. Furthermore, it was shown that XCS can PAC learn a slightly restricted set of k-DNF problems [6]. However, theory for XCSF, that is, for

* Corresponding author. Tel.: +49 931 3182808.
E-mail addresses: patrick.stalph@psychologie.uni-wuerzburg.de (P.O. Stalph), xllora@illigal.ge.uiuc.edu (X. Llorà), deg@illigal.ge.uiuc.edu (D.E. Goldberg), butz@psychologie.uni-wuerzburg.de (M.V. Butz).

0304-3975/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2010.07.007

applications to the real-valued domain,[1] remains sparse. This paper takes the theory from the XCS side and projects it onto the real-valued XCSF domain. Therefore, we first identify factors that influence the learning performance of the system. Next, we derive population size and learning time bounds according to available XCS theory in the binary domain. Building upon this knowledge, we derive a *Scalability Model* with respect to a crucial performance measure, that is, the population size.

For a restricted set of functions, we show that XCSF scales optimally requiring only a constant overhead in its population size to ensure that the evolutionary process can locally optimize the evolving structures. Although underlying assumptions include some restrictions, the model may generally serve as a worst case scenario and, thus, serve as an upper bound on the population size for other functions and applications. Thus, we provide foundations concerning scalability and resource management for XCSF.

Finally, with the worst case scenario in mind, we discuss dimensions of problem difficulty for XCSF—and locally linear learners in general. There is no bad weather, there are just wrong clothes. The same holds for XCSF: Given appropriate representational power, the XCSF system can evolve suitable approximations for very complex functions. Using a simple example we show how *structural alignment* can reduce the problem complexity drastically.

The remainder of this paper is structured as follows. First, we introduce the XCSF Learning Classifier System in Section 2. Next, we focus on relevant factors that influence the performance in Section 3 and discuss requirements for successful learning in Section 4. In turn, we provide a *Scalability Model* in Section 5. The model is validated empirically in Section 6. Finally, we discuss the importance of *Structure Alignment* in Section 7 and provide a simple example. The article ends with a short summary and concluding remarks in Section 8.

2. The XCSF learning classifier system

The XCSF system is a *genetics based machine learning* technique that evolves a population of classifiers in order to approximate functions. Therefore, the population of classifiers represents a set of overlapping, piecewise, typically linear function approximations. Each classifier covers a small subspace of the input space and approximates the function in this subspace. Given an input value, the weighted average of individually matching classifier predictions forms the final predicted function output for the input. As a whole, XCSF is designed to develop one complete function approximation surface represented by its evolving population of piecewise linear function approximations, that is, real-valued classifiers. The difference between predicted value and actual function value is the so called prediction error, which is used to improve the predictions by means of gradient descent, and the condition structure as well by means of a steady state genetic algorithm. The rough workflow is shown in Fig. 1(a)—the details are explained below.

In order to approximate a part of a function, a classifier consists of a condition part and a prediction part. The condition part specifies the part of the input space where the classifier is active. The simplest formulation is just an interval, that is, a hyperrectangle [11,12], but more elaborate geometric shapes such as rotating hyperellipsoids [10] or arbitrary functions using gene expression programming [13] were also developed. The prediction part of the classifier can be any approximator that works online, such as a simple constant approximator that returns the average of all sampled function values.[2] However, most often used is the linear recursive least squares prediction [14], which yields good approximations fast, while results of a linear approximation can be interpreted easily. The interplay of condition and prediction structures is depicted in Fig. 1(b) for a single classifier.

Consider an arbitrary multidimensional function

$$f : \mathbb{R}^n \to \mathbb{R}^m, \qquad \vec{y} = f(\vec{x}). \tag{1}$$

During learning, XCSF receives one sample $\vec{x}_t, \vec{y}_t$ at each iteration t. First, the population is scanned for active classifiers—classifiers that match the input, that is, $\vec{x}_t$ satisfies the condition part of the classifier. These classifiers form the so called match set $[M]$. During initial iterations, the match set is often empty, because no matching classifiers are found in the population. In this case, a new classifier of random size is generated, with its center at the current input $\vec{x}_t$. Depending on the type of condition, the initial size of classifiers is usually specified by a parameter r_0. The so called covering process ensures that $[M]$ contains at least one classifier.

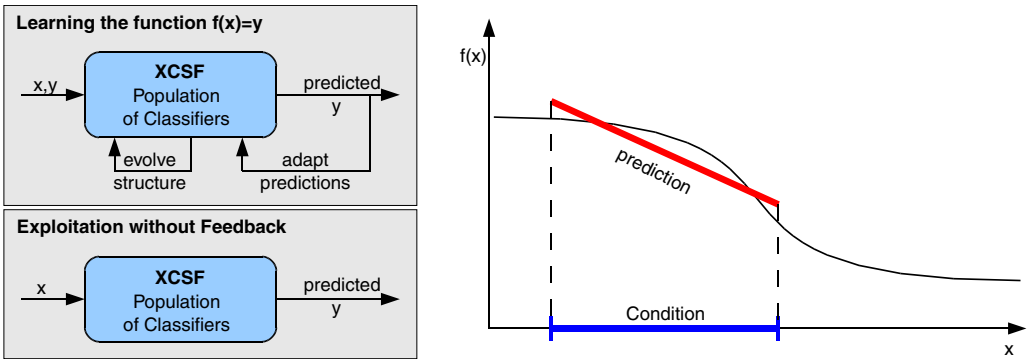
After the match set is determined, each classifier in $[M]$ predicts the function value $\vec{y}_t$ using its prediction part. For a linear predictor, this is just the multiplication of input and prediction weights, e.g. $\vec{p} = \vec{x}_t \cdot \vec{w} + \vec{w}_0$, where $\vec{w}_0$ is an offset and $\langle\cdot\rangle$ is the dot product. However, any kind of function prediction may be used instead of a linear approximator. Before explaining how the final prediction is calculated, additional attributes of a classifier and the related update rules have to be introduced.

All classifiers that participate in the current match set are updated using the current environmental feedback. Given the feedback $\vec{y}_t$ at iteration t and the current predicted value $\vec{p}_t$ of a classifier, the current prediction error ε_t of this classifier is updated using the modified Widrow–Hoff rule

$$\varepsilon_t = \varepsilon_{t-1} + \max\left(\frac{1}{\xi}, \beta\right)(|\vec{p}_t - \vec{y}_t| - \varepsilon_{t-1}), \tag{2}$$

[1] The term *domain* refers to input and output spaces.

[2] Such a prediction is *constant* with respect to the inputs, not time.



(a) Workflow. (b) Classifier illustration

Fig. 1. (a) The basic workflow of XCSF. For each given sample the function value is predicted. Predictions are updated using the error of this prediction and finally the GA operates on the condition structure, where the goal is to minimize the prediction error. (b) A classifier consists of a condition part that specifies the part of the input space where the classifier matches inputs, and a prediction part that approximates the function in this part of the input space. As an example, an interval condition and a linear prediction are depicted.

where ξ is the experience of the classifier, that is, the number of match set participations, and $0 < \beta \leq 1$ specifies the learning rate for well experienced classifiers. The final term describes the change in prediction error, since $|p_t - y_t|$ is the current absolute error and ε_{t-1} is the previous prediction error. With this formula, new inexperienced classifiers quickly adapt, while the update rate depends on β for experienced classifiers. The same technique is used to update the estimated match set size ψ_t, which represents an estimate of the average size of matchsets this classifier participated in so far

$$\psi_t = \psi_{t-1} + \max\left(\frac{1}{\xi}, \beta\right)(|[M]| - \psi_{t-1}). \tag{3}$$

Next, the fitness value of a classifier is derived from its relative accuracy in the match set. Accuracy κ_t is a scaled inverse of the prediction error, where classifiers with an error below the target error ε_0 are considered accurate and receive the maximum accuracy of 1. The accuracy of a classifier is

$$\kappa_t = \min\left(\left(\frac{\varepsilon_0}{\varepsilon_t}\right)^{\nu}, 1\right), \tag{4}$$

where $\nu > 1$ amplifies the penalty for a high prediction error. Now, the current fitness φ_t can be determined using the relative accuracy of the classifier in the match set

$$\varphi_t = \varphi_{t-1} + \beta\left(\frac{\kappa_t}{\sum_{i=1}^{|[M]|}\kappa_{t_i}} - \varphi_{t-1}\right). \tag{5}$$

Again, the learning rate β specifies the update rate. This sophisticated update scheme enforces fitness sharing within the current match set, since accurate classifiers receive a large share of the available fitness. The fitness value specifies the overall quality of a classifier, relative to its competitors, and is required to determine the final prediction as well as for the evolutionary search.

We recall, that the match set is formed by all active classifiers that match the current input x_t at iteration t. The fitness-weighted average of the individual classifier predictions forms the final prediction for the given input and is computed by

$$\hat{y} = \frac{\sum_{i=1}^{|[M]|}\varphi_i p_i}{\sum_{i=1}^{|[M]|}\varphi_i}, \tag{6}$$

where φ_i is the fitness of the i-th classifier in $[M]$ and p_i is the predicted function value of this classifier. Here, we omitted the iteration index t for better readability.

With the fitness values available, a steady-state GA selects two classifiers from the match set using tournament selection. These classifiers are reproduced, crossover is applied with probability χ, while mutation is applied with probability μ. In turn, the new classifiers are inserted in the population. If the maximum population size N is exceeded, classifiers are deleted from the whole population by means of roulette-wheel selection. The probability of deleting a particular classifiers depends

on (1) the experience ξ, (2) the fitness φ, and (3) the crowding in that region, estimated by the average match set size ψ. Given further the average fitness $\varphi_{avg} = \frac{1}{|[M]|}\sum_{i=1}^{|[M]|}\varphi_i$ in the current match set, the relative deletion vote of a classifier equates to

$$\zeta = \begin{cases} \psi \cdot \frac{\varphi_{avg}}{\varphi} & \text{if } \xi > \theta_{del} \wedge \varphi < 0.1\,\varphi_{avg}, \\ \psi & \text{otherwise.} \end{cases} \tag{7}$$

In both cases the deletion probability scales linearly with the estimated number of classifiers responsible for the same region, thus realizing a trend towards removing classifiers from overcrowded regions. Furthermore, classifiers with sufficient experience and a bad fitness value (first case) receive at least a 10 times higher deletion vote, since the precondition $\varphi < 0.1\,\varphi_{avg}$ implies $\frac{\varphi_{avg}}{\varphi} > 10$. In sum, experienced classifiers with low fitness in overcrowded regions receive the highest deletion probability.

Although evolutionary pressures drive classifiers towards accuracy and generality, there is a considerable overhead in the population size due to reproduction and overlap of classifiers. After a suitable solution is evolved, the population can be reduced using the so called condensation technique [15,10]. By simply turning off mutation and crossover for a fixed amount of iterations, only the reproduction and deletion mechanisms act upon the population. Moreover, the matching mechanism is switched to matching a fixed number of closest classifiers (usually set to 20), regardless if the classifiers match. Due to fitness-based selection, those classifiers with high fitness reproduce, while the deletion mechanism preferably removes classifiers with low fitness in crowded regions. Usually, the approximation accuracy is hardly affected, while population size reductions of 90% or even more are possible.

To sum up the core features of XCSF, classifiers predict the function value in a subspace of the input space and the corresponding prediction error is computed, which determines accuracy and fitness. The GA strives to evolve accurate classifiers, while more general classifiers get more reproductive opportunities. Thus, a population of maximally general and accurate classifiers is evolved. Given an input, this population can then be used to predict the corresponding function value.

3. Performance factors

In order to develop a scalability theory for XCSF, we first identify and decompose the factors that influence the learning performance of the LCS. For the XCS system, this has been investigated in detail with respect to Boolean functions [6]. In this article, we tackle this topic with respect to continuous-valued functions.

We distinguish three categories that influence the performance, namely the *function complexity*, XCSF's *representational power*, and the *learning capacity* [16]. Although we distinguish these three categories, they are not independent and several aspects of each may have similar effects on the performance—additionally, there are strong interactions between the categories. Nonetheless, the origin of problem difficulty is different for each. After decomposing each category further into several aspects in the next sections, we then discuss the interactions between them.

3.1. Function complexity

Various factors contribute to the complexity of a function from the function approximation viewpoint. First, the number of problem dimensions and particularly also the number of relevant principal components has a huge impact on the learning performance, since an increase in the number of dimensions is tied to the curse of dimensionality. Second, the structure of the function, for example the gradient or curvature, eventually makes a learning process more difficult. Generally, the "structure" of a function cannot be precisely defined with respect to function complexity, because the resulting complexity is also dependent on the solution structure that is expressible by the learner. However, any structural regularities within local subspaces of the input space of the function can simplify the problem significantly. Extreme imbalances of such regularities, however, may also hinder learning, potentially causing overgeneralizations. Finally, also the sampling distribution of instances, from which XCSF learns, plays an important role. Unbalanced sampling may decrease performance in that an underrepresented subspace may be forgotten. We assume uniform function value sampling. Theory for non-uniformly sampled problem spaces has been developed elsewhere [17].

3.2. Representational power

XCSF's solution representation is made up of two important parts: the condition structure and the predictor. The type of condition structure specifies how the problem can be partitioned to enable accurate local function approximations. Furthermore, the complexity of the condition type, in particular the size of the genotype, defines the size of the evolutionary search space. For example, the mutation operator of an interval type condition can modify upper and lower bounds per dimension, whereas the mutation of a rotating condition also includes the modification of rotation angles. However, if the condition is too simple, the genetic algorithm is unable to find a suitable structure for the predictor. If, on the other hand, the condition type is more complex than the underlying problem, a large part of the evolutionary search space represents a

suitable solution for the GA—thus there is an overhead, but evolution will quickly find the relevant subspace. The GA faces the most difficult situation when only a small subspace of the evolutionary search space, that is, a very specific shape of the condition structure, yields a suitable solution. As an example, assume a high dimensional function with one oblique principal component, e.g. the seven-dimensional oblique sine function $f(x) = \sin(4\pi(x_1 + \cdots + x_7))$ found in [10]. Here the GA has to evolve conditions with very small stretches along the sinusoidal axis and very large stretches along the linear subspaces resulting in large rotated slices. Considering the number of rotation angles, this is a small subspace to search for.

In contrast to the evolving condition structure, the predictor must quickly converge to an optimal approximation of its condition subspace in order to yield a reliable prediction error, which in turn yields the fitness for the evolutionary search. The least squares technique is suitable for this task and for online learning the recursive version is widely used. Concerning the impact on learning performance, the type of predictor (e.g. constant, linear, or polynomial) determines what kind of functions can be accurately approximated in the subspaces defined by the condition part.

In combination, the condition and prediction type define the structure of the solution that is evolved. Several condition structures are available and have been rather exhaustively discussed in the literature, ranging from simple rectangular conditions to rotating hyperellipsoidal conditions [10] or gene expression programming [13]. Similarly, various prediction structures and approximation techniques have been employed including simple constant and linear predictions, linear and polynomial recursive least squares predictions [18], or even Kalman-filtering based linear predictions [19]. Clearly, there is a tradeoff between condition and prediction complexity. That is, if the predictor is simple, a more expressive condition representation is required to chunk the space. On the other hand, if the predictor is flexible, a coarser partition of the input space may suffice.

3.3. Learning capacity

While the posed function complexity and the available solution structures determine the general complexity of the problem, the learning capacity refers to the learning resources available to XCSF. This includes the population size, the number of learning iterations, and the target error—parameters, that are specified by the user. The number of learning iterations refers to the number of iterations XCSF can search for the optimal problem solution.

The size of the solution as well as the number of extra classifiers for evolutionary search is bounded by the maximum population size—an important parameter for XCSF. Set too low, the GA is unable to search for a new and better condition structure without deleting good classifiers. Set too high, evolution does not face the deletion pressure and consequently the unnecessary classifier overhead slows down evolution. Thus, a very large population size is not detrimental but annoying. In our study below, we determine a lower bound on the population size confirming that this lower bound scales optimally given a certain function complexity and representational power.

The target error ε_0 specifies the desired prediction accuracy and, thus, modifies the equilibrium of accuracy and generality. Reducing the target error usually results in smaller conditions, in order to restrict the subspace of the predictor, and, consequently, a larger population size is needed to accurately learn the problem.

3.4. Interactions

We identified three categories, namely the function complexity, XCSF's representational power, and the learning capacity. However, there are strong interactions between those performance factors—no category is independent of the others. Roughly said, a complex function structure may be accurately represented by a suitable solution representation, thus, from XCSF's viewpoint, the function is not complex. Additionally, the learning capacity plays an important role because a high accuracy requirement in conjunction with a low population size demands a problem-fitting solution structure in order to represent the approximation with few highly accurate classifiers. Consider a linear function $f(x) = mx + t$. For XCSF it depends on its representational power, if this function is easy to approximate or not. Given a linear predictor, the function is trivial. Given a constant predictor, a considerable number of classifiers are necessary to accurately approximate $f(x)$, and "accurate" depends on the target error.

To sum up, there are various ways to tune the problem difficulty in terms of population size and learning time, and all performance factors have to be considered when speaking about *difficulty*. Before we model some of these relations, in particular the relations between function structure, target error, and population size, the theory from XCS's binary domain is projected onto the real-valued domain.

4. Learning challenges

For XCS, problem structural factors were analyzed in depth. In consequence, various learning challenges were identified that resulted in several population size and learning time bounds. The bounds essentially characterized the scalability of the XCS system in increasingly large problems with bounded complexity [6].

The learning challenges for XCS were described as (1) a covering challenge, which requires that the problem space is sufficiently covered by the initially generated classifiers, (2) a schema challenge, which requires that there is minimal structure that gives fitness information towards better sub-solutions, (3) a reproductive opportunity challenge, which

requires that these structures can reproduce and thus grow, (4) a learning time challenge, which determines the minimal time necessary to find a solution given a certain initial population, and (5) a solution sustenance challenge, which specifies the resources necessary to sustain the complete solution during continued evolution. We now transfer these challenges into the real-valued XCSF domain, discuss their relevance in the investigated domain, and derive respective population size and time bounds, where applicable.

Many of the bounds derived for XCS are based on the specificity of particularly structured classifiers and their consequent probability of matching $P(\texttt{match})$. As pointed out elsewhere [10], the measure of specificity strongly correlates with the hyper-volume of a classifier in a real-valued input space. Given a one unit hyper-volume of space and given further uniform problem sampling, the hyper-volume specified by a classifier condition – that is, the size of the subspace covered by a classifier – corresponds exactly to the probability of matching a problem instance. Thus, ignoring boundary effects, we carry over the notion of specificity to the real-valued domain and are assuming that the volume of a classifier condition corresponds with its probability of matching.

Given non-uniformly sampled spaces, the matching probability further depends on the sampling and the derived bounds apply to the least frequently sampled subspace, given that the complete expressible space is represented. Given a sampling of a lower-dimensional manifold in the higher-dimensional space, or generally given subspaces that are not sampled at all, then the theory applies to the hyper-volume of the (uniformly) sampled subspace in the expressible full input space, since XCSF only searches within (and relatively close to) the sampled subspace. A detailed study on non-uniformly sampled problem spaces in the data mining domain including an approach on how to detect non-uniform sampling and how to consequently adjust the parameters of XCS to prevent the loss of accuracy can be found elsewhere [17].

While a more exact approximation of the subsequent bounds may be accomplished by also taking the covariance of the evolving classifier volumes into account, the main purpose of this paper is to generally carry over the bounds to the real-valued domain and then verify the scalability of XCSF. More exact bounds, which are generally expected to scale similarly to the bounds derived below, are left for future theoretical work.

4.1. Covering challenge

The covering challenge reflects the observation that the initial population of XCSF should be able to cover the full problem space. Given a particular volume used to initialize classifiers, this volume corresponds directly to the probability of matching $P(\texttt{match})$. Given the probability $P(\texttt{match})$, one can derive the probability that a given problem instance is covered by at least one classifiers given a randomly initialized population of size N:

$$P(\texttt{cover}) = 1 - (1 - P(\texttt{match}))^N . \tag{8}$$

When requiring a sufficiently high probability of covering all input instances of $P(\texttt{cover}) \geq 1 - 10^{-c}$, where $c \geq 1$ is the confidence, a population size bound can be derived:

$$\begin{aligned} & P(\texttt{cover}) \geq 1 - 10^{-c} \\ \Rightarrow \quad & 1 - (1 - P(\texttt{match}))^N \geq 1 - 10^{-c} \\ \Rightarrow \quad & (1 - P(\texttt{match}))^N \leq 10^{-c} \\ \Rightarrow \quad & N \log(1 - P(\texttt{match})) \leq -c \\ \Rightarrow \quad & N \geq \frac{c}{-\log(1-P(\texttt{match}))} . \end{aligned} \tag{9}$$

Using furthermore the inequality $x < -\ln(1 - P(x))$, the population size can be bounded from below by:

$$N > \frac{c}{P(\texttt{match})}. \tag{10}$$

The bound shows that the population size grows linearly in the certainty exponent and inversely linear with the initial classifier volumes. Essentially, the bound can be used to prevent a covering–deletion cycle, in which XCSF would continuously generate random classifiers and delete others, never covering the full problem space. The occurrence of such a cycle can prevent XCSF from functioning at all, because the problem space is never covered and thus the evolutionary algorithm cannot search for better classifier structures. In practice, the initial classifier volumes and population size should thus be set such that the full input space is covered with high probability. The GA will then adapt the classifiers to the problem at hand given a sufficient fitness signal as discussed in the next section.

4.2. Schema and reproductive opportunity challenge

Given a full coverage of the problem space does not assure that learning is successful. In XCS, schema and reproductive opportunity bounds were derived, which assure that substructures that achieve higher fitness values than the initial structures can be (a) detected, or are present in the initial population, and (b) can reproduce. In binary genetic algorithms, such substructures are often termed *building blocks*, as originally proposed in Holland's *schema theory* [20].

As binary GAs minimize (maximize) a binary function, while real-coded GAs minimize (maximize) real-valued functions, one might think that the transfer of the schema theory to real-valued functions is straightforward and various attempts have

been made in this respect [21–25]. However, the difficulty of general numerical optimization problems is unbounded, since the search space is infinite (or at least as high as numerical precision allows) and consequently the number of substructures, e.g. optima or dependencies, can be infinite, too [26]. In real-valued GAs, as well as in evolution strategies, it remains obscured how a building block can be defined and also recombination has been shown to only be useful in particular problems [27]. Thus, it is a hard problem to define a building block for a Learning Classifier System with real-valued inputs and outputs.

For XCSF, it is essential that there is some fitness signal towards better structured and/or smaller classifier conditions, that is, smaller volumes. That is, some of the offspring must have a chance to experience higher accuracy values. The structures that are necessary to receive such a fitness signal, thus providing schema supply and possibly growth, however, strongly depend on the to-be approximated function. Thus, schema and reproductive opportunity bounds are hard to carry over to XCSF. In the problems below, we analyze the simplest problem in this respect, approximating a linear function with constant predictions. In this case, the smaller a classifier volume the higher its fitness, thus always receiving a fitness signal towards higher accuracy. For a successful approximation of more complex functions it has been shown that sufficiently small initial classifier structures may be necessary [10]. For our further analyses we thus assume the presence of a fitness signal. Later, we discuss structural alignment, where the fitness signal is not only relevant for the size of the condition structures but also for their shape and orientation in the input space.

4.3. Learning time challenge

Given there is a sufficiently reliable fitness signal towards more accurate condition structures and given further that these structures have time to reproduce and grow, the time until maximally accurate structures evolved is an essential part of the computational complexity and the scalability of XCSF in increasingly large and complex problems.

For the binary XCS, a domino convergence model [28] was used to estimate the maximum time until the maximally accurate structures are discovered [6]. In this case, only mutation was considered and the time was estimated until reproduction of currently best structures and the generation of even better structures by mutation. Essentially, it was shown that learning time scales linearly in the number of binary dimensions and inversely linear in the minimum classifier volume required.

A similar model can be employed for XCSF, estimating successively successful reproductions and better offspring generations. We simplify the creation of *better* offspring generations to the creation of a more *specialized* offspring, which is also the requirement in our scalability analyses with respect to population size requirements below. In this way, it is possible to estimate the probability of a better offspring generation. Let the mutation factor be uniformly taken from $[0.5, 1.5]$ for each dimension and assume that one mutation takes place per classifier (e.g. by setting $\mu = 1/n$). Thus, the best mutation yields half the volume and the probability for a good mutation (resulting in a smaller volume) is 0.5. Assuming further that all classifiers with smaller volume are correctly identified and consequently reproduce themselves, we can determine the average volume of successively reproduced and mutated classifiers given a current classifier volume of V_t:

$$V_{t+1} = \frac{3}{4} V_t. \tag{11}$$

Given the initial classifier volume for covering V_0, we can estimate the number m of mutations needed to reach the target volume $V_{goal} < V_0$

$$\begin{aligned} & V_0 \left(\frac{3}{4}\right)^m = V_{goal} \\ \Rightarrow \qquad & m = \log_{\frac{3}{4}} \left(\frac{V_{goal}}{V_0}\right). \end{aligned} \tag{12}$$

Note that since we are assuming an equal problem complexity over the whole problem space and XCSF essentially searches locally independently in all problem spaces, the expected time until the generation of one target volume also corresponds to the overall expected time of generating all (equally sized) target volumes in all problem subspaces. We recall that we assumed 50% of the mutations to yield a better fitness value. Until the target volume is reached, m mutations are required, where each mutation reduces the volume to 3/4. An estimate of the number of generations, until the target volume is reached, equates to

$$\begin{aligned} E(\texttt{iterations}) &= \sum_{i=1}^{m} \frac{1}{0.5\, V_0\, (3/4)^i} = \frac{1}{0.5\, V_0} \underbrace{\sum_{i=1}^{m} \left(\frac{4}{3}\right)^i}_{\text{geo. series}} \\ &= \frac{2}{V_0} \cdot \left(\left(\frac{4}{3}\right)^m - 1\right). \end{aligned} \tag{13}$$

Although several simplifications do not allow for a learning time approximation in a concrete scenario, the above considerations allow for a complexity estimate. The value of m depends logarithmically on the distance from initial to final

volume and has an exponential influence on the expected number of iterations. Thus, the learning time scales approximately linear in the distance (in terms of mutations) to the target classifier shape.

4.4. Solution sustenance

Given the previous requirements are fulfilled and, thus, the algorithm evolves a solution to the underlying problem, this solution has to be sustained. Essentially this bound is concerned with the deletion probability, where the goal is to prevent the loss of a niche in the solution representation.

For XCS, this bound was derived for independent solution niches using the steady state derivation of a Markov chain model resulting in a final population size bound [29]. In XCSF, there are of course no independent niches since the problem space is continuous and typically classifiers strongly overlap. Thus, a subspace can be represented by multiple overlapping classifiers. However, once a sufficiently accurate solution is found, a particular subspace will be represented by these overlapping classifiers. Thus, the general theoretical considerations of XCS hold, given a particular final sub-space solution size. This size strongly depends on the problem complexity and the available representational power, as discussed above.

Given a particular niche size, that is, a particular classifier volume and thus a particular probability of matching $P(\texttt{match})$, we can determine the population size that is necessary to assure matching and thus reproduction before complete deletion of (possibly overlapping) niche representatives. Elsewhere [29], it was shown that the distribution of number of niche representatives yields a binomial distribution of the form:

$$u_j = \binom{N}{j} P(\texttt{match})^j (1 - P(\texttt{match}))^{N-j}, \tag{14}$$

where u_j denotes the probability that the particular subspace is represented by j classifiers. Using this equation, the population size N can be bounded ensuring with high probability that XCSF does not lose any of the problem niches, that is, any subsolutions. A subsolution is essentially lost when in state $u_0 = (1 - P(\texttt{match}))^N$. Thus, the probability of losing a niche decreases exponentially with the population size.

When once again requiring a certainty of $1 - 10^{-c}$ of not losing a subspace, where $c \geq 1$ is the confidence, then the following population size bound can be derived:

$$\begin{aligned} & 1 - u_0 \geq 1 - 10^{-c} \\ \Rightarrow \quad & (1 - P(\texttt{match}))^N \leq 10^{-c} \\ \Rightarrow \quad & N \log(1 - P(\texttt{match})) \leq -c \\ \Rightarrow \quad & N \geq \frac{c}{-\log(1-P(\texttt{match}))}. \end{aligned} \tag{15}$$

By means of the inequality $x < -\ln(1 - P(x))$, the population size can be bounded from below by:

$$N > \frac{c}{P(\texttt{match})}, \tag{16}$$

which is the exact same bound as derived for the covering challenge. Thus, also to sustain a sufficiently accurate solution representation, the population size needs to grow linearly in the confidence and inversely linear in the minimal classifier volume that is to be sustained.

Several simplifications were done during this derivation. First, a sufficiently strong fitness signal was assumed to prevent overgeneralizations, essentially assuring that appropriate niche representatives will be reproduced. Second, the overlapping classifiers may still have an influence. In uniformly complex problems, this influence can be assumed to be uniform and thus average out. However, in non-uniformly complex problems further overlapping fitness pressures disallow an exact solution sustenance derivation. Future research is necessary to analyze the solution sustenance challenge in such problems in further detail.

However, since the sustenance of the final solution and particularly the prevention of holes in the solution surface is additionally assured by switching to closest classifier matching after learning ends and condensation is applied [10], the sustenance bound should hold for locally approximately uniformly complex problems.

4.5. Conclusions

In conclusion, this section has shown that as long as the fitness signal is strong enough and the to-be approximated function is not highly non-uniformly sampled or complex, the population size requirements are all in an effectively computable range, that is, the population size scales approximately linear with the confidence of covering or solution sustenance. The learning time is expected to scale linearly with the number of required mutations from initial classifier shape to the target shape.

In the next section we derive a scalability model that links function structure, dimensionality, and the population size. We verify that the computational overhead is only by a constant larger than the optimal solution size required. Thus, we confirm that in uniformly sampled problems with an approximately uniform complexity, XCSF is guaranteed to find an accurate problem solution with high probability and a computational effort that is optimal with respect to the population size requirements.

5. Scalability theory

Having identified relevant performance factors and having derived computational requirements in terms of population size and approximate learning time, the focus of this section is on the relation between the mentioned performance factors and the resulting performance of XCSF in terms of scalability. In particular, we develop a scalability theory that models the relations between

1. function structure and dimensionality,
2. XCSF's solution representation,
3. target error ε_0, and
4. population size N.

Unbalanced sampling is not modeled, since this is investigated elsewhere [17]. Furthermore, the learning time is not included here and assumed to be high enough for XCSF to find the optimal solution.

Up to now, the term ''function structure'' is not precisely defined. We can get a concise grip on this term by defining the complexity of the function by means of the prediction error. Thus, if we want the function to be twice as complex, we just modify the function in such a way that the prediction error is twice as high for the same classifier. The complexity of the structure of a function will thus generally be characterized by the maximally general classifier volume necessary to reach a certain low prediction error.

5.1. Assumptions

Having specified the relations to be modeled, a few assumptions have to be made. First, we assume uniform sampling and a uniform function structure. Next, without loss of generality, the input space is confined to the range $[0, 1]^n$, where n is the number of dimensions. This simplifies equations and enhances the readability. Furthermore, since we want to model influences on the population size and not on the learning time, we assume the number of iterations to be large enough to enable XCSF to converge to the optimal solution.

Besides the function structure, however, the *optimal* solution depends on the representational power XCSF has. Concerning binary functions, four properties of optimal solutions for Learning Classifier Systems were derived [30]. With minor modifications these properties can be transferred to XCSF.

1. completeness — Each possible input is covered, that is, at least one classifier matches an input.
2. correctness — The population predicts the function surface accurately, that is, the prediction error is below the target error ε_0.
3. minimality — The population contains the minimum number of classifiers needed to represent the function completely and correctly.
4. non-overlappingness — No input is matched by more than one classifier.

Assuming that XCSF evolves an *optimal* solution has several implications. While completeness is a rather trivial assumption, correctness implies that each classifier is *accurate*, that is, the prediction error is less or equal to the target error ε_0. Under the assumption of a uniform function structure, minimality and non-overlappingness additionally imply that each classifier is *maximally general*, that is, the volume is as large as possible while the classifier is still accurate. It follows that the prediction error equals the target error.

In sum, we assume a uniform patchwork of equally-sized, non-overlapping classifiers covering the whole input space—for a continuous (infinite) input space this assumption is clearly far from reality. Usually classifiers overlap in order to make sure that the whole space is covered and predictions are formed by weighted averages of individual classifier predictions. However, modeling the overlap is neither trivial nor in the scope of this article. Additionally, a considerable part of the population consists of new classifiers, inserted by the GA after crossover and mutation. Although the condensation technique may be applied to reduce the final population, this method does not guarantee to remove all the overhead due to the evolutionary learning process. In sum, the population size is expected to be larger than specified by the following theoretical considerations due to overlap. Nonetheless, the population size model can serve as a tight lower bound, as will be also experimentally verified.

Due to the uniform function structure, XCSF is either facing a trivial problem when the prediction is expressive enough to represent the uniform structure, or XCSF is facing a checkerboard problem [31], that is, a problem in n dimensions for which the best input space division is a set of uniformly distributed, equally sized hypercubes. We focus on the checkerboard problem and derive minimal population size bounds for covering the input space by accurate and maximally general classifiers.

5.2. Scalability model: classifier volume and population size

With the assumptions defined, we recall the relations to be modeled. In order to gain insights about the scalability of XCSF, we model the influence of *function structure* and *dimensionality* on the *classifier volume* and, consequently, on the *population size*. We consider a function

$$f_\Gamma : [0, 1]^n \to \mathbb{R}, \tag{17}$$

where Γ is a factor that modifies the function structure and n is the number of input dimensions.

An optimal classifier, that is, an accurate and maximally general classifier, has a uniform shape and the same extent in each dimension. Let V be the volume of such a classifier. Given the uniform problem structure in n dimensions, the classifier volume influences the prediction error ε in a polynomial fashion, that is, $\varepsilon^n \sim V$. In order to get a concise grip on the function structure, we require that a linear increase in Γ results in a linear increase in the prediction error of a classifier. Thus, saying that the function structure is now twice as complex means that the prediction error ε of a classifier is twice as high. The following equation summarizes the assumptions and describes the relation of prediction error, function structure, and classifier size

$$\varepsilon = \Gamma \sqrt[n]{V}. \tag{18}$$

Given the target error ε_0 we can now derive the *optimal classifier volume*, that is, the maximal condition size, where the classifier is accurate. Thus, we set $\varepsilon = \varepsilon_0$, solve for the classifier width, and raise to the power n, that is the dimension of the function f_Γ

$$V_{\text{opt}} = \left(\frac{\varepsilon_0}{\Gamma}\right)^n. \tag{19}$$

Since the input space is confined to $[0, 1]^n$, the volume to be covered is 1. Thus, the *minimal population size* is the multiplicative inverse of the optimal classifier volume, given by

$$N_{\text{min}} = \left(\frac{\Gamma}{\varepsilon_0}\right)^n. \tag{20}$$

Basically, Eq. (19) states that an increase in the dimensionality requires an exponential decrease in the classifier volume and an increase in the difficulty of the function, modeled by Γ, requires a decrease in the classifier volume that is polynomial in the number of dimensions. The inverse formulation holds for the minimal population size. Analogous to the function structure, the target error ε_0 has a polynomial influence on classifier volume and population size.

The equations are intentionally simple, hiding a complex geometric problem in the function structure, which is modeled by Γ. For example, consider a four-dimensional sinusoidal function that is approximated by piecewise polynomial predictors embedded in rotated hyperellipsoidal conditions. The geometric problem to be solved gets quite complex, even for simple functions, because the problem depends on the function, the condition, and the prediction type used and has to be solved for any combination all over again. In the following, we show how the equations may be used to derive optimal volume and minimal population size for linear functions, approximated using non-rotating rectangular conditions and constant predictions. Although linear functions are not of interest to the LCS community, using constant predictions makes this a worst case scenario for XCSF concerning the population size and can be considered an abstract exemplary setup. Furthermore, the scalability model can be thoroughly tested using empirical experiments.

5.3. Example: linear functions, rectangular conditions, and constant predictions

Consider a linear function

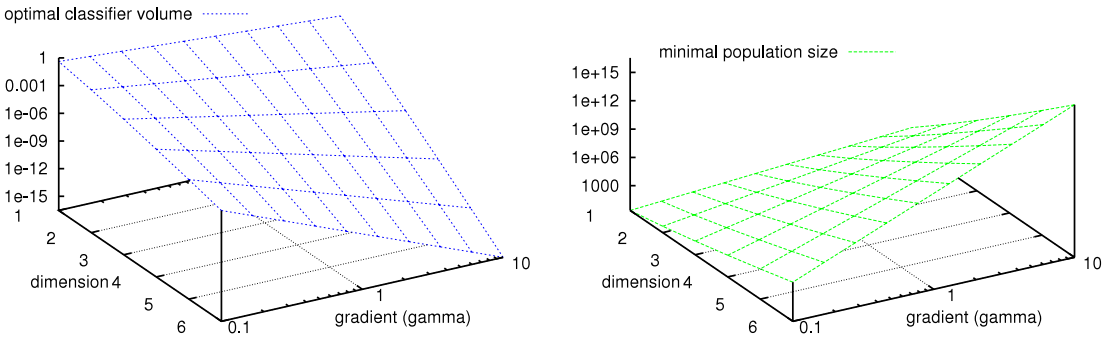
$$\begin{aligned} f_\gamma : [0, 1]^n &\rightarrow \mathbb{R} \\ f_\gamma(x_1, \ldots, x_n) &= \gamma \sum_{i=1}^{n} x_i, \end{aligned} \tag{21}$$

where γ specifies the gradient and, thus, models the function structure Γ consistent with the assumptions made. Since XCSF is equipped with constant predictions only, an increasing gradient requires more classifiers to accurately approximate f_γ. We recall that XCSF faces a checkerboard problem and the optimal solution is a patchwork of small hypercube-shaped classifiers with the constant prediction value equal to the function value at the center of the classifier.

Apparently simple, the calculation of the average prediction error for one classifier gets rather complicated for three or more dimensions. However, numerical integration yields sufficient approximations of the average prediction error ε of a classifier with volume $V = 1$, given gradient γ and dimension n. Using Eq. (18), the prediction error then equals Γ and we can visualize Eqs. (19) and (20) for the linear case. Fig. 2 shows the optimal volume and the minimal population size as a function of the gradient γ in a log–log-fashion for several dimensions. Consequently, the polynomials are linear functions on the plots. The target error was set to the value $\varepsilon_0 = 0.01$, which is also used for the experiments in the next section, where the empirical validation of the stated scalability model proves its viability.

6. Experimental validation

In order to confirm our theoretical considerations, we investigate XCSF's evolved classifier volumes and population size requirements on the linear function f_γ with varying gradients (function structure) from one up to six dimensions. Therefore, we equip XCSF with constant predictions, rectangular conditions, target error $\varepsilon_0 = 0.01$, and a learning time of 500,000



(a) Optimal classifier volume. (b) Minimal population size.

Fig. 2. Theoretical model of the (a) classifier volume and (b) minimal population size, both depending on the dimension as well as the function structure (that is, the gradient γ). The graphs are depicted with $\varepsilon_0 = 0.01$. The axis showing the gradient as well as the vertical axis are log-scaled for better readability.

iterations.[3] The condensation algorithm [15] with "closest classifier matching" [10] is applied during the final 10% of each run to reduce the final population size. To find the minimal population size sufficient to reliably learn the problem, a bisection algorithm (see Appendix) controls the maximum population size parameter of XCSF and varies this parameter until the minimal population size is found. We assume that XCSF has reliably learned the problem if the average prediction error (averaged over 1000 samples) in ten independent runs is below the target error ε_0 for the final 50,000 iterations, that is, the whole duration of the condensation.

When the bisection algorithm terminates, that is, when the minimal population size to reliably learn the function is found, we take a snapshot of the final population and extract the following values: (1) the average classifier volume in the final population, (2) the maximal population size N available during learning, and (3) the final population size, that is, the number of distinct classifiers after condensation. The optimal volume and the minimal population size is calculated using the scalability model (see Section 5.3) and is compared against the empirical results.

6.1. Experiments

Since the computational complexity grows linearly with the population size and polynomially with increasing dimensions, the experiments take a considerable amount of time for high-dimensional functions and thus larger population sizes. In order to reduce the run time, we progressively increased the gradient in the bisection runs within one dimension exponentially, resulting in fewer experiments with large gradients. Furthermore, we used Eq. (20) and solved for γ with the number of classifiers equal to 100 for the initial gradient value and 1000 for the most demanding experiment per dimension. In this way, XCSF should require a reasonable population size although we do not expect XCSF to reach the theoretically determined minimum. Consequently, we can plot the same output (complexity) range for all plots while only adjusting the gradient of the function, that is, the problem difficulty itself—making the plots directly comparable across dimensions.

The results are visualized using log–log-scaled plots (both the vertical and horizontal axis), where the horizontal axis denotes the gradient γ and the vertical axis denotes classifier volume as well as maximum population size during learning and the final size after condensation. The expected polynomial scaling $a \cdot \gamma^n$ for increasing gradient in the regular space is transformed to a line in the shown log–log-plots. Fig. 3 contains six such log–log-plots, one for each dimension from $n = 1$ to $n = 6$, where the theoretical optimum for volume and population size is depicted as a line and the results of the bisection are plotted as crosses. The plots clearly show that empirical data and scalability theory are almost perfectly parallel, confirming that the polynomial order of the model fits reality. Furthermore, concerning the population size there is a small gap (factor 1.8986 ± 0.3784 for all dimensions[4]) from theoretical minimal population size to the actual number of macro classifiers after condensation and a larger gap (factor 17.5748 ± 2.3855) from theory to the maximum number of classifiers during learning. Finally, concerning the classifier volume, the average volume in the final population seems to increase with higher dimensions. For the first dimension the data is nearly perfectly aligned with the theory. For $n = 6$, the empirical results are slightly larger than the theory predicts and not perfectly but still closely aligned.

6.2. Discussion

The empirical results validate the scalability model and, furthermore, suggest that XCSF uses its resources almost optimally. However, there is a constant gap from theoretical population size to the actual condensed size and to the

[3] Other parameters were set as in [10]: $\beta = 0.1, \alpha = 1, \delta = 0.1, \nu = 5, \chi = 1, \mu = 0.05, r_0 = 1, \theta_{GA} = 50, \theta_{del} = 20, \theta_{sub} = 20$. GA subsumption was applied. Uniform crossover was applied.

[4] Mean and standard deviation of the averaged overhead per dimension, that is $\frac{1}{10}\sum_{i=1}^{10} N_i'/N_i$, where 10 is the number of samples per dimension, N_i' is the actual population size of the i-th sample and N_i the corresponding theoretical minimum.

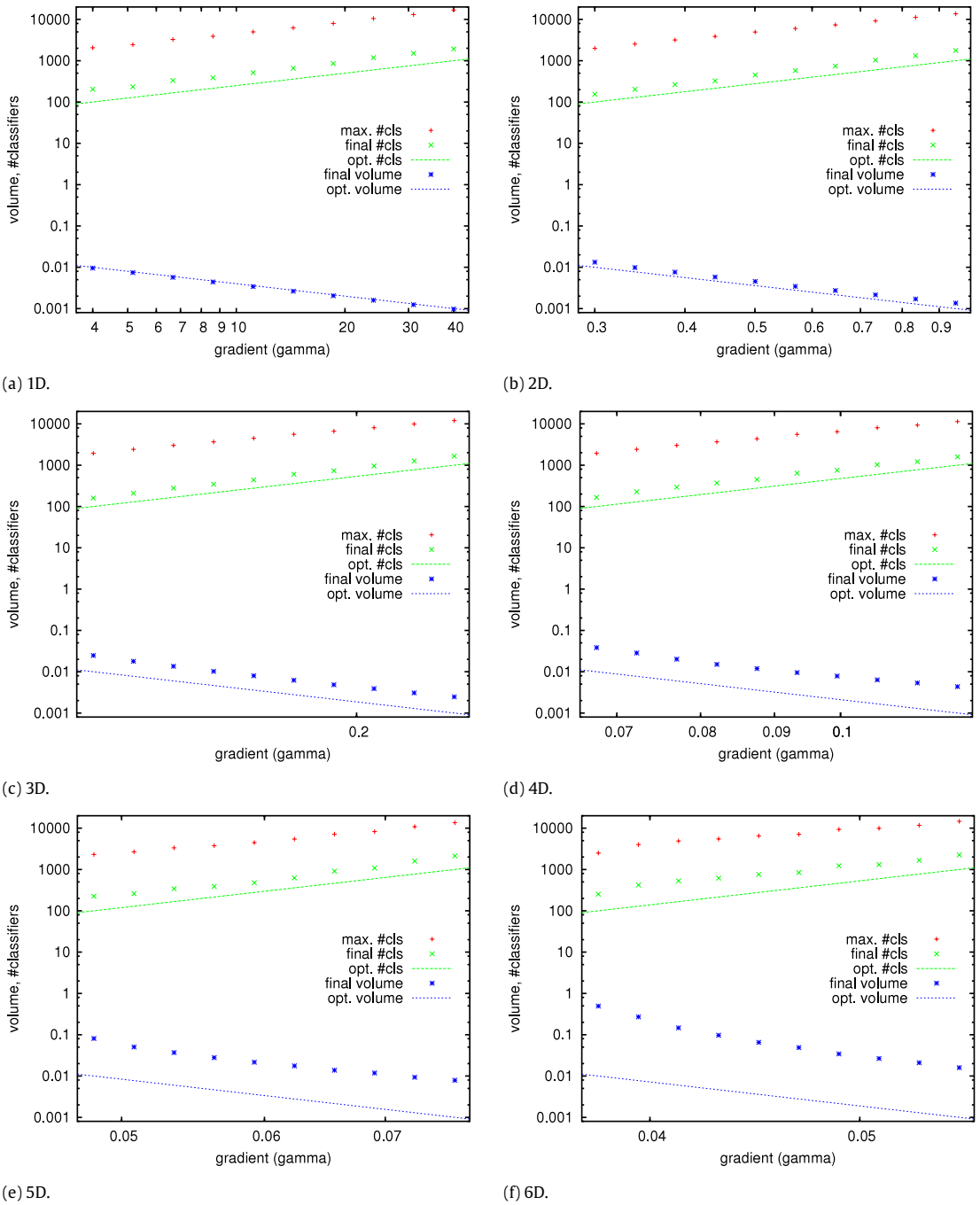


Fig. 3. The maximal population size during learning as well as final population size after condensation and classifier volumes for dimensions one to six in log–log-scaled plots. Crosses represent empirical data from ten independent runs, respectively, and lines represent the developed theory. The same vertical range is depicted for all dimensions, while the horizontal range is set such that theoretical population size ranges from 100 to 1000.

maximum size. Additionally, there is an increasing distance from theory concerning the classifier volume. This section addresses these topics.

Final population size. After condensation is applied for the final 10% of the learning time, there still is a constant gap of around 1.9, that is, the final number of classifiers is approximately 1.9 times higher than the minimal population size given in Eq. (20). There are at least two factors that cause this difference. First and most notably the scalability model assumes a non-overlapping perfect patchwork of classifiers. Furthermore, the condensation algorithm is not a statistical procedure but a heuristic one based on the selection as specified by the genetic algorithm. Consequently, the overlap as well as the

imprecision of condensation are represented by a factor of 1.9—which is quite low especially when considering that this holds for all investigated dimensions.

Maximum population size. The maximum population size is the number of classifiers available for XCSF during learning. Since evolution takes place in the population, there is a considerable number of classifiers which are recently inserted—due to delayed fitness evaluation these classifiers must remain untouched by the genetic algorithm as well as deletion until a reliable fitness value has been determined. Since the environmental feedback only reflects one point of the input space per iteration, this is a slow process and the genetic algorithm is only applied to classifiers that have seen more than $\theta_{GA} = 50$ inputs. Furthermore, the sophisticated deletion mechanism assigns lower deletion probability to inexperienced classifiers. In sum, the evolution requires a considerable portion of extra classifiers for reliable fitness estimates. Thus the multiplicative factor of about 17.6 describes the average overhead for evolutionary search. In other words, during learning XCSF requires approximately 17.6 times the theoretical minimum population size as specified in Eq. (20).

Final classifier volume. We observe increasing distance of the classifier volume from the theory in higher dimensions. A higher dimension increases the search space significantly, but we have a fixed learning time for all experiments. Thus, the evolutionary algorithm has more difficulties to approach the global optimum—resulting in fewer *optimal* classifiers (accurate and maximally general) in the population. However, the final prediction is governed by accurate classifiers and the influence of inaccurate classifiers is strongly reduced (polynomial order $\nu = 5$). Thus, the average prediction error is much lower than Eq. (19) would suggest, when looking at the average classifier volume in the final population. In contrast, we plot the average volume of all classifiers independent of their accuracy. Investigation of the final populations revealed that the average prediction error is indeed slightly below the target error although some oversized classifiers remain in the population.

7. Structure alignment

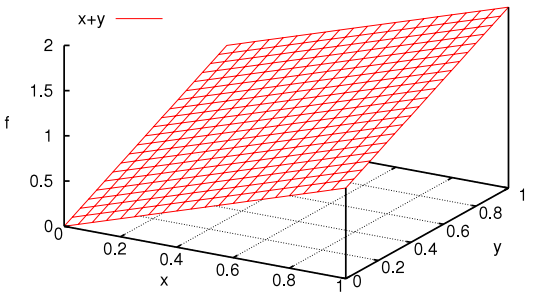
The results validate the *scalability theory*. However, a somewhat worst case was modeled, where XCSF faces a checkerboard problem and no substructures can be exploited. In other words, XCSF's solution representation cannot reflect regularities in the function due to a lack of representational power. Concerning the exemplary linear problem, constant predictions and interval conditions are not suitable to approximate the function, while linear predictions would make the problem trivial. More complex functional mappings are expected to include interdependencies between dimensions so that each dimension cannot be considered independently in the general case. The oblique dependencies require either the predictor to reflect the substructure including interdependencies or the condition structure to allow for an oblique clustering of the input space, such that predictions can be aligned in parallel (or perpendicular) to the dependency-axis. Otherwise the worst case scenario comes to life, where the target error can only be met with a fine-grained patchwork of classifiers.

Consider the linear function $f_\gamma(x_1, \ldots, x_n) = \gamma \sum_{i=1}^{n} x_i$ from our previous experiments. Obviously, a linear predictor would leave the LCS obsolete, since one classifier that covers the complete input space would quickly approach the optimal solution. Lets assume that a constant prediction is necessary for some reason. Taking a closer look at the structure, constant subspaces (perfectly reflected by the predictor) are found perpendicular to the $(1, 1, \ldots, 1)^T$ axis. Consequently, rotation in the rectangular conditions enables XCSF to exploit those constant subspaces. In fact, this changes the problem complexity (in terms of solution size) from polynomial order n to linear complexity. Fig. 4 shows the two-dimensional function f_γ and the difference between non-rotating and rotating conditions with respect to the population size and condition structure. Without rotation the problem is quadratic for $n = 2$, whereas a linear increase in population size shows that rotation boosts the performance drastically.

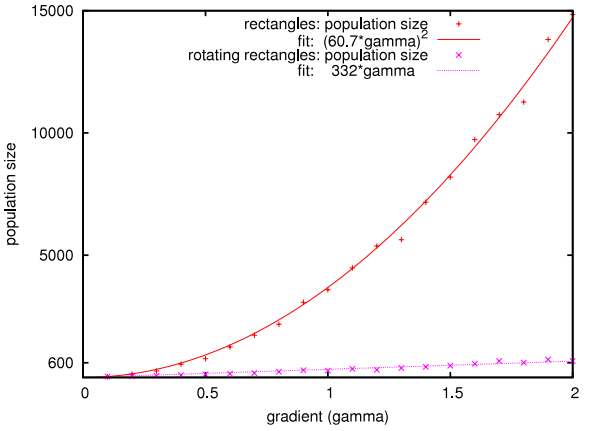
Although rotation in the condition structures makes the search space for the genetic algorithm quadratic in the number of dimensions, eventually increasing the learning time, the overall benefit of a drastically reduced population size outweighs this drawback. This way, the rotation makes the condition structure more general and, thus, more powerful. Furthermore, rotated classifiers may give a hint on oblique principal components of the underlying problem. Rotating condition structures have shown to solve tasks (where other condition structures would require vast population sizes), such as the oblique sine function in multiple dimensions [10] or an application to robot arm control [7,8]. Other algorithms also exploit rotation: For example, the CMA evolution strategy [32] uses rotation to align mutations to the fitness gradient in the search space—effectively speeding up the evolutionary search in problems where the fitness gradient is not parallel to the problem axis.

8. Summary and conclusions

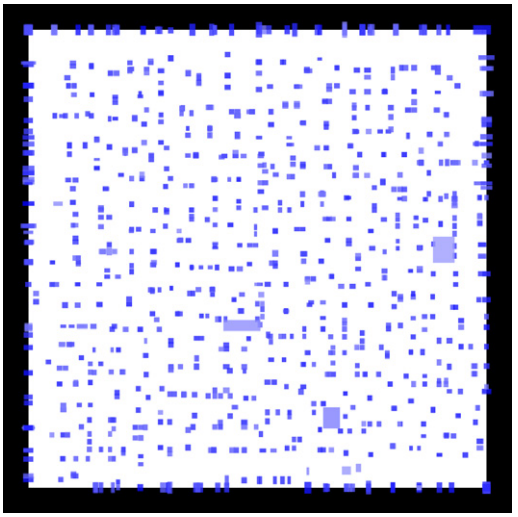
The present work extends available theory on the Learning Classifier System XCSF in several consecutive steps. First, it was shown *how* the system works by decomposing its performance factors in three categories: (1) the function complexity, that is, dimensionality, structure, and sampling, (2) representational power, that is, the condition and prediction structures, and (3) learning capacity, that is population size, learning time, and target error. The interactions within these categories give a rough idea what *difficulty* means in XCSF's terms. Second, it was shown *when* the system works by deriving the minimal requirements for evolution and learning to take place at all as well as a learning time approximation and a solution sustenance bound. Third, building upon that knowledge, a *scalability model* for the population size was derived, where a checkerboard problem served as a worst case scenario.



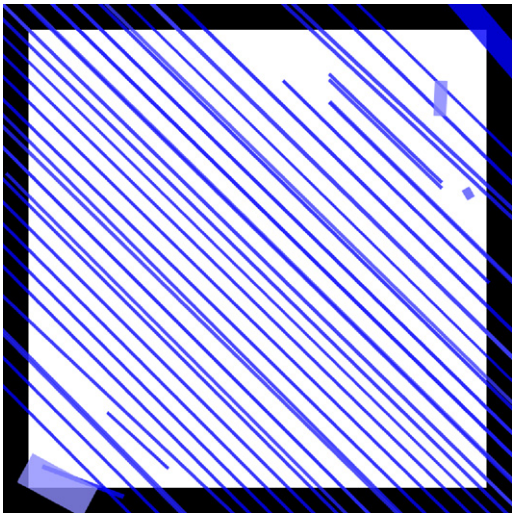
(a) Two-dimensional f_γ with $\gamma = 1$.



(b) Required population size.



(c) Rectangular conditions.



(d) Rotating rectangular conditions.

Fig. 4. (a) Two-dimensional f_γ, depicted with $\gamma = 1$. Note the constant subspaces perpendicular to $x = y$. (b) The required population size, as returned by the bisection algorithm with non-rotating and rotating rectangular conditions. Data points are fitted using the nonlinear least-squares Marquardt–Levenberg algorithm. (c) Clustering of the input space by rectangular conditions, depicted with 20% of their actual size for better readability. (d) Same graphic for rotating conditions.

In particular, the relations between function complexity, target error, and population size were modeled. In sum, the population size depends exponentially on the dimensionality of the problem, polynomially on the problem difficulty — which is defined via the prediction error — as well as the target error. A simple example showed how the model can be used to approximate the minimal population size for a problem, given the problem structure as well as XCSF's condition and prediction structure. Using this example, the model was empirically validated for up to six-dimensional functions, proving its correctness and predictive capabilities. The experiments show that XCSF only requires a small constant overhead in the final solution due to overlap of classifiers, even in higher dimensions. This overhead is considerably larger during learning in order to allow for evolutionary search, but still obeys the derived complexity expectations. Thus, XCSF uses its resources almost optimally.

Finally, having modeled the worst case scenario, it was discussed how *structure alignment* can drastically reduce the problem complexity by enabling the alignment of XCSF's solution representation to principal components of the underlying problem. A final example showed that rotating conditions can effectively turn a polynomial problem into a linear one by exploiting and generalizing dimensional interdependencies. Thus, structure alignment does not only boost XCSF's learning performance, but it also improves its expressive capabilities.

In conclusion, the theoretical study of XCSF confirms that the system is a highly competent learning algorithm that is able to yield optimal performance with constant overhead for a large range of function approximation problems. The solution representations evolved by XCSF are formed out of local predictors—usually partially overlapping locally linear predictors. The solution representation is thus very close to the Locally Weighted Projection Regression (LWPR) algorithm [33,34]. The strongest difference is that the shape and distribution of the receptive fields, that is, the condition structures, are evolved by

a genetic algorithm rather than by statistical techniques, which depend on the error signal from the prediction side. Thus, especially in problems in which the condition structure cannot be directly inferred from the prediction input, as illustrated in a recent robot control application [7,8], XCSF is expected to yield superior results and to be applicable to a wider range of approximation problems. In particular, especially problems in which a given context needs to be clustered to enable the formation of accurate predictions given separate inputs — such as context-dependent effects of motor activity — may constitute one of the most rewarding problem class for XCSF. The theory developed in this paper also confirms theoretically that it is very much worth pursuing this research direction in further detail.

Acknowledgements

The authors acknowledge funding from the Emmy Noether program (German Research Foundation, DFG, BU1335/3-1) and thank the COBOSLAB team.

Appendix. Bisection algorithm

First, the bisection algorithm increases the population size until the given problem is learned. Next, the interval $(l, h]$ is iteratively narrowed, where XCSF is able to learn the function with a population size of h, but fails with l. The algorithm terminates, when the interval size $h-l$ is less than or equal to 0.05 times the upper border, thus realizing a relative precision of 5%.

$h = 2$
// increase population size until function is learned
while not learned **do**
  $h = h * 2$
  learned = XCSF.run(h)
end while
$l = h/2$
// narrow interval $(l, h]$, where l fails and h succeeds
while $(h - l) > (0.05 * h)$ **do**
  $m = l + ((h - l)/2)$ // binary search
  learned = XCSF.run(m)
  if learned **then**
    $h = m$
  else
    $l = m$
  end if
end while
return h

References

[1] D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley Publishing Company, Reading, MA, 1989.
[2] J.H. Holland, J.S. Reitman, Cognitive systems based on adaptive algorithms, SIGART Bulletin 63 (63) (1977) 49.
[3] S.W. Wilson, Classifier fitness based on accuracy, Evolutionary Computation 3 (2) (1995) 149–175.
[4] M.V. Butz, D.E. Goldberg, P.L. Lanzi, Gradient descent methods in learning classifier systems: Improving XCS performance in multistep problems, Tech. rep., Illinois Genetic Algorithms Laboratory (2003).
[5] E. Bernadó-Mansilla, J.M. Garrell-Guiu, Accuracy-based learning classifier systems: Models, analysis, and applications to classification tasks, Evolutionary Computation 11 (2003) 209–238.
[6] M.V. Butz, Rule-Based Evolutionary Online Learning Systems: A Principal Approach to LCS Analysis and Design, Springer-Verlag, 2006.
[7] M.V. Butz, O. Herbort, Context-dependent predictions and cognitive arm control with XCSF, in: GECCO '08: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, ACM, New York, NY, USA, 2008, pp. 1357–1364.
[8] M.V. Butz, G.K. Pedersen, P.O. Stalph, Learning sensorimotor control structures with XCSF: Redundancy exploitation and dynamic control, in: GECCO '09: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, 2009, pp. 1171–1178.
[9] S.W. Wilson, Classifiers that approximate functions, Natural Computing 1 (2002) 211–234.
[10] M.V. Butz, P.L. Lanzi, S.W. Wilson, Function approximation with XCS: Hyperellipsoidal conditions, recursive least squares, and compaction, IEEE Transactions on Evolutionary Computation 12 (2008) 355–376.
[11] S.W. Wilson, Get real! XCS with continuous-valued inputs, in: Learning Classifier Systems, From Foundations to Applications, in: LNAI, vol. 1813, Springer-Verlag, 2000, pp. 209–219.
[12] C. Stone, L. Bull, For real! XCS with continuous-valued inputs, Evolutionary Computation 11 (3) (2003) 299–336.
[13] S.W. Wilson, Classifier conditions using gene expression programming, in: Learning Classifier Systems, 10th International Workshop, IWLCS 2006, Seattle, MA, USA, July 8, 2006, and 11th International Workshop, IWLCS 2007, London, UK, July 8, 2007, Revised Selected Paper, in: Lecture Notes in Artificial Intelligence, Springer-Verlag, 2008, pp. 206–217.
[14] P.L. Lanzi, D. Loiacono, S.W. Wilson, D.E. Goldberg, Prediction update algorithms for XCSF: RLS, Kalman filter, and gain adaptation, in: GECCO '06: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, ACM, New York, NY, USA, 2006, pp. 1505–1512.
[15] S.W. Wilson, Generalization in the XCS classifier system, in: Genetic Programming 1998: Proceedings of the Third Annual Conference, 1998, pp. 665–674.
[16] P.O. Stalph, M.V. Butz, D.E. Goldberg, X. Llorà, On the scalability of XCS(F), in: GECCO '09: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, ACM, New York, NY, USA, 2009, pp. 1315–1322.

[17] A. Orriols-Puig, E. Bernadó-Mansilla, D.E. Goldberg, K. Sastry, P.L. Lanzi, Facetwise analysis of XCS for problems with class imbalances, IEEE Transactions on Evolutionary Computation 13 (2009) 1093–1119.
[18] P.L. Lanzi, D. Loiacono, S.W. Wilson, D.E. Goldberg, Generalization in the XCSF classifier system: Analysis, improvement, and extension, Evolutionary Computation 15 (2) (2007) 133–168.
[19] J. Drugowitsch, A. Barry, A formal framework and extensions for function approximation in learning classifier systems, Machine Learning 70 (2008) 45–88.
[20] J.H. Holland, Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence, The MIT Press, Cambridge, MA, 1992.
[21] D.E. Goldberg, Real-coded genetic algorithms, virtual alphabets, and blocking, Complex Systems 5 (1991) 139–167.
[22] A.H. Wright, Genetic algorithms for real parameter optimization, in: Foundations of Genetic Algorithms, Morgan Kaufmann, 1991, pp. 205–218.
[23] N.J. Radcliffe, Equivalence class analysis of genetic algorithms, Complex Systems 5 (1991) 183–205.
[24] N.J. Radcliffe, Forma analysis and random respectful recombination, in: ICGA, 1991, pp. 222–229.
[25] H. Mühlenbein, D. Schlierkamp-Voosen, Predictive models for the breeder genetic algorithm - I. Continuous parameter optimization, Evolutionary Computation 1 (1993) 25–49.
[26] P.A.N. Bosman, D. Thierens, Numerical optimization with real-valued estimation-of-distribution algorithms, in: Scalable Optimization via Probabilistic Modeling, in: Studies in Computational Intelligence, vol. 33/2006, Springer-Verlag, Berlin Heidelberg, 2006, pp. 91–120 (Chapter 5).
[27] H.-G. Beyer, H.-P. Schwefel, Evolution strategies — a comprehensive introduction, Natural Computing 1 (1) (2002) 3–52.
[28] D. Thierens, D.E. Goldberg, A.G. Pereira, Domino convergence, drift, and the temporal-salience structure of problems, in: Proceedings of the 1998 IEEE World Congress on Computational Intelligence, IEEE Press, New York, NY, 1998, pp. 535–540.
[29] M.V. Butz, D.E. Goldberg, P.L. Lanzi, K. Sastry, Problem solution sustenance in XCS: Markov chain analysis of niche support distributions and the impact on computational complexity, Genetic Programming and Evolvable Machines 8 (2007) 5–37.
[30] T. Kovacs, M. Kerber, What makes a problem hard for XCS?, in: P.L. Lanzi, W. Stolzmann, S.W. Wilson (Eds.), Advances in Learning Classifier Systems, in: Lecture Notes in Computer Science, vol. 1996/2001, Springer-Verlag, Berlin Heidelberg, 2001, pp. 251–258.
[31] C. Stone, L. Bull, An analysis of continuous-valued representations for learning classifier systems, in: L. Bull, T. Kovacs (Eds.), Foundations of Learning Classifier Systems, in: Studies in Fuzziness and Soft Computing, Springer-Verlag, 2005, pp. 127–175.
[32] N. Hansen, A. Ostermeier, Completely derandomized self-adaptation in evolution strategies, Evolutionary Computation 9 (2) (2001) 159–195.
[33] S. Vijayakumar, S. Schaal, Locally weighted projection regression: an O(n) algorithm for incremental real time learning in high dimensional space, in: ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000, pp. 1079–1086.
[34] S. Vijayakumar, A. D'Souza, S. Schaal, Incremental online learning in high dimensions, Neural Computation 17 (2005) 2602–2634.