Lecture Notes in Artificial Intelligence 4998

Edited by R. Goebel, J. Siekmann, and W. Wahlster

Subseries of Lecture Notes in Computer Science

Jaume Bacardit
Ester Bernadó-Mansilla
Martin V. Butz Tim Kovacs
Xavier Llorà Keiki Takadama (Eds.)

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 Papers

Springer

Series Editors

Randy Goebel, University of Alberta, Edmonton, Canada
Jörg Siekmann, University of Saarland, Saarbrücken, Germany
Wolfgang Wahlster, DFKI and University of Saarland, Saarbrücken, Germany

Volume Editors

Jaume Bacardit
University of Nottingham, UK
E-mail: jaume.bacardit@nottingham.ac.uk

Ester Bernadó-Mansilla
Universitat Ramon Llull, 08022 Barcelona, Spain
E-mail: esterb@salle.url.edu

Martin V. Butz
University of Würzburg, 97070 Würzburg, Germany
E-mail: butz@psychologie.uni-wuerzburg.de

Tim Kovacs
University of Bristol, Bristol BS8 1UB, UK
E-mail: kovacs@cs.bris.ac.uk

Xavier Llorà
University of Illinois at Urbana-Champaign, Urbana, IL 61801-2996, USA
E-mail: xllora@uiuc.edu

Keiki Takadama
Tokyo Institute of Technology, Tokyo, 152-8550, Japan
E-mail: keiki@hc.uec.ac.jp

Library of Congress Control Number: 2008938336

CR Subject Classification (1998): I.2, H.3, D.2.4, D.2.8, G.1.6

LNCS Sublibrary: SL 7 – Artificial Intelligence

ISSN 0302-9743
ISBN-10 3-540-88137-9 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-88137-7 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12539123 06/3180 5 4 3 2 1 0

Preface

This volume includes extended and revised versions of the papers presented at the 9th and 10th International Workshops on Learning Classifier Systems (IWLCS 2006 and IWLCS 2007). Both workshops were held in association with the Genetic and Evolutionary Computation Conference (GECCO). IWLCS 2006 was held on July 8th, 2006, in Seattle, USA, during GECCO 2006. IWLCS 2007 was held on July 8th, 2007, in London, UK, during GECCO 2007.

The IWLCS is the annual meeting of researchers wishing to discuss recent developments in learning classifier systems (LCS). At the last IWLCS, the LCS researchers commemorated the 10th anniversary of the workshop and acknowledged the contribution of Stewart Wilson to the field. Following his proposal of the XCS classifier system in 1995, research on LCS was reactivated leading to significant contributions and promising perspectives. The annual IWLCS workshops are the proof of this fruitful research. We include an invited paper from Stewart Wilson. We greatly appreciate his contribution to the volume.

The contents of this book are as follows. First, Bacardit, Bernadó-Mansilla and Butz review LCS research over the past ten years and point out new challenges and open issues in the LCS field. Next, papers investigating knowledge representations are presented. Lanzi et al. analyze the evolution of XCS with symbolic representations using a novel method that identifies useful substructures and tracks the emergence of optimal solutions. Ioannides and Browne investigate the scaling of LCSs using ternary and symbolic representations. Orriols-Puig et al. introduce a fuzzy representation to UCS classifier system and show promising results regarding interpretability and accuracy. The next group of papers provides insight into the system's functioning. Drugowitsch and Barry propose a methodology for designing LCSs based on first identifying the model underlying a set of classifiers and then training the model with standard learners. Orriols-Puig and Bernadó-Mansilla revisit the UCS classifier system and compare it with XCS in several critical scenarios. Next, several papers investigating LCS mechanisms are provided. Loiacono et al. analyze current methods for error estimate in XCSF and propose the Bayes linear analysis to improve this estimate. Smith and Jiang introduce a new LCS architecture with mutual information-based fitness and compare the approach with XCS. Bull designs a model of a simple anticipatory LCS with payoff-based fitness to understand the main learning principles. Mellor describes a learning classifier system that evolves rules expressed as definite clauses over first-order logic and applies this to relational reinforcement learning tasks. Llorà et al. study a Pittsburgh LCS that identifies linkages among the attributes of the problem. The following section of papers explores future directions for LCSs. Wilson investigates the representation of classifier conditions based on gene-expression programming in order to get a better fit with regard to environmental regularities than is the case

with traditional representations. Lanzi et al. design an ensemble XCSF which contains classifiers with different types of prediction functions in the same population, allowing thus the evolution to choose the best representation according to the underlying geometry of the problem. Next, Orriols-Puig et al. design a learning methodology that first infers learning interactions among features and then builds a surrogate model that is used for classification. Bacardit and Krasnogor investigate ensemble techniques for Pittsburgh LCSs and test the approach on a bioinformatics domain. Finally, applications of LCSs are presented. Kurahashi and Terano apply an LCS based on the minimum description length principle and Tabu search to extract plant operation knowledge from time series data. Moioli et al. focus on the application of LCSs to reactive and non-reactive tasks, and compare the performance of a strength-based and an accuracy-based LCS.

This book is the continuation of the five volumes from the previous workshops, published by Springer-Verlag as LNAI 1813, LNAI 1996, LNAI 2321, LNAI 2661, and LNCS 4399. We hope it will be a useful support for researchers interested in learning classifier systems and will provide insights into the most relevant topics and the most interesting open issues. We are grateful to all authors that contributed to this volume as well as to the referees that helped improve the quality of these papers.

May 2008

Jaume Bacardit
Ester Bernadó-Mansilla
Martin V. Butz
Tim Kovacs
Xavier Llorà
Keiki Takadama

Organization

The 9th International Workshop on Learning Classifier Systems (IWLCS 2006) was held on July 8th, 2006, in Seattle, USA, in association with the Genetic and Evolutionary Computation COnference (GECCO) 2006. The 10th International Workshop on Learning Classifier Systems (IWLCS 2007) was held on July 8th, 2007, in London, UK, during GECCO 2007.

Organizing Committee

IWLCS 2006

Tim Kovacs	University of Bristol, UK
Xavier Llorà	University of Illinois at Urbana-Champaign, USA
Keiki Takadama	Tokyo Institute of Technology, Japan

IWLCS 2007

Jaume Bacardit	University of Nottingham, UK
Ester Bernadó-Mansilla	Universitat Ramon Llull, Spain
Martin V. Butz	Universität Würzburg, Germany

Program Committee

Jaume Bacardit	University of Nottingham, UK
Anthony Bagnall	University of East Anglia, UK
Alwyn Barry	University of Bath, UK
Jeffrey K. Basset	George Mason University, USA
Ester Bernadó-Mansilla	Universitat Ramon Llull, Spain
Andrea Bonarini	Politecnico di Milano, Italy
Lashon B. Booker	The Mitre Corporation, USA
Will Browne	University of Reading, UK
Larry Bull	University of West England, UK
Martin Butz	Universität Würzburg, Germany
Brian Carse	University of West England, UK
Lawrence D. Davis	NuTech Solutions, USA
Jan Drugowitsch	University of Bath, UK
Robert Egginton	University of Bristol, UK
Pierre Gérard	University of Paris, France
Ali Hamzeh	Iran University of Science and Technology, Iran
Francisco Herrera	Universidad de Granada, Spain
John H. Holmes	University of Pennsylvania, USA
Abdollah Homaifar	North Carolina A&T State University, USA
Tim Kovacs	University of Bristol, UK

Pier Luca Lanzi	Politecnico di Milano, Italy
Xavier Llorà	University of Illlinois at Urbana-Champaign, USA
Javier Gómez Marín-Blázquez	Universidad de Murcia, Spain
Drew Mellor	University of Newcastle, Australia
Luis Miramontes-Hercog	Intelisis, China
Jorge Muruzabal	Universidad Rey Juan Carlos, Spain
Albert Orriols-Puig	Universitat Ramon Llull, Spain
Sonia Schulenburg	University of Edinburgh, UK
Olivier Sigaud	Laboratoire d'Informatique de Paris, France
Robert E. Smith	University College London, UK
Wolfgang Stolzman	Daimler Chrysler, Germany
Keiki Takadama	Tokyo Institute of Technology, Japan
Atsushi Wada	Advanced Telecommunications Research Institute, Japan
Tom Westerdale	University of London, UK
Stewart W. Wilson	Prediction Dynamics, USA
Zhanna Zatuchna	University of East Anglia, UK

Table of Contents

New Directions

Applications

Learning Classifier Systems: Looking Back and Glimpsing Ahead

Jaume Bacardit[1], Ester Bernadó-Mansilla[2], and Martin V. Butz[3]

[1] ASAP research group, School of Computer Science, Jubilee Campus, Nottingham, NG8 1BB and Multidisciplinary Centre for Integrative Biology, School of Biosciences, Sutton Bonington, LE12 5RD, University of Nottingham, UK
jaume.bacardit@nottingham.ac.uk
[2] Grup de Recerca en Sistemes Intel.ligents, Enginyeria i Arquitectura La Salle, Universitat Ramon Llull, Quatre Camins 2, 08022 Barcelona, Spain
esterb@salle.url.edu
[3] Department of Psychology, University of Würzburg, Röntgenring 11, 97070 Würzburg, Germany
butz@psychologie.uni-wuerzburg.de

Abstract. Over the recent years, research on Learning Classifier Systems (LCSs) got more and more pronounced and diverse. There have been significant advances of the LCS field on various fronts including system understanding, representations, computational models, and successful applications. In comparison to other machine learning techniques, the advantages of LCSs have become more pronounced: (1) rule-comprehensibility and thus knowledge extraction is straightforward; (2) online learning is possible; (3) local minima are avoided due to the evolutionary learning component; (4) distributed solution representations evolve; or (5) larger problem domains can be handled. After the tenth edition of the International Workshop on LCSs, more than ever before, we are looking towards an exciting future. More diverse and challenging applications, efficiency enhancements, studies of dynamical systems, and applications to cognitive control approaches appear imminent. The aim of this paper is to provide a look back at the LCS field, whereby we place our emphasis on the recent advances. Moreover, we take a glimpse ahead by discussing future challenges and opportunities for successful system applications in various domains.

1 Introduction

Learning Classifier Systems (LCSs) are robust machine learning techniques that can be applied to classification tasks [17,6], large-scale data mining problems [81,11], or robot control and cognitive system applications [33,62], among others. The well-established field has its origin in John Holland's work on cognitive systems [59,56], initiated with his seminal book on adaption in natural and artificial systems [58]. Time has seen research on several distinct approaches and paradigms. Two classic examples of these are the Michigan approach [57] versus the Pittsburgh approach [104] and also the strength-based Michigan LCSs [57] versus the more recent accuracy-based Michigan LCS [112].

J. Bacardit et al. (Eds.): IWLCS 2006/2007, LNAI 4998, pp. 1–21, 2008.

Recent years have seen an explosion in quantity and diversity of LCS research. Advances have been made on various frontiers including different condition representations beyond the traditional binary/ternary rules (rules for continuous attributes [81], hyperellipsoids [28], representations based on S-expressions [79,21], etc.), other problem classes (function approximation tasks [77,87], clustering [110]), smarter exploration mechanisms [36,85,10], and various theoretical advances [34,26,92,95].

The main meeting point of the LCS community, the International Workshop on Learning Classifier Systems, celebrated its 10th edition in 2007. This gives us the opportunity to take a look at the evolution of the whole LCS field from a wider perspective. In this chapter, we give an overview of the main areas of LCS research in recent years and which challenges and opportunities are laying ahead. In short, the aim of this chapter is to provide a summary of past, present, and future LCS research.

The chapter is structured as follows. Section 2 concentrates on the past by describing briefly the origins of LCS research, its motivation, development, and first successes. Section 3 surveys present LCS research. It touches on many recent advances, which we categorize along the lines of representation, learning, theory, and application. Section 4 discusses future challenges and opportunities. Based on the state-of-the-art survey, we outline various future research and application directions, which may exploit the LCS strengths and improve their weaknesses. Section 5 summarizes and concludes.

2 LCSs: Types and Approaches

John Holland, the father of LCSs, took a biological point of view and consequently introduced the LCS framework as a *cognitive systems* framework [59,56,57]. Inspired by principles from biology, psychology, production systems, and Darwinian evolution, he designed CSs as systems that evolve production rules in order to convert given input sensations, as well as potentially internal state representations, into useful motor commands. Rules were evaluated by basic reinforcement learning mechanisms—the infamous *bucket-brigade* algorithm [60]—and rule structure evolved by means of genetic alterations and fitness-based selection.

Due to the availability of a recent excellent LCS survey [73], rather than focusing on a historic overview on LCS research, this section gives a short introduction to the basic LCS architecture and the fundamental differences between Pittsburgh and Michigan-style LCSs. It is hoped that this section (1) forms the basis for the rest of this chapter and (2) gives a general introduction to what LCSs are.

2.1 Basic LCS Components

It might be debatable which systems may be considered LCSs. However, in order to get a grasp onto the system functionality, it seems important to identify the minimal components that are usually part of an LCS:

- A set of classifiers, that is, a set of rule-like structures, where rules usually have a condition-prediction form. This set, as it will be seen later when we describe the two main LCS paradigms, is often identified as a *population*, where each classifier in the set has its own individual identity, while other times classifiers are just part of a whole and studying them separately does not always provide good insight. For simplicity in the next paragraphs we will talk about populations, even if it is not entirely appropriate.
- Classifier/population evolution mechanisms, potentially enhanced with heuristics, which are designed to improve rule structures over time.
- Classifier/population evaluation mechanisms, which identify the quality of a rule or population of rules.

These components are specified in a rather general sense. However, the three components immediately imply some of the most important considerations in LCS research and application. First, the population of classifiers implies that LCSs are meant to evolve *distributed* problem solutions, in which individual classifiers specify suitable sub-solutions. Thus, LCS somewhat follow a mixture of experts approach. The overall solution to the problem is thus not represented in an individual rule but in the concert of rules represented in a classifier population.

Second, since rule structures are evolved by evolutionary-inspired, distributed learning techniques and this evolutionary process depends on fitness estimates, which are derived by the employed evaluation mechanism, LCSs are highly *interactive learning mechanisms*. Thus, the interaction—or the race—between sufficiently accurate evaluations and sufficiently focused evolution needs to be balanced to ensure successful learning. The various LCS systems accomplish this in some way or the other, as will be seen below.

Finally, due to the evolutionary-based structural search, LCSs usually work exceptionally competitive in problems in which either the signal for rule structures cannot be determined directly from the feedback signal, or, if a suitable feedback signal is available, directed error-based structural learners tend to get stuck in local minima. Thus, due to the interactive evaluation-evolution approach, LCSs do process feedback signals but they do not convert this signal directly into structural search biases, but use evolutionary mechanisms to induce a more thorough search that is only indirectly dependent on the feedback. Thus, LCSs are more likely to find globally-optimal solutions in particularly challenging problems, which require distributed problem solutions but in which heuristic search mechanisms tend to prematurely converge to local optima.

2.2 Michigan vs. Pittsburgh LCSs

One of the most fundamental distinction in LCS research is that of Michigan-style vs. Pittsburgh-style LCSs. Holland proposed the Michigan-style ones [57], while De Jong and his students [104,105] proposed the Pittsburgh-style LCS approach. A great historical description of the development of these two LCS paradigms is available elsewhere [40]. Several main distinctions between the two approaches can be drawn:

- Individuals structure
- Problem solution structure
- Individuals competition/cooperation
- Online vs. offline learning

The first most fundamental distinction is the structure of an individual. While in Michigan systems each individual is a classifier, in Pittsburgh systems each individual is a set of classifiers. The other distinctions are a consequence of the first one: in Pittsburgh systems the solution to the problem is the best individual of the population. The individuals in the population compete to solve the problem and for reproductive opportunities, while in Michigan systems the solution *is* the population, that is, the classifiers in the population cooperate to solve the problem, while they compete for reproductive opportunities. The last distinction is just a consequence of the previous distinctions, and will be discussed later in this section. Figure 1 illustrates the main difference between the two systems.

Due to the classifier-based competition in Michigan-style LCSs, the population is usually continuously evaluated and evolved by steady-state GA techniques. Pitt-style systems, on the other hand, require longer evaluation periods until the next generation of populations can evolve, since the fitness of the whole population rather than of individual classifiers needs to be assessed. Thus, Michigan-style systems are typically applied in interactive online learning problems while Pitt-style systems are rather suitable for offline learning problems. Nonetheless, either system has also been applied to the other problem type.

Another consequence of the rule-competition vs. population-competition difference is the typical form of the final solution. While Michigan-style systems typically evolve highly distributed problem solutions involving a rather large number of rules (typically hundreds if not more), Pitt-style systems usually evolve more compact populations involving only few rules in a population (less than a hundred). As a consequence, it can be expected that Pitt-style systems are more suitable when compact solutions with few rules are expected to solve the

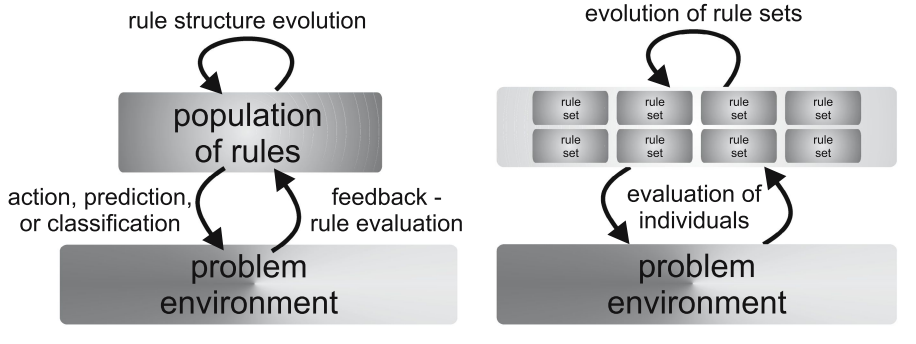


Fig. 1. While Michigan-style LCSs evolve one population of rules, in which the rules compete for offspring generation, Pittsburgh-style LCSs evaluate and evolve multiple populations, which compete with each other for reproductions

problem at hand. Michigan-style systems, on the other hand, are better suited if more distributed solutions are searched for.

3 Recent Advances in LCSs

This section contains an overview of the recent research in the LCS field. Our aim is to provide a spotlight of the different directions towards which the field is advancing. Thus, although our intention is to provide a good description of the overall advances of the field, for a more detailed survey including further historic remarks, the interested reader is referred to the mentioned LCS survey [73].

Classical Michigan/Pittsburgh LCS systems were rule systems with ternary condition structures, discrete actions, and real-valued predictions that used some form of evolutionary component to learn. *Present* LCS research has thoroughly analyzed these representations and mechanisms in several, often facet-wise, theoretical models. Moreover, it has gone beyond these simple representations and is currently investigating the usage of advanced evaluation and evolution mechanisms, advanced representations, and the application to more diverse, real-world problem domains.

This section shows that the current LCS research is very diverse, tackling many different—albeit partially converging—frontiers towards which this field is advancing. We organize this section in ten subsections in which these advances can be placed. Starting from the representation of conditions, actions, and predictions, we move on to classifier competition, the evolutionary component, and theoretic considerations and continue with with issues on solution interpretability and efficiency enhancement techniques. We conclude this section with a short survey on LCS application domains and current cognitive system approaches.

3.1 Condition Structure

In this category we place the advances in the condition part of the knowledge representations, that is, the way in which the feature space is partitioned when a problem is solved. Traditionally, LCSs have used knowledge representations based on rules for binary/nominal attributes. The ternary representation of Michigan LCSs [50,112] or the representation of the GABIL Pittsburgh LCSs [41] are two examples of classic knowledge representations.

Over time, other kinds of knowledge representations were proposed. The main bulk of them were intended to deal with continuous attributes, something that previous representations could not do. The earliest approach for continuous attributes [116] was still a rule representation, but this time using hyperrectangles as the conditions for the classifiers. This approach has been the most popular one in recent years [107,3,83,29,14]. Other alternatives are using rule representations based on fuzzy logic [39], decision trees and synthetic instances used as the core of a nearest neighbor classifier [82], or hyperellipsoid conditions [28,35].

Another kind of representation advance is the use of symbolic expressions to define classifier conditions [78,2,79,21]. This kind of representation may be the most flexible one, in the sense that it can specify the most diverse types of

problem subspaces. However, due this high diversity, it can also be considered one of the hardest to learn suitable condition structures reliably.

3.2 Action Structures

While conditions partition the problem space, actions, or classifications, propose a solution to the specified problem subspace. We analyze here the different alternatives regarding the action part of the classifiers. Traditionally, the responses of LCS systems were static. That is, in a classification problem the possible responses were the different classes of the domain. Each classifier had a static associated class. For multi-step domains, the different responses were the different discrete movements an agent could execute.

A more recent approach goes beyond these discrete action forms by proposing computed actions [74]. In this case, each classifier does not have an associated class label, but a prediction function that computes an action based on the input pattern that matched the classifier. This prediction function can be a linear combination of the inputs with a weight vector. Thus, in LCSs with computed actions, action choice does not only depend on the subspace in which a classifier condition is satisfied, but also on the action computation executed within the specified subspace.

3.3 Prediction Structure

While original LCS rules had a constant prediction, which was updated by gradient-based techniques, different kinds of advanced prediction structures and prediction estimation techniques have been employed recently. First of all, LCSs can be applied to classes of problems beyond classification/multi-step domains. The most prominent of these application domains is that of function approximation/regression tasks [117,28,35,77,87] or clustering [110]. Initial function-approximation LCS approaches tackled the regression problem as a piece-wise linear approximation, where the problem was solved by the cooperation of multiple classifiers, each of which handled a different piece of the feature space, and the continuous output of each classifier was computed as a linear combination of the input and the weight vector of the classifier. Initially, this weight vector was adjusted using a simple delta rule [117,28,35], although recently more sophisticated methods such as Recursive Least Squares or Kalman Filters [77] have been employed. More recently, LCSs have gone beyond linear approximations by also exploring the possibility of using polynomial predictions [75], neural predictions [88], and Support Vector Regression [87].

3.4 Classifier Competition

Given a specific input problem instance, individual rules usually propose one action and prediction. However, since usually many classifier match a certain input, another concern is the selection of the actual action and prediction amongst all the matching classifiers available. That is, given a set of classifiers that match an input pattern, the LCS should choose the classifier/s that produce the response. In the Pittsburgh approach, the traditional solution is to organize the classifiers

in a decision list [101] (an ordered set of rules), and the first classifier in the list that matches an input pattern is the one used to predict its output. In the Michigan approach, the prediction is usually made cooperatively by all the classifiers of the match or action set. Some recent advances in this topic are the usage of explicit default rules at the end of the decision list of Pittsburgh LCSs [8] or the use of better accuracy estimates of classifiers [89,103] and principled classifier voting [19] for Michigan LCSs. Also the usage of ensemble learning methods is worth mentioning, which integrates the collective prediction of a set of models (populations of classifiers in a Michigan LCS [69] or sets of rules extracted from multiple runs of a Pittsburgh LCS [9]) using some principled fashion.

3.5 Rule Structure Evolution Mechanisms

Evolutionary mechanisms explore the space of classifier structures. In simple LCSs, this has been done by simple mutation techniques (random changes in the ternary condition representation) and simple crossover techniques (typically applying two-point crossover). Some of the recent advances, however, noted that such a simple crossover application may be disruptive, consequently applying Estimation of Distribution Algorithms (EDAs) [80], which generate a model of the problem structure and then explore the search space based on this model. There are studies of the application of EDAs for both the Michigan [36] and Pittsburgh [85] approaches. An alternative to EDAs in the context of smarter exploration mechanisms is the integration of local search techniques within an evolutionary algorithm, generally known as Memetic Algorithms [68], with examples for both Michigan [120] and Pittsburgh [10] LCSs. Also, mutation rates have been adjusted using self-adaptive mutation [22,25]. In this case, the search operators do not improve themselves but rather are evolved to become more efficient for the current exploration mechanism.

3.6 Theory and Robustness

There have been various theoretical advancements in LCSs, which gives more detailed explanations of how, why, and when an LCS works. The theoretical advancements may be separated in the analysis of the evolutionary component and the evaluation component of the LCS system.

For the evaluation component, Wilson [115,112] has shown that his ZCS and XCS systems essentially approximate the Q-value function. Drugowitsch and Barry [43] provide an excellent mathematical foundation of the rule evaluation mechanisms in LCSs and particularly their relation to standard machine learning and adaptive filtering techniques, including Kalman filtering. Generally, rule evaluation can be considered a gradient-based, steepest-decent approximation that should adapt the prediction estimation value of a classifier maximally efficiently. Given good approximations, rule evolution can be applied successfully.

On the rule evolution side, the seminal paper on *a Theory of Generalization and Learning in XCS* [34] has shown how the evolutionary component in LCSs picks-up signals of more suitable classifier structures and consequently evolves

those. Due to the strong importance of proper selection pressures, various methods have been investigated, including proportionate selection with different scaling factors [66] as well as tournament selection methods [37]. Selection pressure was explicitly modeled in [96], where tournament selection was found to be more robust than roulette wheel selection.

Meanwhile, generalization applies due to a preference of reproducing more general classifiers. Moreover, this paper has shown that a general basic support of structure needs to be available to ensure successful classifier evolution. [30] has further derived a minimal bound for the population size necessary to evolve boundedly complex classifier structures. Finally, [32] has derived another minimal population size bound that is necessary to ensure complete solution sustenance. All these bounds were used to confirm the PAC-learning capabilities of the XCS classifier system in k-DNF binary problem domains [26]. These theoretical advancements still await their extension into the real-valued realm, in which a volume-based classifier condition representation may lead to similar results.

Moreover, it has been shown that class imbalances pose some difficulties to LCS learners. Generally, learners are usually biased toward the majority class when they are exposed to domains with high class imbalances. LCSs also suffer from these difficulties and can potentially forget infrequent patterns during the incremental learning progress. The conditions necessary for successful learning under such conditions have been theoretically investigated in [92,95]. In these studies, the conditions for the discovery and maintenance of minority-class niches are identified. Also a number of resampling approaches have been experimentally investigated to favor the discovery of infrequent patterns [91,93].

3.7 Interpretability and Compaction

While many efforts have relied on improving accuracy of LCSs, interpretability has also been identified as a relevant issue to get enhanced applicability of LCSs. This is an issue that has bothered both Michigan and Pittsburgh researchers. However, the approaches taken have been different. Pittsburgh LCSs usually include mechanisms for evolving compact rule sets in the search process, for example, by means of using minimum description length principles [7] or multiobjective approaches searching both for accurate and minimal rule sets [13,49,63]. On the contrary, Michigan LCSs cannot include such a direct preference for compact rule sets in the evolutionary search and thus, they usually result in large rule sets. One of the reasons is that LCSs are always performing an exploration process, so that once the evolution is stopped, the rule set contains many inexperienced rules. In such cases, Kovacs [67] suggested the use of a condensation phase, where the GA was disabled to allow the formation of optimal rule sets. On the other hand, in domains with continuous attributes where LCSs use non-discretized representations, LCSs tend to evolve large numbers of rules that consist of many partially overlapping rules that cannot be subsumed during the exploration process. In these cases, compaction algorithms that prune excess rules with minimum loss of accuracy are proposed. The use of compaction algorithms was first proposed by Wilson for XCS with hyperrectangle representation [118] and later studied in [42,97,45,121].

Fuzzy representations have been proposed as an alternative way for getting highly interpretable rule sets. There are a number of approaches using fuzzy representations in Pittsburgh and hybrid LCSs [109,64]. In Michigan LCSs, there were early approaches such as [111,100,18]. Recently, fuzzy representations have been introduced in XCS [38] and later in UCS [94].

3.8 Efficiency Enhancement Techniques

Regarding the methods that alleviate the run-time of LCSs, many alternatives also exist. Some methods apply various kinds of windowing techniques [4] that allow the LCS to use only a subset of the training examples for fitness computation. Various policies exist to choose the training subset and the frequency in which this subset is changed. In [44] a taxonomy of such methods is given.

Parallel implementations of various LCS paradigms exist [82,23,81]. The GALE system [82] is an especially interesting example due to its fine-grained parallel design, where the topology and communications of the parallel model are a direct consequence of the population topology and distribution.

A widely explored efficiency enhancement approach in evolutionary computation is the use of fitness surrogates, that is, *cheap* estimators of the fitness function [65]. This approach has been recently explored within the LCS field [98,86] by constructing fitness surrogates based on an estimated model of the problem structure. Finally, there has also been some work in speeding up the matching operations of classifier conditions for both nominal and continuous representations [84,81] based on the usage of vectorial instructions (SSE, Altivec, etc.) available on modern day microprocessors.

3.9 Applications

With regard to applications, a clear aspect where LCSs have shown to perform competently in comparison to a broad range of machine learning techniques is in data mining tasks [17,6]. Until recently, most of the available datasets were of relatively small size. Now, and mainly thanks to the usage of efficiency enhancement techniques explained in the previous subsection, LCSs have also been applied to much larger real datasets in bioinformatics [108] or biomedical [81] domains, containing hundreds of thousands of instances.

Other real world examples of application of LCSs are the automatic learning of fighter aircraft maneuvers [102], LCSs applied to medical domains [61] or to control problems in a steel hot strip mill [20]. There have also been some studies of the application of LCSs to stream data mining [1,54], where there is a continuous flow of examples arriving at a very fast rate, which requires that LCS learn and produce a prediction in very short time. An overview of recent applications including an extensive bibliography of LCS applications can be found elsewhere [24].

3.10 Cognitive Systems

Since Holland's introduction [59], LCSs have also played an important role in adaptive behavior research and the animat problem—research on the develop-

ment of artificial animals and cognitive robots [113,114]. Moreover, they have shown to be strongly related to reinforcement learning and particular to the problem of online generalization in Markov decision processes [71,115,112]. Recently, various results have shown competitive performance of XCS on benchmark reinforcement learning problems, such as the mountain car problem [76], and various studies have shown that XCS can maintain long reward chains and is able to generalize very well over large problem spaces [31,29]. Thus, LCSs can be considered partially superior alternatives to standard reinforcement learning algorithms and related machine learning approaches. They have the particular advantage that the balance between GA and reward propagation and approximation can be maintained in large problem spaces, consequently learning stable payoff distributions with a highly generalized set of accurate classifiers.

There have also been advances in Partially observable Markov decision processes (POMDP). XCS was enhanced with internal registers and has been shown to consequently evolve emergent internal representations that were able to distinguish aliasing states in the environment [72]. However, the scalability of the taken approach has not been shown and other researchers have tackled the problem with various other LCS approaches, such as Pitt-style policy learners [70] or the AgentP classifier framework, which uses learning heuristics to overcome the POMDP problem [122]. Despite all these efforts, the POMDP problem is far from being solved also in the LCSs realm.

AgentP actually belongs to the class of anticipatory learning classifier systems (ALCS), which form explicit predictions about sensory consequences of actions. These systems contain classifiers that encode condition-action-next state perception triples. Various forms of ALCSs exist including the original ACS [106], the enhanced, online generalizing ACS2 [27], the mentioned AgentP, YCS [47], and MACS [46]. In comparison with policy learners, the systems have the advantage that they learn a predictive model of the environment so that they are able to flexibly adjust their actual behavior by simulating possible future behaviors internally. This can be most effectively done with dynamic programming principles [46] but also partial updates have been investigated in accordance with the Dyna architecture in reinforcement learning [27]. For future research, it seems particularly appealing to extend these systems into real-world domains and to modularize them to be able to efficiently represent distinct but related problem spaces of the environment.

4 Challenges and Opportunities

The near future points to several research challenges and various application opportunities, some of which are also shared with the machine learning community as a whole. Of common interest are issues such as applying learners beyond the traditional classification problems, extracting information from real-world datasets, system scalability, and rule selection. Besides the machine learning relation, though, advanced, modular system designs and resulting applications to complex robotics and cognitive systems tasks, amongst other domains, appear

imminent. In the following, we list the, in our opinion, most promising research directions, including advanced system designs and various application opportunities. At the end of the section, we emphasize the general need in machine learning for system cookbooks, that is, principled methodologies for system applications. For LCSs in particular, the practitioner needs to be further guided to be able (1) to choose the best LCS for the problem at hand and (2) to suitably adjust the chosen LCS to optimally prepare it for the application challenge.

4.1 Problem Structure and LCS Modules

A current opportunity for LCS systems is to exploit their easy knowledge extraction possibilities to extract useful patterns for the integration of unsupervised learning and semi-supervised learning mechanisms. Some approaches have already been proposed such as those building clusters by taking advantage of the generalization capabilities of classifiers [110]. The reverse question needs to be further investigated, though, that is, if the clusters can again be used for the solution of classification problems. The XCS system essentially combines clustering and classification and clusters *for* the generation of accurate classifications.

Semi-supervised data mining approaches, where not all instances are labeled, could benefit from combined clustering plus potential classification approaches. Other frameworks, such as unsupervised learning, may also be exploited by LCSs. Mining association rules can be addressed with LCSs that are searching for the most frequent patterns among the attributes. In these cases, generalization is a key issue, and LCSs are ready to conquer large problem spaces with the appropriate generalization mechanisms. Thus, LCSs are ready to be applied to domains in which partially pure clustering and partially problem space clustering for accurate classifications or predictions are necessary.

So far most LCSs have been flat, processing input and converting that input into a classification, behavior, or prediction. The great problem space structuring capabilities, however, suggest the generation of more modularized and hierarchical LCSs. That is, since LCSs have been shown to be able to automatically identify building-block structures in problem domains [36], it appears imminent that modular LCS systems make these structures explicit, abstract them, and use the abstracted concepts for further processing. A first approach in this direction can be found in this book [99]. Due to the LCS principle of clustering-for-prediction, however, more modular and hierarchical LCSs, which may process subsets of input dimensions, abstract the information, and merge it in higher level structures seem possible. Once such system architectures will be successfully designed, a whole new dimension of LCS systems and successful LCS applications will be at hand.

While such modular, hierarchical LCS systems may be applied to various problem domains, the application in the cognitive systems realm appears most appealing. Over the recent years, research in cognitive neuroscience, psychology, and the mind in general has emphasized two very important aspects of brain structure and functionality: interactive modularity and sensorimotor codes [48,52,53,90,119]. That is, the brain structures sensory and motor information

in various modules, whereby the purpose of these modules is (1) to satisfy motivational modules and (2) to serve control modules for successful behavioral executions. Thus, many sensorimotor codes are found in the brain, which encode the dependence of sensory information on motor commands in various forms and multimodal modules. LCSs can structure sensory information for successful prediction and motor control. Thus, they have the potential to directly develop sensorimotor codes. Once advanced system modularity and further interactivity of LCS systems is realized, then also the interactive modularity of sensorimotor codes may be mimicked. Thus, the design of advanced, cognitive LCSs appears to be within our grasp.

4.2 LCS Cookbook

Many different learning algorithms have been proposed and evaluated experimentally in a number of domains. No difference in LCSs: Various LCSs have been proposed and applied to various problem domains—each of which with some claimed superiorities shown by some evaluations in suitable problems or problem classes. With such a variety of available methods, the practitioner finds it difficult to choose a learner for a given application. Which LCS is better suited for a given problem? Which are the conditions of applicability of an LCS? Although we wish to be able to give exact answers to these questions, it is currently still a big challenge to give precise system design recommendations given a particular problem. In fact, often the trouble already starts at the problem definition itself and particularly the to-be expected problem structures.

Thus, a big challenge is the development of further theoretical understanding of which types of problems exist and which kind of LCS, or learning method in general, is most suited to solve each type of problem. Goldberg has approached this challenge with the definition of boundedly difficult problems in various optimization problem domains [51]. The theoretical performance analyses of the XCS classifier system have moved along a similar vein and identified various problem properties that influence problem difficulty [29]. Another approach works on the categorization of problems by means of geometrical descriptors, such as the separability of classes or the discriminant power of attributes [15,55,12]. These works have identified some features that are critical to the success of learners and can act as predictors of the learners' performance [16]. However, there is much more work needed to further understand the intrinsic characteristics of data and find the key properties relevant for the identification of the most potent learning algorithm.

Another dimension of problem complexity, somewhat easier to describe but nonetheless equally difficult for LCS systems, is the size of the problem. That is, how can we make sure that LCS performance is not degraded when tackling problems of larger sizes. Scalability analysis of LCSs in such domains is still at its beginning, but the theoretical knowledge on different facets of the problem is available [5,29,95]. Thus, compound approaches that tackle all the problem facets in real-world data mining applications are pending.

Further extensions of such investigations in the LCS realm appear in close reach, including further analyses of adversarial LCS problems and further studies

of which features are critical for LCS success. These studies are expected to close the gap between LCSs and other machine learning methods. Moreover, they are expected to lead towards more precise knowledge of the relative strengths and weaknesses of the available learning systems and, in particular, the domains of competence of different LCS learners. On this road, we expect that LCSs and evolutionary approaches for machine learning will be progressively better known and will become accepted in the machine learning community as a whole. Meanwhile, the particular strengths of LCSs will become appreciated, such as learning robustness, versatility due to the availability of several representations and balanced learning influences, and the explanatory power of LCSs.

4.3 Data Mining

Clearly the available applications in the data mining domain have not exploited the full potential of LCSs. Knowledge extraction and exploitation are still at the beginning. Moreover, the systems' flexibility has not been exploited to its fullest.

In particular, it appears that there is still a lot of room for applying LCSs in real-world domains, far beyond studies that are based on the known *toy* problems from the UCI repository. Although the problems from the UCI repository have been labeled "real-world problems", and although they technically mostly are, they do not fully represent the difficulties of real-world data mining applications. LCSs may be applied to mine interesting patterns from very large datasets, containing hundreds of thousands of registers and a great number of attributes, plus all associated complexities altogether (including high instance imbalances, noise, missing information, partially labeled instances, etc).

Searching patterns through datasets structured in some manner, different from the usual plain file, can also be a difficult challenge to any learning scheme. Data may be presented in complex structures. While the traditional form consists of a number of instances, each characterized by a fixed number of attributes with associated class, data is now often presented in more intricate ways. Medical records contain diverse data sets, which were collected from many sources: some may have a variable number of medical tests associated, others may contain results from related tests performed over a variable number of individuals of the same family, etc. Thus, incomplete information, multiple-instance learning, and varied types of data are some of the difficulties that LCSs will need to face.

As we have mentioned in section 3.9, there are some initial examples of LCS application to large-scale real datasets. These examples have shown that, indeed, LCS can be applied successfully to these kind of domains, providing accurate solutions and high explanatory power due to its rule-based representations. However, one question remains unanswered in a broad and systematic sense: how can we guarantee that LCSs are well adjusted when applied to data mining domains? The answer is not simple, but we think that the challenges that we presented in the above subsections are important steps towards this answer, specially the cookbook part. When we are able to (1) determine which LCS modules are best suited for the domain at hand, thanks to all the problem complexity metrics, and (2) know how to parametrize our LCSs appropriately,

by using principled policies derived from the theoretical analysis made from each LCS component, then we will have surpassed an important milestone towards successful advanced LCS applications; and even more importantly, the community will have a sound instruction set of how to get the best of the available LCS technology. While the necessary bits and pieces are available, the whole set is currently still being put together.

5 Conclusions

LCSs have come a long way. The first LCSs were mainly biologically inspired and designed as admittedly simple but flexible adaptive systems. Modern LCS applications focused mainly on the data mining challenge. Over the last decade or so, LCS research has progressed towards a solid system understanding, it has created a theoretical foundation of LCS learning concepts, and it has shown LCS competitiveness in various machine learning challenges.

While there are still challenges to be solved, we believe that these challenges are actual opportunities for future successful research efforts and even potentially groundbreaking system applications. LCSs are ready to solve complex real-world problems in the data mining domain but also in the cognitive systems domain and others. The rest of this book provides a great overview of current research advances and application approaches. Various pointers to further recent literature are available throughout the book. Thus, we hope that these IWLCS post-workshop proceedings once again give a useful overview of current system progresses and encourage further effort along the plotted research directions. Only future research can ultimately verify the apparent opportunities.

Acknowledgments. J.Bacardit acknowledges the support of the UK Engineering and Physical Sciences Research Council (EPSRC) under grant GR/T07534/01. E.Bernadó-Mansilla acknowledges the support of *Enginyeria i Arquitectura La Salle, Universitat Ramon Llull*, and the *Ministerio de Educación y Ciencia* under grant TIN2005-08386-C05-04. M.Butz acknowledges funding from the Emmy Noether program of the German research foundation (grant BU1335/3-1) and likes to thank his colleagues at the department of psychology and the COBOSLAB team.

References

1. Abbass, H.A., Bacardit, J., Butz, M.V., Llora, X.: Online adaption in learning classifier systems: Stream data mining. Technical Report 2004031, Illinois Genetic Algorithms Lab, University of Illinois at Urbana-Champaign (2004)
2. Ahluwalia, M., Bull, L.: A genetic programming-based classifier system. In: Proceedings of the Genetic and Evolutionary Computation Conference, vol. 1, pp. 11–18. Morgan Kaufmann, San Francisco (1999)
3. Bacardit, J., Garrell, J.M.: Analysis and improvements of the adaptive discretization intervals knowledge representation. In: Deb, K., et al. (eds.) GECCO 2004. LNCS, vol. 3103, pp. 726–738. Springer, Heidelberg (2004)

4. Bacardit, J., Goldberg, D., Butz, M., Llorà, X., Garrell, J.M.: Speeding-up pittsburgh learning classifier systems: Modeling time and accuracy. In: Yao, X., Burke, E.K., Lozano, J.A., Smith, J., Merelo-Guervós, J.J., Bullinaria, J.A., Rowe, J.E., Tiňo, P., Kabán, A., Schwefel, H.-P. (eds.) PPSN 2004. LNCS, vol. 3242, pp. 1021–1031. Springer, Heidelberg (2004)
5. Bacardit, J.: Pittsburgh Genetics-Based Machine Learning in the Data Mining era: Representations, generalization, and run-time. PhD thesis, Ramon Llull University, Barcelona, Catalonia, Spain (2004)
6. Bacardit, J., Butz, M.V.: Data mining in learning classifier systems: Comparing XCS with gassist. In: Advances at the frontier of Learning Classifier Systems, pp. 282–290. Springer, Heidelberg (2007)
7. Bacardit, J., Garrell, J.M.: Bloat control and generalization pressure using the minimum description length principle for a pittsburgh approach learning classifier system. In: Proceedings of the 6th International Workshop on Learning Classifier Systems. LNCS (LNAI). Springer, Heidelberg (in press, 2003)
8. Bacardit, J., Goldberg, D.E., Butz, M.V.: Improving the performance of a pittsburgh learning classifier system using a default rule. In: Kovacs, T., Llorà, X., Takadama, K., Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 2003. LNCS (LNAI), vol. 4399, pp. 291–307. Springer, Heidelberg (2007)
9. Bacardit, J., Krasnogor, N.: Empirical evaluation of ensemble techniques for a pittsburgh learning classifier system. In: Ninth International Workshop on Learning Classifier Systems (IWLCS 2006). LNCS (LNAI). Springer, Heidelberg (to appear, 2006)
10. Bacardit, J., Krasnogor, N.: Smart crossover operator with multiple parents for a pittsburgh learning classifier system. In: GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation, pp. 1441–1448. ACM Press, New York (2006)
11. Bacardit, J., Stout, M., Krasnogor, N., Hirst, J.D., Blazewicz, J.: Coordination number prediction using learning classifier systems: performance and interpretability. In: GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation, pp. 247–254. ACM Press, New York (2006)
12. Basu, M., Ho, T.K.E.: Data Complexity in Pattern Recognition. Springer, Heidelberg (2006)
13. Bernadó-Mansilla, E., Llorà, X., Traus, I.: Multiobjective Learning Classifier Systems. In: Multi-Objective Machine Learning. Studies in Computational Intelligence, vol. 16, pp. 261–288. Springer, Heidelberg (2006)
14. Bernadó-Mansilla, E., Garrell-Guiu, J.M.: Accuracy-based learning classifier systems: Models, analysis and applications to classification tasks. Evolutionary Computation 11, 209–238 (2003)
15. Bernadó-Mansilla, E., Ho, T.K.: Domain of Competence of XCS Classifier System in Complexity Measurement Space. IEEE Transactions on Evolutionary Computation 9, 82–104 (2005)
16. Bernadó-Mansilla, E., Kam Ho, T.: On Classifier Domains of Competence. In: Proceedings of the 17th International Conference on Pattern Recognition, vol. 1, pp. 136–139 (2004)
17. Bernadó-Mansilla, E., Llorà, X., Garrell, J.M.: XCS and GALE: a comparative study of two learning classifier systems with six other learning algorithms on classification tasks. In: Fourth International Workshop on Learning Classifier Systems - IWLCS-2001, pp. 337–341 (2001)

18. Bonarini, A.: Evolutionary Learning of Fuzzy rules: competition and cooperation. In: Fuzzy Modelling: Paradigms and Practice, pp. 265–284. Kluwer Academic Press, Norwell (1996)
19. Brown, G., Kovacs, T., Marshall, J.A.R.: Ucspv: principled voting in ucs rule populations. In: GECCO 2007: Proceedings of the 9th annual conference on Genetic and evolutionary computation, pp. 1774–1781. ACM Press, New York (2007)
20. Browne, W.: The development of an industrial learning classifier system for data-mining in a steel hot strip mill. In: Bull, L. (ed.) Applications of Learning Classifier Systems, pp. 223–259. Springer, Heidelberg (2004)
21. Browne, W.N., Ioannides, C.: Investigating scaling of an abstracted lcs utilising ternary and s-expression alphabets. In: GECCO 2007: Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation, pp. 2759–2764. ACM Press, New York (2007)
22. Bull, L., Hurst, J., Tomlison, A.: Self-adaptive mutation in classifier system controllers. In: Meyer, J.A., Berthoz, A., Floreano, D., Roitblatt, H., Wilson, S. (eds.) From Animals to Animats 6 - The Sixth International Conference on the Simulation of Adaptive Behaviour. MIT Press, Cambridge (2000)
23. Bull, L., Studley, M., Bagnall, A., Whittley, I.: Learning classifier system ensembles with rule-sharing. IEEE Transactions on Evolutionary Computation 11, 496–502 (2007)
24. Bull, L. (ed.): Applications of Learning Classifier Systems. Springer, Heidelberg (2004)
25. Bull, L.: On lookahead and latent learning in simple LCS. In: GECCO 2007: Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation, pp. 2633–2636. ACM, New York (2007)
26. Butz, M.V., Goldberg, D.E., Lanzi, P.L.: Computational complexity of the XCS classifier system. In: Bull, L., Kovacs, T. (eds.) Foundations of Learning Classifier Systems. Studies in Fuzziness and Soft Computing, pp. 91–126. Springer, Heidelberg (2005)
27. Butz, M.V.: Anticipatory learning classifier systems. Kluwer Academic Publishers, Boston (2002)
28. Butz, M.V.: Kernel-based, ellipsoidal conditions in the real-valued XCS classifier system. In: GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation, pp. 1835–1842. ACM Press, New York (2005)
29. Butz, M.V.: Rule-Based Evolutionary Online Learning Systems: A Principled Approach to LCS Analysis and Design. Springer, Heidelberg (2006)
30. Butz, M.V., Goldberg, D.E.: Bounding the population size in XCS to ensure reproductive opportunities. In: Proceedings of the Fifth Genetic and Evolutionary Computation Conference (GECCO 2003), pp. 1844–1856 (2003)
31. Butz, M.V., Goldberg, D.E., Lanzi, P.L.: Gradient descent methods in learning classifier systems: Improving XCS performance in multistep problems. IEEE Transactions on Evolutionary Computation 9, 452–473 (2005)
32. Butz, M.V., Goldberg, D.E., Lanzi, P.L., Sastry, K.: Problem solution sustenance in XCS: Markov chain analysis of niche support distributions and the impact on computational complexity. Genetic Programming and Evolvable Machines 8, 5–37 (2007)
33. Butz, M.V., Hoffmann, J.: Anticipations control behavior: Animal behavior in an anticipatory learning classifier system. Adaptive Behavior 10, 75–96 (2002)
34. Butz, M.V., Kovacs, T., Lanzi, P.L., Wilson, S.W.: Toward a theory of generalization and learning in XCS. IEEE Transactions on Evolutionary Computation 8, 28–46 (2004)

35. Butz, M.V., Lanzi, P.L., Wilson, S.W.: Hyper-ellipsoidal conditions in xcs: rotation, linear approximation, and solution structure. In: GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation, pp. 1457–1464. ACM Press, New York (2006)
36. Butz, M.V., Pelikan, M., Llorà, X., Goldberg, D.E.: Automated global structure extraction for effective local building block processing in XCS. Evol. Comput. 14, 345–380 (2006)
37. Butz, M.V., Sastry, K., Goldberg, D.E.: Strong, stable, and reliable fitness pressure in XCS due to tournament selection. Genetic Programming and Evolvable Machines 6, 53–77 (2005)
38. Casillas, J., Carse, B., Bull, L.: Fuzzy-xcs: A michigan genetic fuzzy system. IEEE Transactions on Fuzzy Systems 15, 536–550 (2007)
39. Cordón, O., Herrera, F., Hoffmann, F., Magdalena, L.: Genetic Fuzzy Systems. Evolutionary tuning and learning of fuzzy knowledge bases. World Scientific, Singapore (2001)
40. De Jong, K.: Learning with genetic algorithms: An overview. Mach. Learn. 3, 121–138 (1988)
41. DeJong, K.A., Spears, W.M.: Learning concept classification rules using genetic algorithms. In: Proceedings of the International Joint Conference on Artificial Intelligence, pp. 651–656. Morgan Kaufmann, San Francisco (1991)
42. Dixon, P.W., Corne, D.W., Oates, M.J.: A Preliminary Investigation of Modified XCS as a Generic Data Mining Tool. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 2001. LNCS (LNAI), vol. 2321, pp. 133–150. Springer, Heidelberg (2002)
43. Drugowitsch, J., Barry, A.: A formal framework and extensions for function approximation in learning classifier systems. Machine Learning 70, 45–88 (2008)
44. Freitas, A.A.: Data Mining and Knowledge Discovery with Evolutionary Algorithms. Springer, Heidelberg (2002)
45. Fu, C., David, L.: A Modified Classifier System Compaction Algorithm. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 920–925. Morgan Kaufmann Publishers Inc., San Francisco (2002)
46. Gérard, P., Meyer, J.A., Sigaud, O.: Combining latent learning and dynamic programming in MACS. European Journal of Operational Research 160, 614–637 (2005)
47. Gérard, P., Sigaud, O.: Adding a generalization mechanism to YACS. In: Proceedings of the Third Genetic and Evolutionary Computation Conference (GECCO 2001), pp. 951–957 (2001)
48. Ghahramani, Z., Wolpert, D.M.: Modular decomposition in visuomotor learning. Nature, 392–395 (1997)
49. Ghosh, A., Nath, B.: Multi-objective rule mining using genetic algorithms. Information Sciences 163, 123–133 (2004)
50. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Publishing Company, Inc., Reading (1989)
51. Goldberg, D.E.: The Design of Innovation: Lessons from and for Competent Genetic Algorithms. Kluwer Academic Publishers, Dordrecht (2002)
52. Grush, R.: The emulation theory of representation: Motor control, imagery, and perception. Behavioral and Brain Sciences 27, 377–396 (2004)
53. Haruno, M., Wolpert, D.M., Kawato, M.: Hierarchical mosaic for movement generation. In: Ono, T., Matsumoto, G., Llinas, R., Berthoz, A., Norgren, R., Nishijo, H., Tamura, R. (eds.) Excepta Medica International Coungress Series, vol. 1250, pp. 575–590. Elsevier, Amsterdam (2003)

54. Dam, H.H., Lokan, C., Abbas, H.A.: Evolutionary online data mining: An investigation in a dynamic environment. In: Evolutionary Computation in Dynamic and Uncertain Environments, pp. 153–178. Springer, Heidelberg (2007)
55. Ho, T.K., Basu, M.: Measuring the complexity of classification problems. In: 15th International Conference on Pattern Recognition, pp. 43–47 (2000)
56. Holland, J.H.: A cognitive system with powers of generalization and adaptation (Unpublished manuscript) (1977)
57. Holland, J.H., Reitman, J.S.: Cognitive systems based on adaptive algorithms. In: Hayes-Roth, D., Waterman, F. (eds.) Pattern-directed Inference Systems, pp. 313–329. Academic Press, New York (1978)
58. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press (1975)
59. Holland, J.H.: Adaptation. In: Rosen, R., Snell, F. (eds.) Progress in theoretical biology, vol. 4, pp. 263–293. Academic Press, New York (1976)
60. Holland, J.H.: Properties of the bucket brigade algorithm. In: Proceedings of an International Conference on Genetic Algorithms and their Applications, pp. 1–7 (1985)
61. Holmes, J.H., Durbin, D.R., Winston, F.K.: The learning classifier system: an evolutionary computation approach to knowledge discovery in epidemiologic surveillance. Artificial Intelligence In Medicine 19, 53–74 (2000)
62. Hurst, J., Bull, L.: A neural learning classifier system with self-adaptive constructivism for mobile robot learning. Artificial Life 12, 1–28 (2006)
63. Ishibuchi, H., Nakashima, T., Murata, T.: Three-objective genetics-based machine learning for linguistic rule extraction. Information Sciences 136, 109–133 (2001)
64. Ishibuchi, H., Nojima, Y.: Analysis of interpretability-accuracy tradeoff of fuzzy systems by multiobjective fuzzy genetics-based machine learning. International Journal of Approximate Reasoning 44, 4–31 (2007)
65. Jin, Y.: A comprehensive survey of fitness approximation in evolutionary computation. Soft Comput. 9, 3–12 (2005)
66. Kharbat, F., Bull, L., Odeh, M.: Revisiting genetic selection in the xcs learning classifier system. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2005), pp. 2061–2068 (2005)
67. Kovacs, T.: XCS Classifier System Reliably Evolves Accurate, Complete and Minimal Representations for Boolean Functions. In: Roy, R., Chawdhry, P., Pant, R. (eds.) Soft Computing in Engineering Design and Manufacturing, pp. 59–68. Springer, Heidelberg (1997)
68. Krasnogor, N., Smith, J.: A tutorial for competent memetic algorithms: model, taxonomy and design issues. IEEE Transactions on Evolutionary Computation 9, 474–488 (2005)
69. Bull, L., Studley, M., Bagnall, A.J., Whittley, I.: On the use of rule sharing in learning classifier system ensembles. In: Proceedings of the 2005 Congress on Evolutionary Computation (2005)
70. Landau, S., Picault, S., Sigaud, O., Gérard, P.: Further comparison between AT-NoSFERES and XCSM. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 2002. LNCS (LNAI), vol. 2661, pp. 99–117. Springer, Heidelberg (2003)
71. Lanzi, P.L.: An analysis of generalization in the XCS classifier system. Evolutionary Computation 7, 125–149 (1999)
72. Lanzi, P.L.: Adaptive agents with reinforcement learning and internal memory. In: From Animals to Animats 6: Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior, pp. 333–342 (2000)

73. Lanzi, P.L.: Learning classifier systems: then and now. Evolutionary Intelligence 1, 63–82 (2008)
74. Lanzi, P.L., Loiacono, D.: Classifier systems that compute action mappings. In: GECCO 2007: Proceedings of the 9th annual conference on Genetic and evolutionary computation, London, England, pp. 1822–1829. ACM Press, New York (2007)
75. Lanzi, P.L., Loiacono, D., Wilson, S.W., Goldberg, D.E.: Extending xcsf beyond linear approximation. In: GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation, pp. 1827–1834. ACM, New York (2005)
76. Lanzi, P.L., Loiacono, D., Wilson, S.W., Goldberg, D.E.: Classifier prediction based on tile coding. In: GECCO 2006: Genetic and Evolutionary Computation Conference, pp. 1497–1504 (2006)
77. Lanzi, P.L., Loiacono, D., Wilson, S.W., Goldberg, D.E.: Prediction update algorithms for XCSF: RLS, kalman filter, and gain adaptation. In: GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation, pp. 1505–1512. ACM Press, New York (2006)
78. Lanzi, P.L., Perrucci, A.: Extending the representation of classifier conditions part II: From messy coding to s-expressions. In: Banzhaf, W., Daida, J., Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M., Smith, R.E. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, Orlando, Florida, USA, vol. 1, pp. 345–352. Morgan Kaufmann, San Francisco (1999)
79. Lanzi, P.L., Rocca, S., Solari, S.: An approach to analyze the evolution of symbolic conditions in learning classifier systems. In: GECCO 2007: Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation, pp. 2795–2800. ACM Press, New York (2007)
80. Larranaga, P., Lozano, J. (eds.): Estimation of Distribution Algorithms, A New Tool for Evolutionnary Computation. Genetic Algorithms and Evolutionnary Computation. Kluwer Academic Publishers, Dordrecht (2002)
81. Llorà, X., Priya, A., Bhargava, R.: Observer-invariant histopathology using genetics-based machine learning. Natural Computing, Special issue on Learning Classifier Systems (in press, 2008)
82. Llorà, X., Garrell, J.M.: Knowledge-independent data mining with fine-grained parallel evolutionary algorithms. In: Proceedings of the Third Genetic and Evolutionary Computation Conference, pp. 461–468. Morgan Kaufmann, San Francisco (2001)
83. Llorà, X., Reddy, R., Matesic, B., Bhargava, R.: Towards better than human capability in diagnosing prostate cancer using infrared spectroscopic imaging. In: GECCO 2007: Proceedings of the 9th annual conference on Genetic and evolutionary computation, pp. 2098–2105. ACM Press, New York (2007)
84. Llorà, X., Sastry, K.: Fast rule matching for learning classifier systems via vector instructions. In: GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation, pp. 1513–1520. ACM Press, New York (2006)
85. Llorà, X., Sastry, K., Goldberg, D.E., delaOssa, L.: The x-ary extended compact classifier system: Linkage learning in pittsburgh LCS. In: Proceedings of the 9th International Workshop on Learning Classifier Systems - IWLCS 2006. LNCS (LNAI). Springer, Heidelberg (in press, 2006)
86. Llorà, X., Sastry, K., Yu, T.L., Goldberg, D.E.: Do not match, inherit: fitness surrogates for genetics-based machine learning techniques. In: GECCO 2007: Proceedings of the 9th annual conference on Genetic and evolutionary computation, pp. 1798–1805. ACM, New York (2007)
87. Loiacono, D., Marelli, A., Lanzi, P.L.: Support vector regression for classifier prediction. In: GECCO 2007: Proceedings of the 9th annual conference on Genetic and evolutionary computation, pp. 1806–1813. ACM Press, New York (2007)

88. Luca Lanzi, P., Loiacono, D.: XCSF with neural prediction. Evolutionary Computation, CEC 2006. IEEE Congress on (0-0 0) 2270–2276 (2006)
89. Marshall, J.A.R., Brown, G., Kovacs, T.: Bayesian estimation of rule accuracy in ucs. In: GECCO 2007: Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation, pp. 2831–2834. ACM Press, New York (2007)
90. O'Regan, J.K., Noë, A.: A sensorimotor account of vision and visual consciousness. Behavioral and Brain Sciences 24, 939–1031 (2001)
91. Orriols-Puig, A., Bernadó-Mansilla, E.: The Class Imbalance Problem in UCS Classifier System: Fitness Adaptation. In: Proceedings of the 2005 Congress on Evolutionary Computation, vol. 1, pp. 604–611. IEEE Computer Society Press, Los Alamitos (2005)
92. Orriols-Puig, A., Bernadó-Mansilla, E.: Bounding XCS's Parameters for Unbalanced Datasets. In: Proceedings of the 2006 Genetic and Evolutionary Computation Conference, vol. 2, pp. 1561–1568. ACM Press, New York (2006)
93. Orriols-Puig, A., Bernadó-Mansilla, E.: The Class Imbalance Problem in Learning Classifier Systems: A Preliminary Study. In: Kovacs, T., Llorà, X., Takadama, K., Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 2003. LNCS (LNAI), vol. 4399, pp. 164–183. Springer, Heidelberg (2007)
94. Orriols-Puig, A., Casillas, J., Bernadó-Mansilla, E.: Fuzzy-UCS: A Michigan-style Learning Fuzzy-Classifier System for Supervised Learning. IEEE Transactions on Evolutionary Computation (in press, 2008)
95. Orriols-Puig, A., Goldberg, D., Sastry, K., Bernadó-Mansilla, E.: Modeling XCS in Class Imbalances: Population Size and Parameter Settings. In: Proceedings of the 2007 Genetic and Evolutionary Computation Conference, vol. 2, pp. 1838–1845. ACM Press, New York (2007)
96. Orriols-Puig, A., Sastry, K., Lanzi, P., Goldberg, D., Bernadó-Mansilla, E.: Modeling selection pressure in XCS for proportionate and tournament selection. In: Proceedings of the 2007 Genetic and Evolutionary Computation Conference, vol. 2, pp. 1846–1853. ACM Press, New York (2007)
97. Orriols Puig, A., Bernadó-Mansilla, E.: Analysis of Reduction Algorithms in XCS Classifier System. In: Recent Advances in Artificial Intelligence Research and Development. Frontiers in Artificial Intelligence and Applications, vol. 113, pp. 383–390. IOS Press, Amsterdam (2004)
98. Orriols-Puig, A., Bernadó-Mansilla, E., Sastry, K., Goldberg, D.E.: Substructrual surrogates for learning decomposable classification problems: implementation and first results. In: GECCO 2007: Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation, pp. 2875–2882. ACM, New York (2007)
99. Orriols-Puig, A., Sastry, K., Goldberg, D.E., Bernadó-Mansilla, E.: Substructural surrogates for learning decomposable classification problems. In: Bacardit, J., et al. (eds.) IWLCS 2006/2007. LNCS (LNAI), vol. 4998. Springer, Heidelberg (2008)
100. Parodi, A., Bonelli, P.: A new approach to fuzzy classifier systems. In: 5th International Conference on Genetic Algorithms, pp. 223–230. Morgan Kaufmann, San Francisco (1993)
101. Rivest, R.L.: Learning decision lists. Machine Learning 2, 229–246 (1987)
102. Smith, R.E., El-Fallah, A., Ravichandran, B., Mehra, R., Dike, B.A.: The fighter aircraft LCS: A real-world, machine innovation application. In: Bull, L. (ed.) Applications of Learning Classifier Systems, pp. 113–142. Springer, Heidelberg (2004)
103. Smith, R.E., Jiang, M.K.: A learning classifier system with mutual-information-based fitness. Evolutionary Computation, 2007. CEC 2007. IEEE Congress on (25-28 Sept. 2007) (2173)–2180

104. Smith, S.F.: A Learning System Based on Genetic Algorithms. PhD thesis, University of Pittsburgh (1980)
105. Smith, S.F.: Flexible learning of problem solving heuristics through adaptive search. In: Proceedings of the Eighth International Joint Conference on Artificial Intelligence, Los Altos, CA, pp. 421–425. Morgan Kaufmann, San Francisco (1983)
106. Stolzmann, W.: Anticipatory classifier systems. In: Genetic Programming 1998: Proceedings of the Third Annual Conference, pp. 658–664 (1998)
107. Stone, C., Bull, L.: For real! XCS with continuous-valued inputs. Evolutionary Computation Journal 11, 298–336 (2003)
108. Stout, M., Bacardit, J., Hirst, J.D., Krasnogor, N.: Prediction of recursive convex hull class assignments for protein residues. Bioinformatics (in press, 2008)
109. Suzuki, T., Kodama, T., Furuhashi, T., Tsut, H.: Fuzzy modeling using genetic algorithms with fuzzy entropy as conciseness measure. Information Sciences 136, 53–67 (2001)
110. Tamee, K., Bull, L., Pinngern, O.: Towards clustering with XCS. In: GECCO 2007: Proceedings of the 9th annual conference on Genetic and evolutionary computation, pp. 1854–1860. ACM Press, New York (2007)
111. Valenzuela-Rendón, M.: The fuzzy classifier system: A classifier system for continuously varying variables. In: Fourth International Conference on Genetic Algorithms (ICGA), pp. 346–353. Morgan Kaufmann, San Francisco (1991)
112. Wilson, S.W.: Classifier fitness based on accuracy. Evolutionary Computation 3, 149–175 (1995)
113. Wilson, S.W.: Knowledge growth in an artificial animal. In: Proceedings of an International Conference on Genetic Algorithms and Their Applications, pp. 16–23 (1985)
114. Wilson, S.W.: Classifier systems and the animat problem. Machine Learning 2, 199–228 (1987)
115. Wilson, S.W.: ZCS: A zeroth level classifier system. Evolutionary Computation 2, 1–18 (1994)
116. Wilson, S.W.: Get real! XCS with continuous-valued inputs. In: Booker, L., Forrest, S., Mitchell, M., Riolo, R.L. (eds.) Festschrift in Honor of John H. Holland, Center for the Study of Complex Systems, pp. 111–121 (1999)
117. Wilson, S.W.: Classifiers that approximate functions. Natural Computing: an international journal 1, 211–234 (2002)
118. Wilson, S.W.: Compact Rulesets from XCSI. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 2001. LNCS (LNAI), vol. 2321. Springer, Heidelberg (2002)
119. Wolpert, D.M., Kawato, M.: Multiple paired forward and inverse models for motor control. Neural Networks 11, 1317–1329 (1998)
120. Wyatt, D., Bull, L.: A memetic learning classifier system for describing continuous-valued problem spaces. In: Hart, W., Krasnogor, N., Smith, J. (eds.) Recent Advances in Memetic Algorithms, pp. 355–396. Springer, Heidelberg (2004)
121. Wyatt, D., Bull, L., Parmee, I.: Building Compact Rulesets for Describing Continuous-Valued Problem Spaces Using a Learning Classifier System. In: Parmee, I. (ed.) Adaptive Computing in Design and Manufacture, vol. VI, pp. 235–248. Springer, Heidelberg (2004)
122. Zatuchna, Z.V.: AgentP: A Learning Classifier System with Associative Perception in Maze Environments. PhD thesis, School of Computing Sciences, UEA (2005)

Analysis of Population Evolution in Classifier Systems Using Symbolic Representations

Pier Luca Lanzi[1,2], Stefano Rocca[1], Kumara Sastry[2], and Stefania Solari[1]

[1] Artificial Intelligence and Robotics Laboratory (AIRLab)
Politecnico di Milano. I-20133, Milano, Italy
pierluca.lanzi@polimi.it,
rocca.ste@gmail.com, solari.stefania@gmail.com
[2] Illinois Genetic Algorithm Laboratory
University of Illinois at Urbana-Champaign, Urbana, Illinois, USA
lanzi@illigal.ge.uiuc.edu, kumara@kumarasastry.com

Abstract. This paper presents an approach to analyze population evolution in classifier systems using a symbolic representation. Given a sequence of populations, representing the evolution of a solution, the method simplifies the classifiers in the populations by reducing them to their "canonical form". Then, it extracts all the subexpressions that appear in all the classifier conditions and, for each subexpression, it computes the number of occurrences in each population. Finally, it computes the trend of all the subexpressions considered. The expressions which show an increasing trend through the course of evolution are viewed as building blocks that the system has used to construct the solution.

1 Introduction

Generalization is one of the most important features of learning classifier systems which heavily relies on the representation of classifier conditions. Among the several condition representations introduced in the literature [1,4,6,12,13,14,15,16], symbolic (GP-like) representation is probably the most general one but also the most computationally demanding one. Symbolic conditions are also daunting to study. The bloat that affects variable size and GP-like representations [2,6,7,8] and the possibly high number of classifiers involved makes the analysis of this type of systems more difficult [10].

In this paper, we present a method to analyze the evolution of symbolic conditions in classifier systems and apply it to analyze how XCS with GP-like conditions solves two simple problems. The results we present demonstrate that our method can trace how the genetic pressure in XCS favors the evolution of subexpressions that are useful to construct an optimal solution to the target problem. The method takes as input a sequence of evolved populations, it extracts a set of frequent patterns which identify useful pieces of the problem solution, so to track the emergence of the optimal solution. When applied to experiments that reached optimality, the method shows what interesting pieces of knowledge XCS recombined to come up with the solution. When applied to experiments that did

J. Bacardit et al. (Eds.): IWLCS 2006/2007, LNAI 4998, pp. 22–45, 2008.

not reach optimality, the method shows that interesting patterns are still present in the populations but a higher number of uninteresting (overly general or inaccurate) patterns dominates the populations. The method works as follows. Given a sequence of populations, representing the evolution of a problem solution, the method (i) reduces the classifiers in the populations to their "canonical form" (Section 2.1) and (ii) it simplifies the populations (Section 2.2); then, (iii) it extracts all the subexpressions that appear in all the classifier conditions (Section 2.3) and (iv) for each subexpression it computes the number of occurrences in each population (Section 3); at last, (v) for each subexpression considered, it computes the trend of the subexpressions extracted in the populations (Section 4). In this context, the expressions with an increasing trend through the course of evolution are viewed as the building blocks that the system has used to construct the solution.

2 Subexpressions Extraction

Given a sequence of populations representing the evolution of a solution, the proposed method transforms each classifier in the populations into a "canonical form" so that two syntactically different but structurally equivalent classifiers are represented in the same way, i.e., by the same "canonical form". The simplified classifiers are then analyzed to extract the recurrent subexpressions which might be viewed as the elementary blocks that the system uses to construct the final solution.

2.1 The Canonical Form

A classifiers is transformed into its "canonical form" through four steps: (i) simplification of the classifier condition, (ii) conversion of the condition into disjunctive normal form, (iii) normalization and (iv) sorting of the expressions that compose the conjuncts.

Simplification. To reduce the amount of useless and redundant code, we initially simplify classifier conditions before extracting the subexpressions. For this purpose, we apply a set of simplification rules and specialized operators [3] which in this work are provided by Mathematica® [9]. Details about Mathematica® and the simplification functions we developed can be found in [10]. This first step brings classifier conditions into a more compact form. For example, it maps the condition "$X0 + X1 + (1 - 1)X0 - 10 > -5\ AND\ 5(X0 - X0) = 0$" into the more compact "$X0 + X1 - 5 > 0$".

Conversion into disjunctive normal form. If Boolean operators are included, individuals are then expanded into their the disjunctive normal form. The disjunctive normal form of an expression is a disjunction (sequence of ORs) consisting of one or more disjuncts, each of which is a conjunction (AND) of one or more Boolean expressions. This step splits classifier conditions into a set of subexpressions grouped by a set of "OR" clauses. For example, it transforms the

condition "$X0 = 4\ AND\ (X1 = 1\ OR\ X1 = 2)$" into "$X0 = 4\ AND\ X1 = 1\ OR\ X0 = 4\ AND\ X1 = 2$" in which three subexpressions ("$X0 = 4\ AND\ X1 = 1$", "$X0 = 4$", and "$AND\ X1 = 2$") are grouped together by two OR clauses. This step is performed by Mathematica® (see [10] for details).

Normalization. Although all the expressions have been simplified in a disjunctive normal form, two equivalent subexpressions may still be syntactically different. For example, "$2(X0 + X1) < 9$" and "$X1 + X0 < 9/2$" are equivalent, they cannot be further simplified, but they are syntactically different. To tackle this issue, we have developed a routine in Mathematica® that rearranges an elementary expression into a normalized form such that two equivalent expressions become syntactically the same. This routine subtracts the second term of the equality or inequality to both sides of the equality/inequality forming a second term that is 0. Then it expands out products and positive integer powers in the first term, and divides both sides by the first coefficient of the first term. This routine is described in details in [10]. By applying this routine to the two elementary expressions previously introduced we obtain the same elementary expression "$1 - \frac{2}{9}X0 - \frac{2}{9}X1 > 0$". Since sometimes this normalized form can be difficult to read, each elementary expression is replaced with a chosen elementary expression used to represent all the elementary expressions with the same normalized form. After this replacement, equivalent expressions differently represented are mapped into the same, syntactically equivalent, expression. Note that, if an individual contains only Boolean operators, this step is unnecessary because all the elementary expressions are literals.

Sort. Two expressions consisting in the same set of subexpressions conjuncted in a different order are syntactically different but actually equivalent. For example, "$X0 > 0\ AND\ X1 > 3$" and "$X1 > 3\ AND\ X0 > 0$" are syntactically different but equivalent in that they represent the same concept. To avoid this issue it is possible to split each expression around the "AND" clauses, alphabetically sort the obtained set of subexpressions, and then rebuild them using "AND" clauses. For instance, by applying this procedure to the previous expression we obtain the same expression "$X0 > 0\ AND\ X1 > 3$".

2.2 Population Simplification

All the symbolic conditions in the population are reduced to the previously described "canonical form". For this purpose we use a data structure M_s which maps each expression to its normalized form. This mapping assures that in each set of subexpressions, built using the same M_s, equivalent subexpressions have the same representation. In addition, if an expression has already been found it is not necessary to recompute its canonical form. Elements in M_s are pairs (e_i, s_i) in which each expression e_i is mapped into its simplified expression s_i that is also used to represent all the elementary expressions that are equivalent to e_i.

Algorithm 1. Simplification of the population P_t

```
1: Load M_s
2: Load P_t
3: for all i : i ∈ P_t do {simplifies the individual}
4:    e_s ←Simplify(i) {converts individual into a normal disjunctive form}
5:    e_dnf ←Expand(e_s)
6:    S_OR ←Split(e_dnf,"OR") {normalize the elementary expression and sorts them}
7:    for all s_e : s_e ∈ S_OR do
8:       i_s ←NormalizedForm(M_s,s_e)
9:    end for
10:   i ← ∅ {substitutes each individual with the "canonical form"}
11:   for all s_e : s_e ∈ S_OR do
12:      if i = ∅ then
13:         i ← i + s_e
14:      else
15:         i ← i+"OR"+s_e
16:      end if
17:   end for
18: end for
```

Algorithm 1 describes the method we developed to simplify classifier populations. At first, the population P_t and the structure M_s are loaded (steps 1 and 2, Algorithm 1). Then for each individual i in the population P_t (step 3, Algorithm 1): (i) the individual is simplified by applying the first step described in the previous section (step 4, Algorithm 1); (ii) the simplified individual e_s is transformed in its disjunctive normal form e_{dnf} by applying the second step described in the previous section, obtaining (step 5, Algorithm 1); (iii) the set of subexpressions S_{OR} is obtained by splitting e_{dnf} around the "*OR*" clauses (step 6, Algorithm 1); (iv) each subexpression s_e is then replaced with its representation in a normalized form, applying the steps three and four described in the previous section (steps 7 and 8, Algorithm 1); (v) an individual i is replaced with a new individual built joining the obtained subexpressions with "*OR*" (steps 11 to 16, Algorithm 1).

Algorithm 2 shows the routine **NormalizedForm** used by Algorithm 1 to compute the normalized form of an expression s_e by applying the steps three and four described in the previous section. Initially, the set S_{AND} is computed by splitting s_e around the *AND* clauses (step 1, Algorithm 2). For each expression e_e in S_{AND} the following steps are performed: (i) search in M_s for the elementary expression and save in t the corresponding elementary expression that has to be used (step 3, Algorithm 2); (ii) if the elementary expression e_e has been found replace it with the corresponding elementary expression previously found (step 4 and 5, Algorithm 2); (iii) compute the normalized form of the elementary expression using Mathematica® and save it in c (step 7, Algorithm 2); (iv) find the elementary expression in M_s corresponding to an elementary expression with the same normalized form of e_e and put it in e_c (step 8, Algorithm 2); (v) add the mapping (e_e, e_c) to M_s (step 9, Algorithm 2). After all the elementary

Algorithm 2. Routine **NormalizedForm**

NormalizedForm(M_s,s_e)
1: $S_{AND} \leftarrow$Split(s_e,"AND")
2: **for all** $e_e : e_e \in S_{AND}$ **do** {extracts the normalized expression of e_e}
3: $t : (e_e, t) \in M_s$
4: **if** $t \neq \emptyset$ **then**
5: $e_e \leftarrow t$
6: **else** {computes the normalized form of e_e}
7: $c \leftarrow canonicalForm(e_e)$ {extracts the elementary expression with the same normalized form from M_s}
8: $e_c : (e, e_c) \in M_s \wedge canonicalForm(e) = c$ {adds the pair to M_s}
9: $M_s \leftarrow M_s \cup (e_e, e_c)$
10: $e_e \leftarrow e_c$
11: **end if**
12: **end for**
13: $sort(S_{AND})$
14: $s_e \leftarrow \emptyset$ {builds the new subexpression}
15: **for all** $e_e : e_e \in S_{AND}$ **do**
16: **if** $s_e = \emptyset$ **then**
17: $s_e \leftarrow s_e + e_e$
18: **else**
19: $s_e \leftarrow s_e$+"AND"+e_e
20: **end if**
21: **end for**
22: return s_e

expressions in S_{AND} have been modified, S_{AND} is alphabetically sorted (step 13, Algorithm 2) and the subexpression s_e is replaced with an empty one (step 14, Algorithm 2). Finally, the expressions in S_{AND} are joined with "AND" in order to form the new subexpression (steps 15 to 20, Algorithm 2).

2.3 Extraction of Subexpressions

We extract the subexpressions appearing in the population by splitting the classifier conditions, expressed in the canonical form, around certain operators. The issue in this case is how to decide around which operators we should split the conditions. We need subexpressions that represent nuggets of interesting information about the problem solution. As an example, consider the problem of learning the inequality "$X0 + X1 > 4$" with $X0, X1 \in \mathbb{N}$. Suppose we have the individual "$3X0 > 10$". If we split the expression "$3X0 > 10$" around the "$>$" symbol we obtain two subexpressions that do not represent any interesting information about the solution. The same happens if we split a subexpression like "$X1 * (X1 + X2)$" around "$*$": in this case each of the two subexpressions defines a sort of "context" for the other one. Furthermore consider the expression "$X1 > 3\ AND\ NOT(X0 = 0)$". If we split this expression around the "AND" clause we obtain the two subexpressions "$X1 > 3$" and "$NOT(X0 = 0)$". In this

Algorithm 3. Algorithm to compute the frequencies of the subexpressions in population P_t

```
1: n ← 0
2: S_n ← ∅
3: for all i ∈ P_t do {extracts the subexpressions}
4:    S_OR ← Split(i, OR)
5:    for all s_e : s_e ∈ S_OR do {updates the number of occurrences of the subexpression
      s_e}
6:       UpdateOccurrences(S_n, id[i], s_e)
7:       n ← n + 1
8:    end for
9: end for
10: for all s_n : s_n ∈ S_n do {computes the frequencies}
11:    s_n.occ ← s_n.occ/n
12:    output s_n
13: end for
```

case the second subexpression does not represent interesting information about the problem solution. This happens because also in this case each subexpression defines a sort of "context" for the other. Finally consider another expression, that is "$X1 > 3\ AND\ NOT(X0 = 0)\ OR\ X0 + X1 > 4$". If we split this expression around "OR" we obtain the two subexpressions "$X1 > 3\ AND\ NOT(X0 = 0)$" and "$X0 + X1 > 4$", which both represent very interesting information about the problem solution. This is because two subexpressions joint by "OR" are independent, therefore we choose to extract subexpressions splitting individuals around "OR". Furthermore individuals are represented by a disjunctive normal form, so there is no problem in doing it.

3 Subexpressions Counting

After a population has been simplified and all the classifiers in it have been reduced to a canonical form, the classifiers conditions are analyzed and the subexpressions in the populations are counted. This phase takes as input a population P_t of individuals represented in their canonical form and produces a data set of triplets S_t each one consisting of (i) a normalized subexpression extracted from the conditions expressed in the "canonical form", (ii) the number of occurrences of the subexpression in the population P_t, and (iii) the set of the individuals in P_t which contains subexpressions that are equivalent to the first element of the triplet. The set S_t contains all the distinct subexpressions found in the population P_t and maps them to the conditions where such subexpressions were found. By applying this extraction procedure to a sequence of evolving populations, represented in canonical form, it is possible to track the evolution of subexpressions appearing in the population.

Subexpressions with an increasing trend may be viewed as the useful component that the system exploits to build up the final solution; subexpressions with a constant or decreasing trend may be considered useless or too specific.

Algorithm 4. Routine **UpdateOccurrences**

```
UpdateOccurrences(S_n,id[i],s_e)
 1: j ← −1 {searches for the subexpression s_e in S_n}
 2: j : S_n[j].subexpression = s_e
 3: if j > −1 then {the subexpression has been found}
 4:    S_n[j].occ ← S_n[j].occ + 1
 5:    S_n[j].individuals ← S_n[j].individuals ∪ id[i])
 6:    return TRUE
 7: else {the subexpressions has not been found, creates a new triplet}
 8:    S_n ← S_n ∪ (s_e, 1, id[i])
 9:    return TRUE
10: end if
11: return FALSE
```

The frequencies of the subexpressions in the population are computed using Algorithm 3. The algorithm is applied to a population P_t of classifiers that have been already transformed in their canonical form and outputs the set of the subexpressions found in the population with their frequencies. First the set S_n (step 1, Algorithm 3) and the total number of occurrences of the subexpressions n (step 2, Algorithm 3) are initialized. Then for each individual i in the population P_t (step 3, Algorithm 3): (i) the set of subexpressions S_{OR} is obtained by splitting the individual i around "OR" (step 4, Algorithm 3); (ii) for each subexpression s_e in S_{OR} the number of occurrences of s_e is updated and also the total number of occurrences is updated (steps 5, 6, and 7, Algorithm 3). After the set S_n has been built, the following cycle is performed. For each triplet s_n in S_n (step 10, Algorithm 3): (i) the number of occurrences $s_n.occ$ of the subexpression is replaced with the frequency of the subexpression (step 11, Algorithm 3); (ii) the triplet s_n is the output (step 12, Algorithm 3). The procedure **UpdateOccurrences**, which updates the number of occurrences of a subexpression s_e in S_n, is reported as Algorithm 4. The procedure works as follows: (i) the subexpression s_e is searched in S_n (step 2, Algorithm 4); (ii) if it has been found (step 3, Algorithm 4), the number of occurrences of the subexpression is incremented (step 4, Algorithm 4), and the identifier of the individual is added to the set of the individuals that contain the subexpression s_e (step 5, Algorithm 4); (iii) otherwise a new triplet with the subexpression s_e, the number of occurrences equal to one, and the identifier of the individual $id[i]$ is added to S_n (step 8, Algorithm 4). Given a population P_t, the algorithm outputs a set S_t of triplets each one consisting of (i) a subexpression, (ii) the frequency of the subexpression in the population P_t, and (iii) the set of the individuals in P_t that contain subexpressions that are semantically equivalent to the first element of the triplet. By applying this algorithm to the populations

we obtain a set of S_t. Two equivalent subexpressions which represent the same concept and belong to two different S_t (i.e., appear to two different populations) are also syntactically equivalent. Therefore, since it is possible to identify the same subexpression (the same concept) along the sequence of population, it is also possible to analyze the trend of a subexpression in all the populations P_t produced during an experiment.

4 Analysis of Evolving Populations

We can combine the extraction (Section 2) and the counting (Section 3) steps to build up a method to analyze the evolution of classifiers populations in XCS with symbolic (GP-like) conditions [6,5]. The method takes as input a sequence of evolving population $\{P_t\}_{t=0,1,\ldots}$, it applies the extraction and the counting procedures to each population, and then it computes the trend of the most frequent subexpressions in the evolving populations. The idea is that the analysis of such a trend can help in explaining how XCS constructs solutions of symbolic conditions from bits and pieces of existing conditions. Subexpressions that are useful in terms of problem solution should show an increasing trend; in contrast, subsexpressions that are useless in terms of problem solution (e.g., because overly general or overly specific) are hypothesized to have a non-increasing trend.

5 Experimental Validation

To validate the proposed approach, we applied it to analyze how XCS with GP-based conditions [6] solves two simple problems: (i) the learning of the inequality "$X0 + X1 > 4$" with $X0, X1 \in \mathbb{N}$ and $X0, X1 \in [0, 9]$, taken from [17]; and (ii) the learning of the inequality "$X0 * X1 > 7$" with $X0, X1 \in \mathbb{N}$ and $X0, X1 \in [0, 9]$. For this purpose, we applied the standard experimental design we used elsewhere for similar problems [6,5]; classifier conditions are defined over the set of functions $F = \{NOT,\ AND,\ OR,\ >,\ =,\ +,\ -,\ *\}$ and the set of terminals $T = \{0,\ \ldots,\ 9,\ X0,\ X1\}$. For each experiment, we performed 10 runs and saved all the intermediate populations. We partitioned each population in two subsets, one for each possible actions; each subset contains the classifiers in the population that advocate the same action. For each subset, the following steps are performed: (i) extraction, to simplify each classifier into its canonical form; (ii) counting, to extract the set of all subexpressions with their frequency; then, for each run, (iii) we analyze the trend of the frequencies of each subpopulations identified by each available action. The results we present confirm what we expected: the frequency of general subexpressions that represent useful information about the problem has a growing trend, whereas the frequency of less important subexpressions has a constant or decreasing trend. Note that, as a proof of principle, in this paper we analyze only two runs for each problem: one in which XCS evolved an optimal solution; one in which XCS did not evolve an optimal solution.

5.1 Experiment 1: Sum of Two Variables

At first, we considered the problem of learning the inequality "$X0 + X1 > 4$" with $X0, X1 \in \mathbb{N}$ and $X0, X1 \in [0, 9]$ using a population of 1000 classifiers. We performed 10 runs, each one consisting of 50000 learning problems. For each run, we saved the population every 1000 steps, so at the end of each run we had a sequence of 50 populations. Note that we only performed 50000 learning problems for each run, so to have at least one run that did not reach optimal performance. Here, we analyze two runs: one is an example of successful convergence, one is an example of a run that did not reach optimality.

The first run we consider reached optimality after about 44000 steps. We begin by analyzing the classifiers that advocate action 0 and represent the negation of the inequality "$X0 + X1 > 4$", i.e., "$X0 + X1 < 5$". Figure 1 shows the trend of the frequencies of recurrent subexpressions. We have plotted the subexpressions with the highest average frequency. From the plot reported in Figure 1, it is possible to note that subexpressions representing interesting information about the problem have an increasing trend; whereas less relevant subexpressions have a decreasing trend. To better analyze the trends of subexpressions in population we apply linear regression to the data collected and obtain the plot reported in Figure 2. From this plot we extract the following recurrent subexpressions with an increasing trend: (i) "$X0^2 * X1 = 3$", (ii) "$9 * (X0 + X0 * X1 + X1^2) < 27 + 26 * X1$", (iii) "$5 * X0 < 14$ *AND* $X1 < 3$", (iv) "$X0 * X1^2 = 3$ *AND* $X1 < 3$", (v) "$X0 * X1^2 = 3$", and (vi) "$5 > X0$ *AND* $X1 < 1$". The subexpression (iv) can be further simplified to "$X0 = 3$ *AND* $X1 = 1$" because $X0, X1 \in \mathbb{N}$, but Mathematica® [9] could not perform this simplification because we have

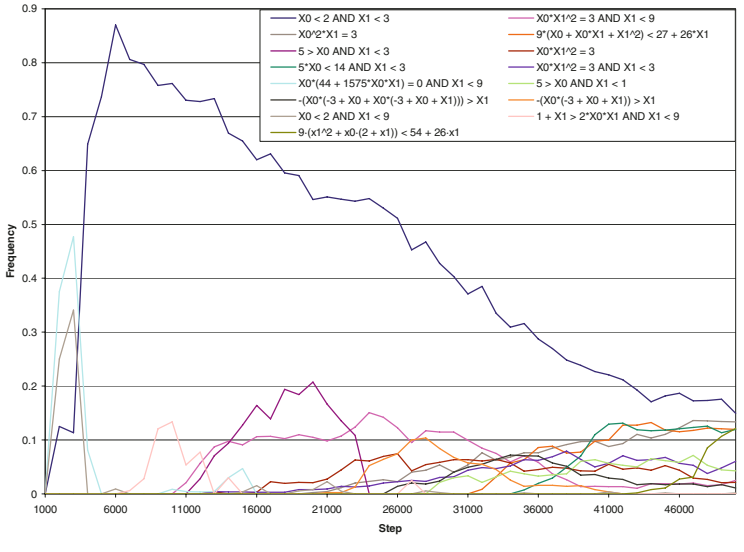


Fig. 1. Frequency trends of the subexpressions for the classifiers with action 0

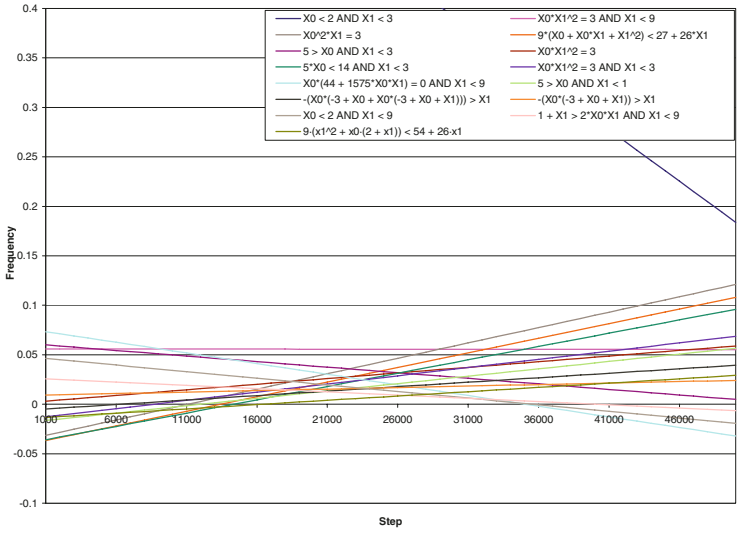


Fig. 2. Linear regression of the frequencies of recurrent subexpressions of Experiment 1, first run. The advocated action is 0.

not specified the variable domain (specifying the variable domain can lead to simplifications too difficult to be performed). All these subexpressions represent interesting information about the problem and are all subsets of "$X0 + X1 < 5$". Some of these subexpressions, like (i) and (v), are very specific but are not included in the other subexpressions. This is the cause of their growth in spite of their specificity. The subexpression "$X0 < 2\ AND\ X1 < 3$" instead has a decreasing trend because it is a subset of (iii), that has an increasing trend. The subexpression "$1+X1 > 2*X0*X1\ AND\ X1 < 9$ is not a subset of "$X0+X1 < 5$" so it does not represent a correct mapping for the action 0. In fact, the subsexpression has a decreasing trend. Note that the union of the subexpressions with an increasing trend listed before is almost a complete mapping between variable values and the action 0: only "$X0 = 0\ AND\ X1 = 4$" is not included in any of these subexpressions but is included in (vii) "$9*(X1^2 + X0*(2+X1)) < 54+26*X1$". This last subexpression has an increasing trend but a low frequency because it appears only after step 43000. Furthermore (vii) includes (ii), but (vii) has not replaced (ii) because (vii) has appeared too late in the evolution. Figure 3 shows the areas represented by the subexpressions (ii) and (vii).

We now analyze the classifiers that advocate action 1 that represent the inequality "$X0 + X1 > 4$". Figure 4 shows the trend of the frequencies of recurrent subexpressions. We have plotted the subexpressions with the highest average frequency. Also in this case we note that subexpressions which represent interesting information about the problem have an increasing trend, while useless subexpressions have a decreasing trend. The plot in Figure 5 reports the results of linear regression applied to the same data. From this plot we extract

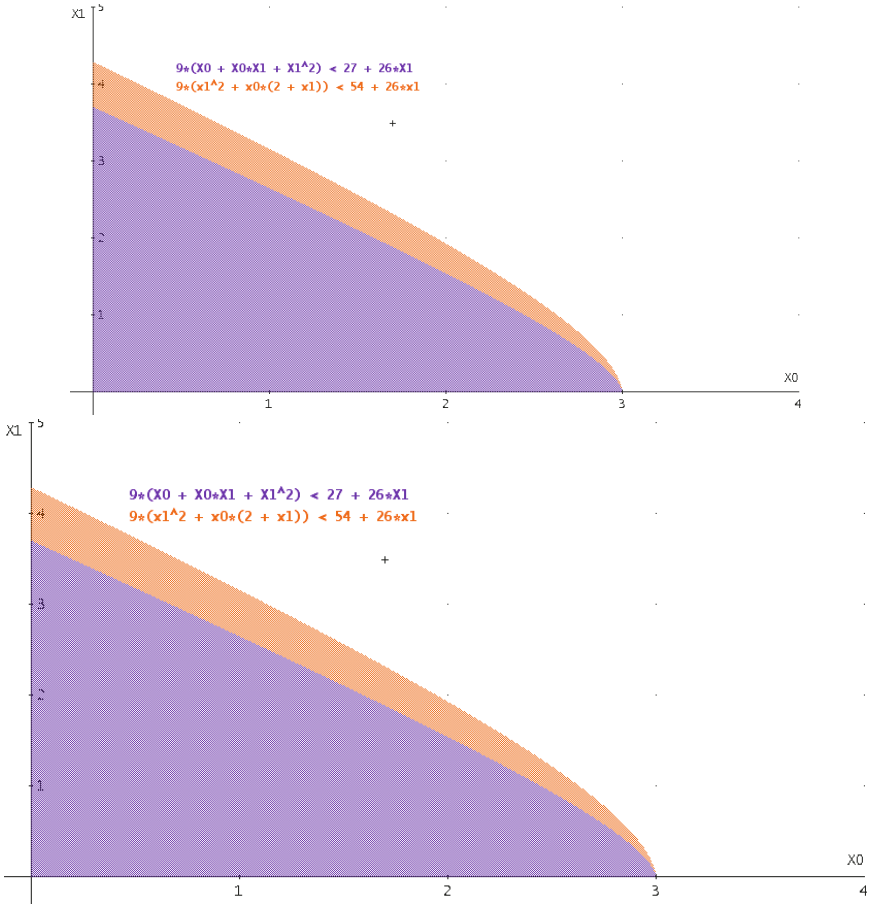


Fig. 3. Subexpressions "$9 * (X1^2 + X0 * (2 + X1)) < 54 + 26 * X1$" and "$9 * (X0 + X0 * X1 + X1^2) < 27 + 26 * X1$"

the following recurrent subexpressions with an increasing trend: (i) "$6 > X1$ AND $X0 > 4$", (ii) "$7 > X0$ AND $X1 * (-3 + X0 + X1) > X1$", and (iii) "$X1 * (-3 + X0 + X1) > X1$". Note that the subexpression (iii) is equivalent to "$NOT(X1 = 0)$ AND $X0 + X1 > 4$". Also in this case, all the recurrent subexpressions with an increasing trend represent interesting information about the problem, none of them is wrong. The subexpression (ii) is contained in the subexpression (iii) but it is not replaced by it. Although it is possible to note that the quick growth of the subexpression (ii) has stopped around step 15000, with the emergence of the subexpression (iii), then (ii) has decreased and after step 30000 has an almost constant trend. So the more general subexpression (iii) has grown hindering (ii). It is possible to note that the subexpression (iii) is contained in many other subexpressions like "$X0 > 0$ AND $X1 * (-3 + X0 + X1) > X1$ AND

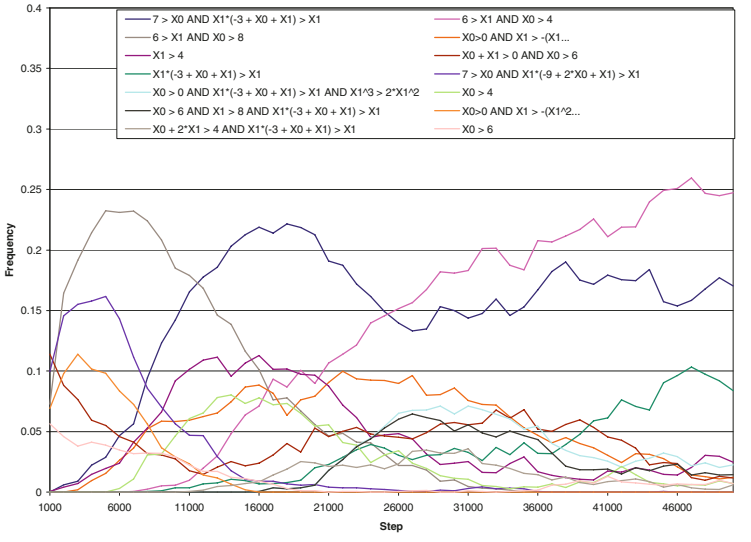


Fig. 4. Trend of the frequencies of recurrent subexpressions of Experiment 1, first run. The advocated action is 1.

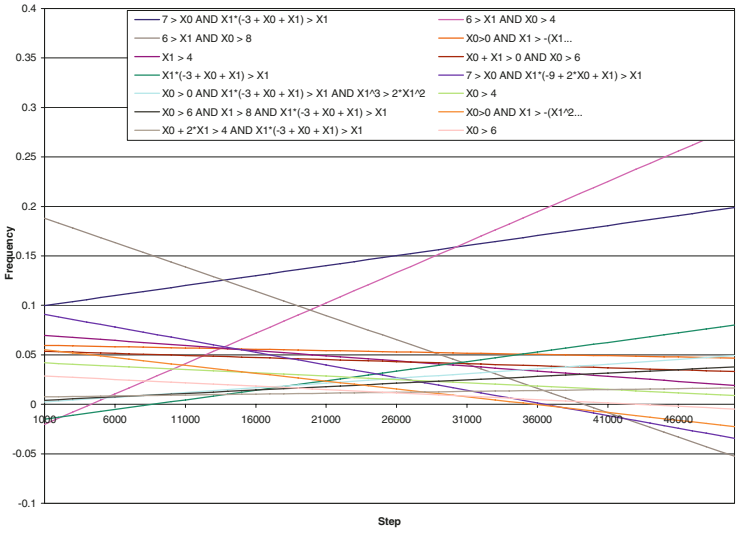


Fig. 5. Linear regression of the frequencies of recurrent subexpressions of Experiment 1, first run. The advocated action is 1.

$X1^3 > 2 * X1^2$" that has a slightly increasing trend. Subexpressions much less general, like "$6 > X1$ *AND* $X0 > 8$", have a decreasing trend. In this case, the union of the subexpressions with an increasing trend listed before is a complete mapping between variable values and the action 1.

The second run we consider did not reach optimal solution in 50000 steps, therefore all the populations contain some classifiers with a low fitness. We first analyze the classifiers that advocate action 0 which represent the inequality "$X0 + X1 < 5$". Figure 6 shows the trend of the frequencies of recurrent subexpressions. We have plotted the subexpressions with the highest average frequency. In this case none of the reported subexpressions represent interesting information about the problem. Figure 7 reports the linear regression from which it is possible to see that almost none of them has an increasing frequency trend. Some subexpressions, like "$X0 > 0$", disappear in some steps. That is because they disappear as a lone subexpression and they become a part of another subexpression, i.e. now they are ANDed with other subexpressions. Furthermore, most of these subexpressions are wrong because they are not a subset of "$X0 + X1 < 5$": one notable example is the only subexpression with an increasing trend, "$X0 * (44 + 1575 * X0 * X1) = 0$", that is equal to "$X0 = 0$". Furthermore this subexpression has an high oscillatory trend. To better understand the behavior exhibited by the subexpressions reported in these plots we analyzed the number of classifiers advocating each action. The average number of classifiers that advocate action 0 is about 5, while for action 1 is 991 and in the previous run was 247 for action 0 and 747 for action 1. It is easy to note that the number of classifiers that advocate action 0 in this run is too small to extract interesting information about subexpressions. Furthermore the system has not reached optimal solution so some classifiers have a high prediction error.

We then analyze the classifiers that advocate action 1, which. They represent the inequality "$X0 + X1 > 4$". Figure 8 shows the trend of the frequencies of recurrent subexpressions. Although the system has not reached optimal solution, it is possible to see that subexpressions that represent interesting information about the problem have an increasing trend, while useless subexpressions have a decreasing trend. This is because the average number of classifiers for each population that advocate action 1 is high, about 991. The plot in Figure 9 reports the linear regression. From this plot we extract the following recurrent subexpressions with an increasing trend: (i) "$X0 > 0$ *AND* $X1 > 7/2$", (ii) "$X0 > 4$", (iii) "$X1 > 6$", and (iv) "$X1 > 5$". Although the system has not reached optimal solution, all the recurrent subexpressions with an increasing trend listed before represent interesting information about the problem and none of them is wrong. None of these subexpressions is contained in the other. Wrong subexpressions or subexpressions that are subset of others have a decreasing trend. For example, consider the subexpression "$X1 > 7/2$": it has a decreasing trend and it is wrong because it is not a subset of "$X0 + X1 > 4$". Another example, consider the subexpression "$X0 > 7$": it has a decreasing trend and it is a subset of the subexpression (ii). It is possible to note that the quick growth of the subexpression "$X0 > 7$" has stopped around step 3000, with

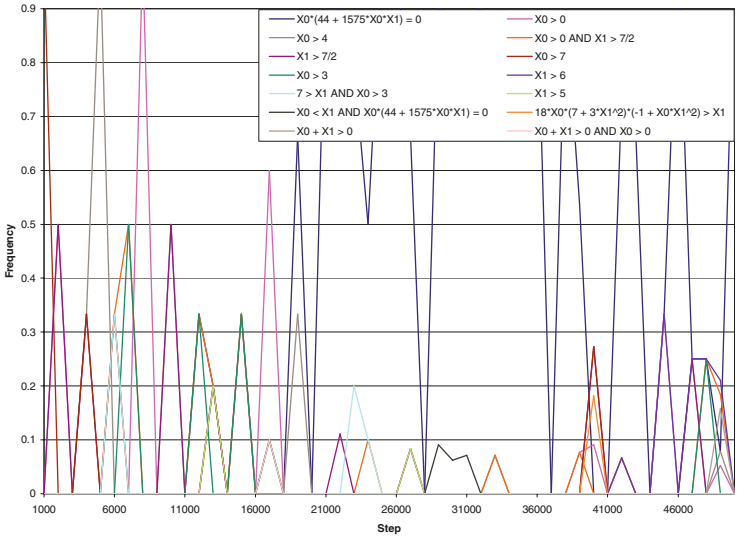


Fig. 6. Trend of the frequencies of recurrent subexpressions of Experiment 1, second run. The advocated action is 0.

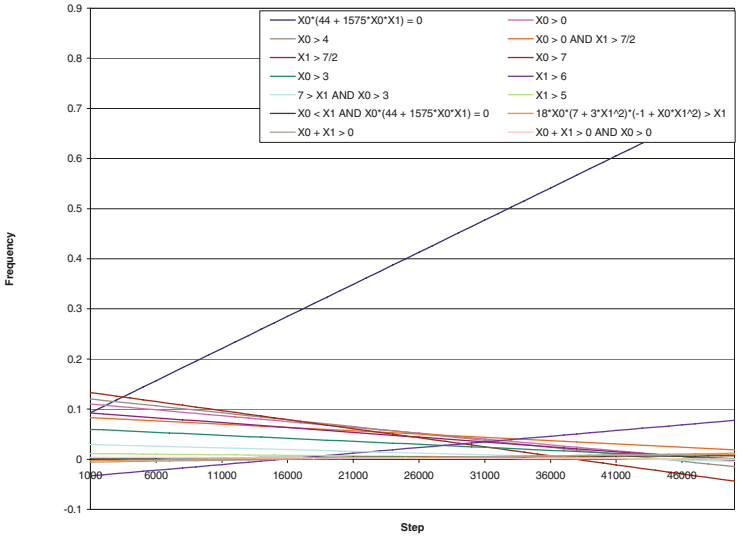


Fig. 7. Linear regression of the frequencies of recurrent subexpressions of Experiment 1, second run. The advocated action is 0.

the emergence of the subexpression (ii), then the frequency of "$X0 > 7$" has decreased. The union of the increasing subexpressions listed before represents an almost complete mapping between variable values and the action 1, only "$X0 = 0\ AND\ X1 = 5$" is not contained in any of them.

5.2 Experiment 2: Multiplication of Two Variables

In the second experiment, we considered the learning of "$X0 * X1 > 7$" with $X0, X1 \in \mathbb{N}$ and $X0, X1 \in [0, 9]$. We run XCS with symbolic conditions with a population of 3000 classifiers; 10 runs were performed, each one consisting of 200000 learning problems; in each run we saved the population every 10000 steps, so at the end of each run we have a sequence of 20 populations. Here, we analyze two runs: one is an example of successful convergence, one is an example of a run which did not reach optimality.

The first run reaches optimality after about 20000 steps. We first analyze the classifiers that advocate action 0 which represent the negation of the inequality "$X0 * X1 > 7$", that is "$X0 * X1 \leq 7$". Figure 10 shows the trend of the subexpressions with the higher average frequency. The plots in Figure 10 show that only one recurrent subexpression has an increasing trend. To better analyze this we apply linear regression and obtain the plots in Figure 11. This plot confirms what previously noted: only one subexpression "$X1 + 7 * X0 * (1 + 15 * X1) < 832$" has an increasing trend. Another subexpression has an increasing trend but its max frequency is 0.008 so it is not relevant. Figure 12 shows the area covered by "$X1 + 7 * X0 * (1 + 15 * X1) < 832$" and the curve "$X0 * X1 = 7$", i.e., the only increasing subexpression represents a complete solution to the problem

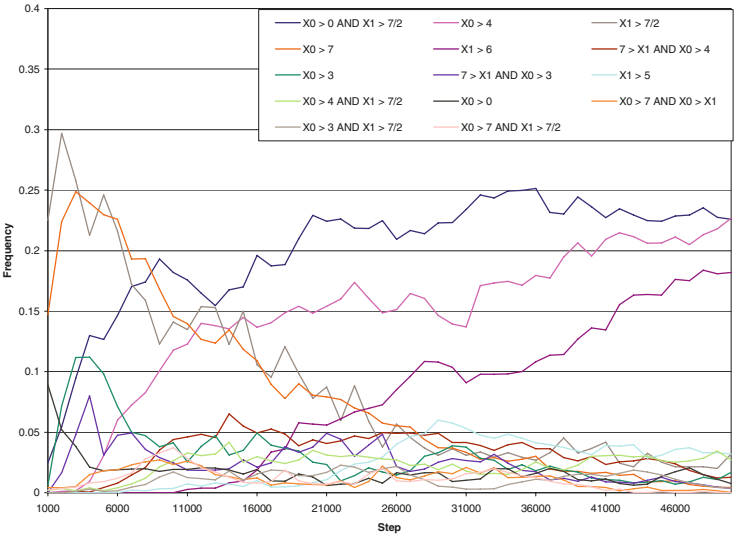


Fig. 8. Trend of the frequencies of recurrent subexpressions of Experiment 1, second run. The advocated action is 1.

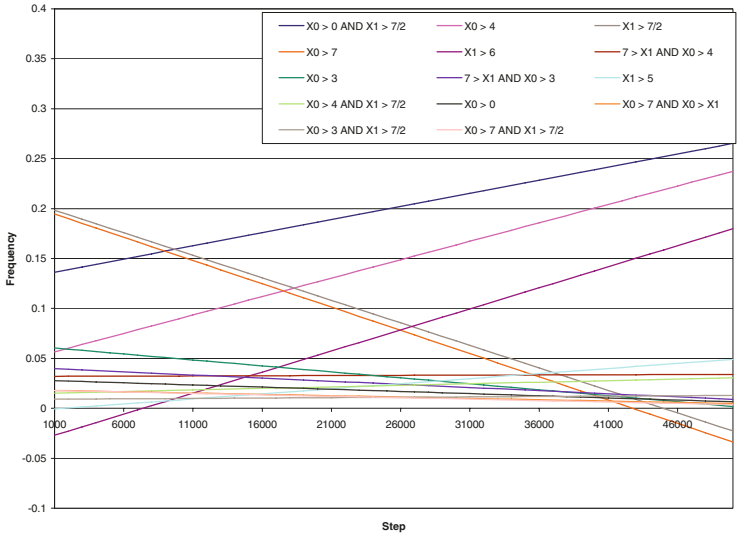


Fig. 9. Linear regression of the frequencies of recurrent subexpressions of Experiment 1, second run. The advocated action is 1.

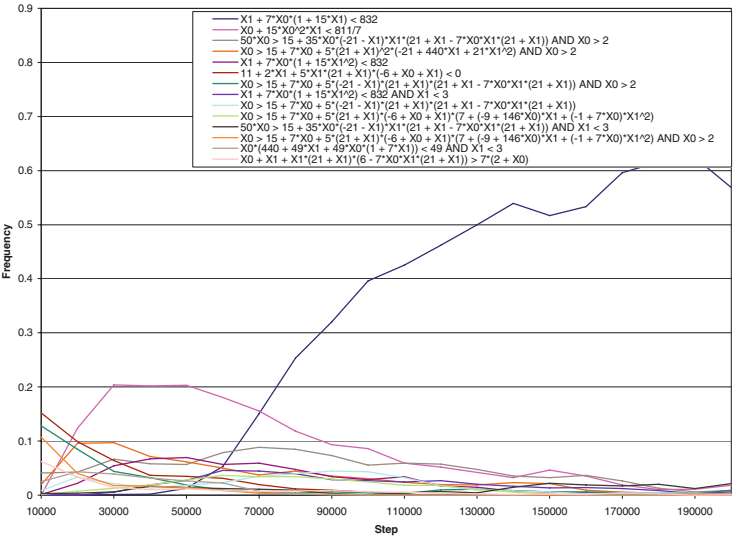


Fig. 10. Trend of the frequencies of recurrent subexpressions of Experiment 2, first run. The advocated action is 0.

for action 0. From the plot it is also possible to note that, when $X0, X1 \in \mathbb{N}$, "$X1+7*X0*(1+15*X1) < 832$" is equivalent to "$X0*X1 \leq 7$". Figure 12 also shows the area of the subexpression "$X0+15*X0^2*X1 < 811/7$" which is more specific than "$X1+7*X0*(1+15*X1) < 832$". In fact, the former condition

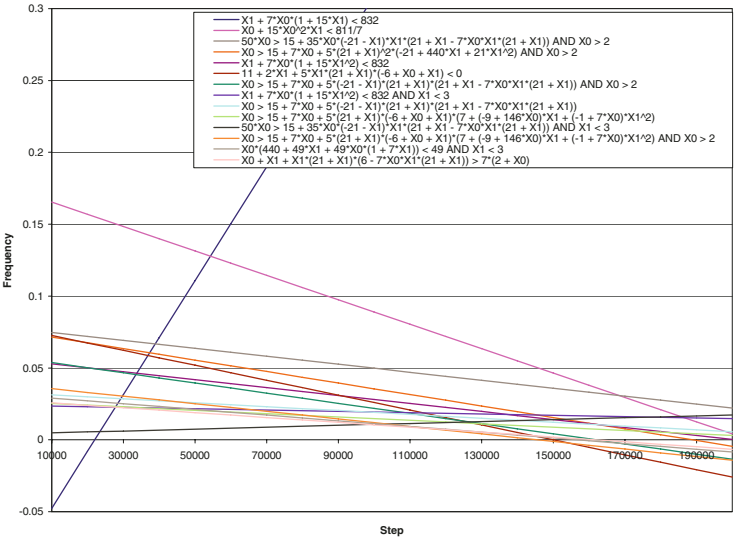


Fig. 11. Linear regression of the frequencies of recurrent subexpressions of Experiment 2, first run. The advocated action is 0.

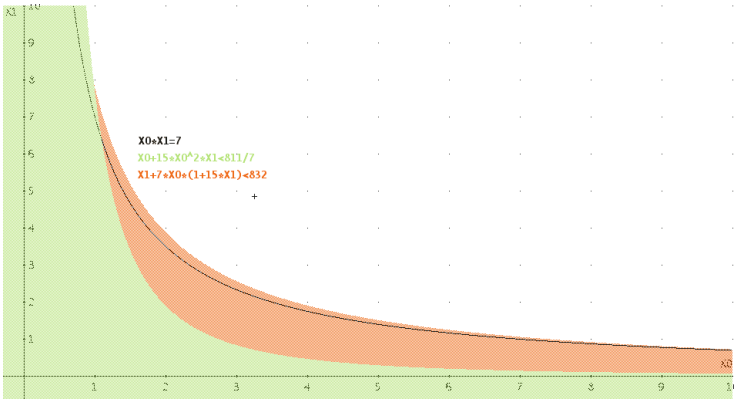


Fig. 12. Curve "$X0 * X1 = 7$" and subexpressions "$X1 + 7 * X0 * (1 + 15 * X1) < 832$" and "$X0 + 15 * X0^2 * X1 < 811/7$"

has an increasing trend for the first 30000 steps until the more general condition appears; then its frequency remains constant and after step 50000 it decreases while the frequency of the more general "$X1 + 7 * X0 * (1 + 15 * X1) < 832$" increases.

Then, we analyze the classifiers that advocate action 1 which represent the inequality "$X0 * X1 > 7$". Figure 13 shows the trend of recurrent subexpressions in the population. There is a subexpression that has a much higher frequency and many subexpressions that behave similarly such as "$16 * X0 * X1 >$

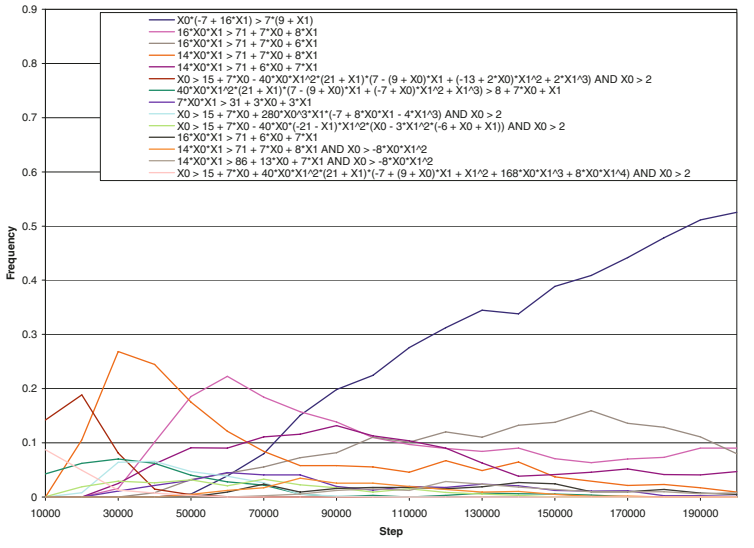


Fig. 13. Trend of the frequencies of recurrent subexpressions of Experiment 2, first run. The advocated action is 1.

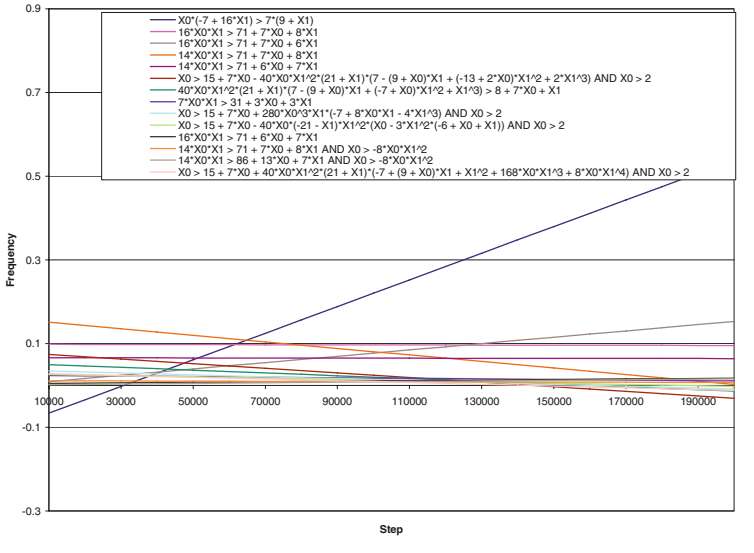


Fig. 14. Linear regression of the frequencies of recurrent subexpressions of Experiment 2, first run. The advocated action is 1.

$71+7*X0+8*X1$" and "$16*X0*X1 > 71+7*X0+6*X1$". Figure 14 reports the linear model extracted from the same data. From this plots we identify two subexpressions with an increasing trend: (i) "$X0*(-7+16*X1) > 7*(9+X1)$",

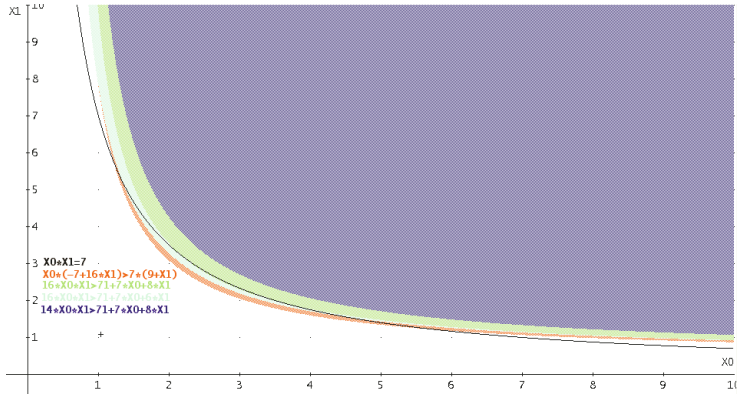


Fig. 15. Curve "$X0 * X1 = 7$"and subexpressions "$X0 * (-7 + 16 * X1) > 7 * (9 + X1)$", "$16 * X0 * X1 > 71 + 7 * X0 + 6 * X1$", "$16 * X0 * X1 > 71 + 7 * X0 + 6 * X1$", and "$14 * X0 * X1 > 71 + 7 * X0 + 8 * X1$"

and (ii) "$16 * X0 * X1 > 71 + 7 * X0 + 6 * X1$". The two subexpressions are very similar, (i) is equal to "$16 * X0 * X1 > 63 + 7 * X0 + 7 * X1$", and to many others subexpressions but they are the only ones with an increasing trend of the frequency. Figure 15 shows the areas of (i), (ii) and other two subexpressions with a very similar form, in addition to the curve "$X0 * X1 = 7$". From Figure 15 we note that (i) is the most general and, given that $X0, X1 \in \mathbb{N}$,

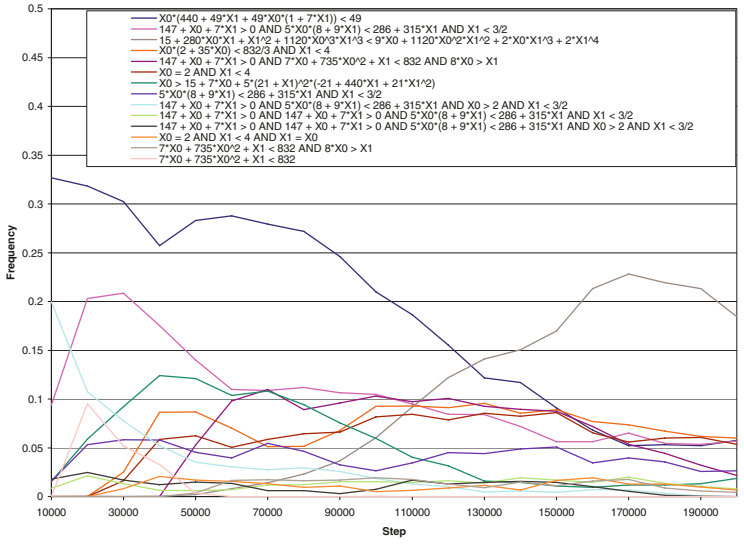


Fig. 16. Trend of the frequencies of recurrent subexpressions of Experiment 2, second run. The advocated action is 0.

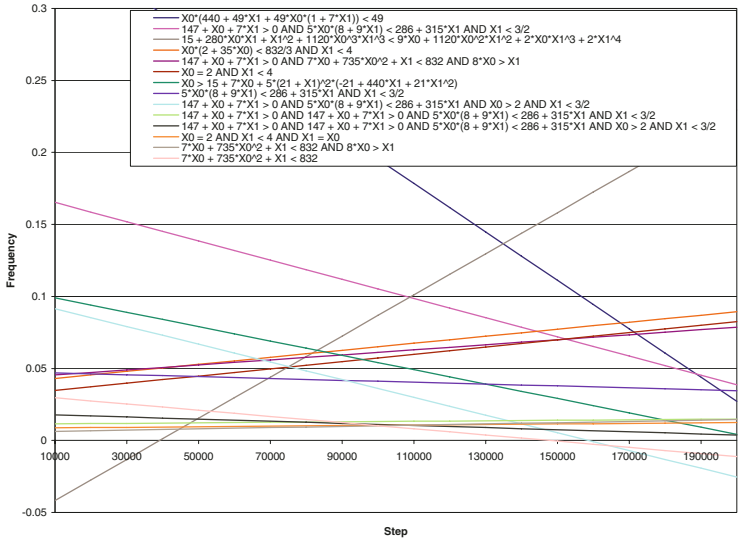


Fig. 17. Linear regression of the frequencies of recurrent subexpressions of Experiment 2, second run. The advocated action is 0.

it is equal to "$X0 * X1 > 7$". The subexpression (ii) is more specific, but its frequency grows very slowly compared to the subexpression (i). The other two subexpressions represented are "$16 * X0 * X1 > 71 + 7 * X0 + 6 * X1$", and "$14 * X0 * X1 > 71 + 7 * X0 + 8 * X1$". The frequencies of the last two subexpressions grow in the first steps and then decreases in favor of more general

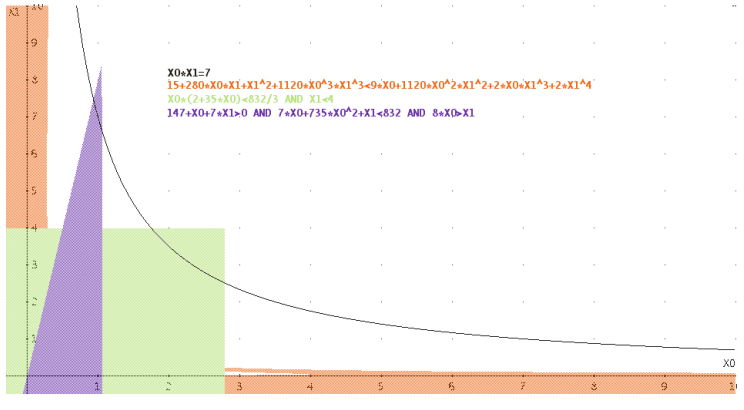


Fig. 18. Curve "$X0 * X1 = 7$" and subexpressions that have an increasing frequency trend. Experiment 2, second run, advocated action 0.

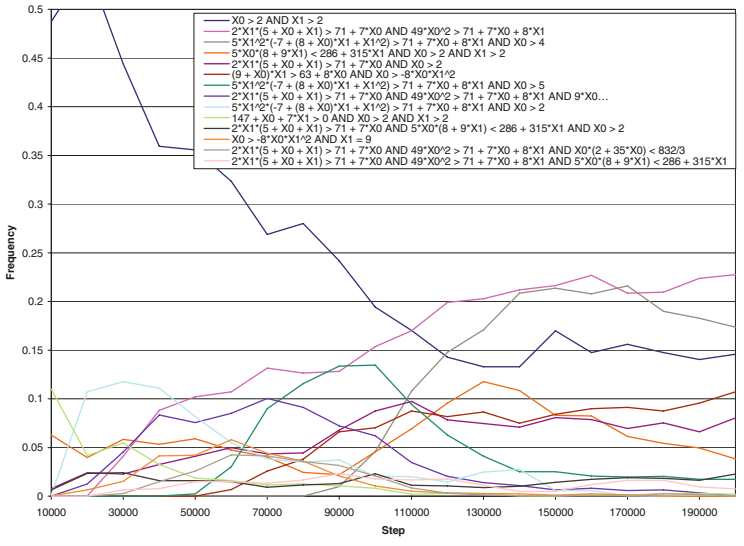


Fig. 19. Trend of the frequencies of recurrent subexpressions of Experiment 2, second run. The advocated action is 1.

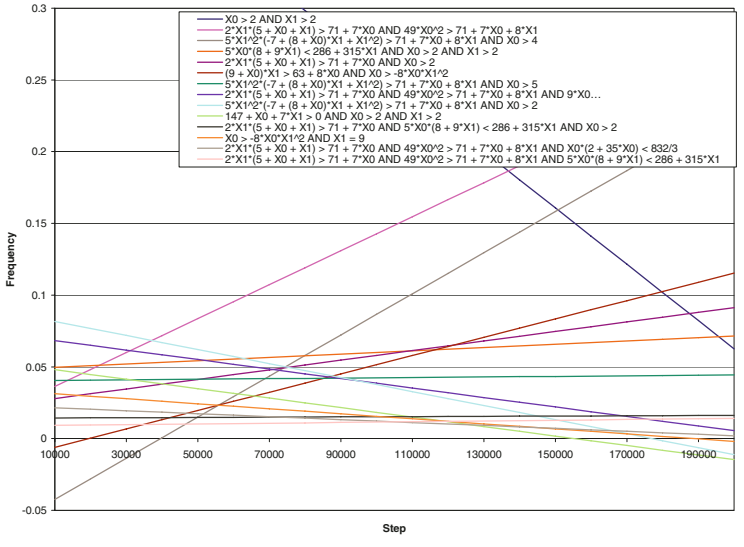


Fig. 20. Linear regression of the frequencies of recurrent subexpressions of Experiment 2, second run. The advocated action is 1.

subexpressions like (i). These plots confirm what noted in the previous experiments: the subexpression with an increasing frequency trend, (i) and (ii), represent a complete solution for action 1.

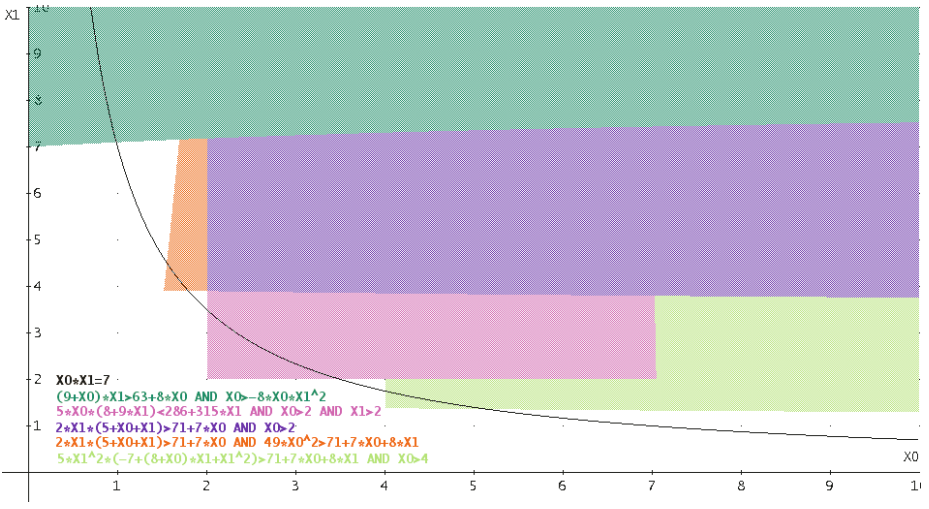


Fig. 21. Curve "$X0 * X1 = 7$" and subexpressions that have an increasing frequency trend. Experiment 2, second run, advocated action 1.

The second run does not reach optimality, Therefore, all the evolved populations contain classifiers with a low fitness. We first analyze the classifiers that advocate action 0 representing the inequality "$X0 * X1 \leq 7$". Figure 16 shows the trend of the recurrent subexpressions with the higher average frequency; Figure 17 reports the results of the linear regression applied to the same data. There are four recurrent subexpressions with an increasing trend: one is "$X0 = 2$ AND $X1 < 4$", the areas of the other three subexpressions are reported in Figure 18. The four subexpressions do not represent "$X0 * X1 \leq 7$" completely but they are close. In fact, with more learning problems, the population would probably converge to optimality; but when the learning was stop, the maximally accurate maximally general solution was not present in the population. We then analyze the classifiers that advocate action 1 which represent the inequality "$X0 * X1 > 7$". Figure 19 shows the trend of the recurrent subexpressions with the higher average frequency. Analyzing the linear regression of the frequencies reported in Figure 20 we extract 5 subexpressions with an increasing frequency trend. The areas represented by these subexpressions are reported in Figure 21. Note that the five subexpressions do not represent the complete solution to the problem for action 1. As before, XCS was stopped before it could reach optimality. However the five subexpressions with an increasing trend represent interesting information about the problem.

6 Summary

We have introduced a method to analyze the evolution of XCS with symbolic conditions. We have applied our approach to analyze two experiments involving

the learning of a simple numeric expression. By applying the proposed method, we have been able to show how XCS with GP-based conditions, XCSGP, builds up an optimal solution by favoring the evolution of *good* subexpressions that are useful with respect to the target problem. Our approach can also be used in a constructive way: after the frequent subexpressions have been identified, they can also be used to guide the generation of better offspring [10].

References

1. Booker, L.B.: Representing Attribute-Based Concepts in a Classifier System. In: Gregory, J.E. (ed.) Proceedings of the First Workshop on Foundations of Genetic Algorithms (FOGA 1991), pp. 115–127. Morgan Kaufmann, San Mateo (1991)
2. Dignum, S., Poli, R.: Generalisation of the limiting distribution of program sizes in tree-based genetic programming and analysis of its effects on bloat. In: GECCO 2007: Proceedings of the 9th annual conference on Genetic and evolutionary computation, pp. 1588–1595. ACM Press, New York (2007)
3. Koza, J.R.: Hierarchical automatic function definition in genetic programming. In: Whitley, L.D. (ed.) Foundations of Genetic Algorithms 2, Vail, Colorado, USA, 24–29, 1992, pp. 297–318. Morgan Kaufmann, San Francisco (1992)
4. Lanzi, P.L.: Extending the Representation of Classifier Conditions Part I: From Binary to Messy Coding. In: Banzhaf, W., et al. (ed.) Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999), Orlando (FL), July 1999, pp. 337–344. Morgan Kaufmann, San Francisco (1999)
5. Lanzi, P.L.: Mining interesting knowledge from data with the XCS classifier system. In: Spector, L., Goodman, E.D., Wu, A., Langdon, W.B., Voigt, H.-M., Gen, M., Sen, S., Dorigo, M., Pezeshk, S., Garzon, M.H., Burke, E. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), 7-11 July 2001, pp. 958–965. Morgan Kaufmann, San Francisco (2001)
6. Lanzi, P.L., Perrucci, A.: Extending the Representation of Classifier Conditions Part II: From Messy Coding to S-Expressions. In: Banzhaf, W., et al. (ed.) Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999), Orlando (FL), July 1999, pp. 345–352. Morgan Kaufmann, San Francisco (1999)
7. Luke, S., Panait, L.: A comparison of bloat control methods for genetic programming. Evolutionary Computation 14, 309–344 (2006)
8. Poli, R.: A simple but theoretically-motivated method to control bloat in genetic programming. In: Ryan, C., Soule, T., Keijzer, M., Tsang, E.P.K., Poli, R., Costa, E. (eds.) EuroGP 2003. LNCS, vol. 2610, pp. 204–217. Springer, Heidelberg (2003)
9. Wolfram Research. Mathematica 5, `http://www.wolfram.com`
10. Rocca, S., Solari, S.: Building blocks analysis and exploitation in genetic programming. Master's thesis (April 2006) Master thesis supervisor: Prof. Pier Luca Lanzi. Electronic version available from, `http://www.dei.polimi.it/people/lanzi`
11. Schaffer, J.D. (ed.): Proceedings of the 3rd International Conference on Genetic Algorithms (ICGA 1989), George Mason University, June 1989. Morgan Kaufmann, San Francisco (1989)
12. Schuurmans, D., Schaeffer, J.: Representational Difficulties with Classifier Systems. In: Schaffer (ed.) [11], pp. 328–333, `http://www.cs.ualberta.ca/~jonathan/Papers/Papers/classifier.ps`
13. Sen, S.: A Tale of two representations. In: Proc. 7th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, pp. 245–254 (1994)

14. Shu, L., Schaeffer, J.: VCS: Variable Classifier System. In: Schaffer (ed.) [11], pp. 334–339, http://www.cs.ualberta.ca/~jonathan/Papers/Papers/vcs.ps
15. Wilson, S.W.: Get real! XCS with continuous-valued inputs. In: Booker, L., Forrest, S., Mitchell, M., Riolo, R.L. (eds.) Festschrift in Honor of John H. Holland, pp. 111–121. Center for the Study of Complex Systems (1999), http://prediction-dynamics.com/
16. Wilson, S.W.: Mining Oblique Data with XCS. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 2000. LNCS (LNAI), vol. 1996. Springer, Heidelberg (2001)
17. Wilson, S.W.: Mining oblique data with XCS. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 2000. LNCS (LNAI), vol. 1996, pp. 158–176. Springer, Heidelberg (2001)

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

Charalambos Ioannides and Will Browne

Cybernetics, University of Reading, Reading, UK
siu03ci@rdg.ac.uk
w.n.browne@rdg.ac.uk

Abstract. Utilising the expressive power of S-Expressions in Learning Classifier Systems often prohibitively increases the search space due to increased flexibility of the encoding. This work shows that selection of appropriate S-Expression functions through domain knowledge improves scaling in problems, as expected. It is also known that simple alphabets perform well on relatively small sized problems in a domain, e.g. ternary alphabet in the 6, 11 and 20 bit MUX domain. Once fit ternary rules have been formed it was investigated whether higher order learning was possible and whether this staged learning facilitated selection of appropriate functions in complex alphabets, e.g. selection of S-Expression functions. This novel methodology is shown to provide compact results (135-MUX) and exhibits potential for scaling well (1034-MUX), but is only a small step towards introducing abstraction to LCS.

1 Introduction

Although the scalability of Learning Classifier System [1][2] technique is good (70 Multiplexer MUX problem solvable) the increased dimensionality eventually restricts the technique (135 MUX problem is difficult to solve). The search space increase is inherent in the problem and the representative space of the alphabet. Introducing S-Expressions as an LCS may reduce the problem search space by mapping to a different problem description, but this will be at the cost of increasing the mappings available and hence increasing the representative search space.

Learning Classifier Systems (LCS) were originally designed as artificial cognitive systems [3], but are now established as an effective machine learning technique. LCS's power lies in combining Reinforcement Learning and Evolutionary Algorithms in order to create and promote a set of production (IF condition THEN action) rules. Generalisation is core to LCS, but it is hypothesized here that the learning of concept-based knowledge related to experiences can be improved by developing abstraction methods.

Generalisation enables a LCS to ignore unimportant conditions in environmental messages – similar to humans storing just the key features of an episode in memory. However, it is postulated that humans also abstract higher order patterns from these stored episodes into semantic memory, which is not currently addressed in LCS.

J. Bacardit et al. (Eds.): IWLCS 2006/2007, LNAI 4998, pp. 46–56, 2008.

The benefit of higher order patterns is in scaling within a problem domain as they capture higher order features, e.g. the importance of address bits in the MUX problem domain. This work attempts to mimic human behaviour by generating features from small problem solutions, through utilising appropriate alphabets to facilitate abstraction, which will assist in learning larger scale problems. More practically; it is to be investigated if domain-relevant functions will assist learning of multiplexers both in the sense of faster learning and smaller s-expressions. Without domain knowledge the appropriate function(s) for a problem need to be autonomously discovered.

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

The widely used LCS, XCS (Accuracy-based Fitness LCS) [4], has been well tested [5][6]. XCS scales well on the MUX problems as the learning is polynomial complexity [9]. However, XCS still struggles to solve the 135-MUX problem due to large processing requirements whereas humans are capable of understanding the concept behind all MUX problems. The aim of this work is to develop a method based on abstraction so that XCS scales to increasing sized problems of similar type.

2 Background

In the classical XCS framework [2] the classifiers that represent the environment use ternary alphabet {0, 1, #} and the assignment of an Action to a classifier is inflexible (i.e. preset and unchanging throughout the classifier's 'life'). The use of the # (don't care) symbol further limits the potential for flexibility since it is simply a positional relation acting on a specific position within the classifier's condition string rather than between its conditions. It acts as the OR function (i.e. '0' OR '1'). In effect this (#) facilitates the important feature of generalization in the XCS paradigm.

The main characteristics that any single classifier possesses are:

1. Condition part $C - C \in \{0,1,\#\}^l$
2. Action part $A - A \in \{0,1\}$
3. Reward Prediction R
4. Estimated Error ε
5. Fitness F

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

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

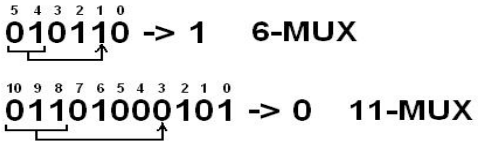


Fig. 1. Multiplexer problem

3 Design of the S-XCS System

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

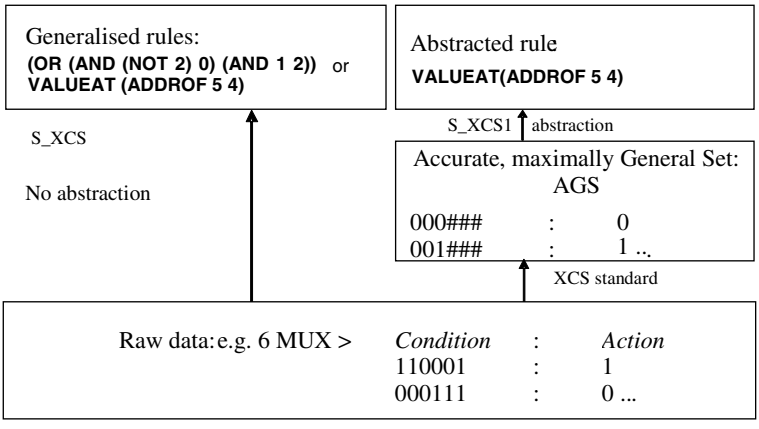


Fig. 2. Developed framework for investigating abstraction

The episodes were learnt as in a classical XCS to produce a maximally general accurate population, termed Accurate, maximally General Set (AGS). This AGS was then the environment where each generalised rule was an input message passed to the S-Expression based XCS, termed S-XCS1. This was contrasted with a single population version (without AGS) that directly learnt from the raw environmental messages, again utilising the S-Expression alphabet, in order to discover any benefits of abstraction.

Following the S-Expression paradigm [7][8] the classifiers' building blocks consists of either non-terminals (i.e. functions) or terminals (i.e. variables or constants), see figure 3. These can be tailored to the domain, e.g. VALUEAT which returns the bit value of a position within the classifier and ADDROF which returns the integer counterpart of a string of bits. Firstly, tailored functions are tested to confirm (or otherwise) that pre-solving the problem with domain knowledge is possible in this setting. Secondly, it is to be tested whether the most appropriate tailored functions can be identified automatically. Tailored functions rely upon domain knowledge, which reduces the applicability of the technique unless used with other general functions.

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

<var> := "0" | "1" |"2" | ….. "k"
```

Fig. 3. Backus–Naur form grammar that generates classifiers

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

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

Other methods were also appropriately altered, e.g. Classifier Covering or completely removed, e.g. Subsumption Deletion. Coverage was triggered when the average fitness was below 0.5 in an action set and created a random valid tree. In initial tests numerosity tended to be small due to the diversity of the S-Expressions, which tended to distort fitness measurements. Thus, numerosity was not utilised in the fitness update and absolute accuracy was used as fitness instead of the relative accuracy.

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

4 Results

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

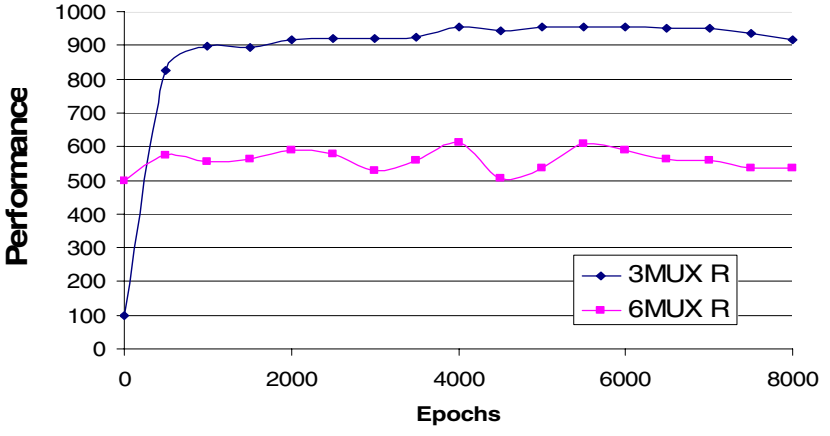


Fig. 4. Performance - the weighted average of 10 runs of the average fitness of the last 50 epochs (exploit trials) of the population - of S_XCS on 3-MUX & 6-MUX problem

System parameters are learning rate β=0.2, initial error ε=0.5, accuracy parameters α=0.1, ε_0=10, r_{imm}=1000, ν=5, GA threshold θ_{GA}=25, experience deletion threshold θ_{del}=20, mutation rate μ=0.01, tournament size τ=0.2, crossover possibility χ=1, population size N=400 [2].

Table 1. S_XCS on 3-MUX Optimal Population extract

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

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

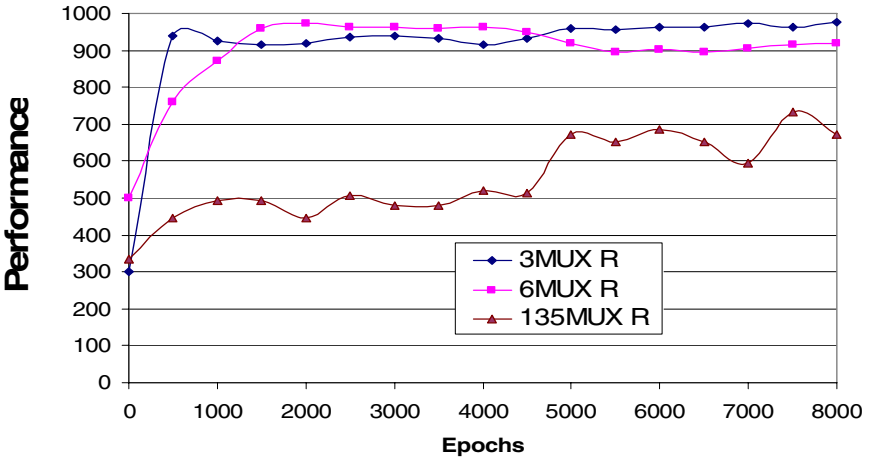


Fig. 5. S_XCS1 without AGS on 3-MUX, 6-MUX & 135-MUX problem (8000 epochs)

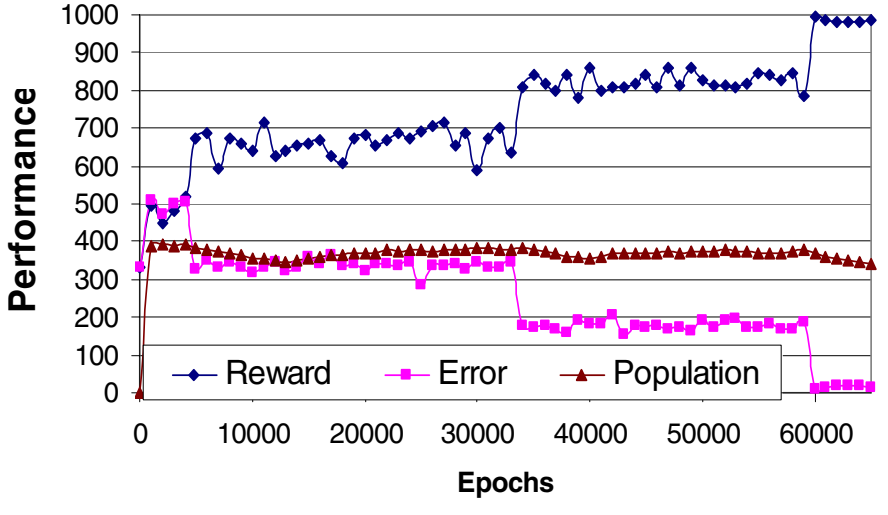


Fig. 6. S_XCS1 without AGS on 135-MUX problem (65000 epochs)

The system will discover the most compact functions suited to the domain, provided sufficient time is allowed; see figure 6 & table 2. This compaction time was greatly reduced by the use of abstraction, compare figure 6 with 8, where generalisation is accomplished first using the ternary alphabet (where possible) and then higher order knowledge is gained by training on the optimum set, see figure 8.

It was hypothesised that the results of training on small sized domains could aid the selection of tailored functions in larger domains. A limited set of the functions most suited to the MUX problem domain was discovered on solvable domains, e.g. MUX < 70. These were utilised in S_XCS1, but the results of did not provide significant advantage, comparing figure 7 with figure 5.

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

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

The known ternary solution, AGS, was tested as the optimum set can not be learnt practically for >135 MUX. Thus, this tests the potential scalability of S-Expressions when used for higher order learning, rather than the ability to learn. The system scales well, see figure 8, with the best discovered solutions shown in tables 3 (highest fitness in bold). Unnecessary functions, such as POWEROF, were less likely to appear with abstraction than without, see table 2. An example of a maximally compact rule produced was for the 135 MUX, where the value returned is that addressed by the bit string between the terminals128-134.

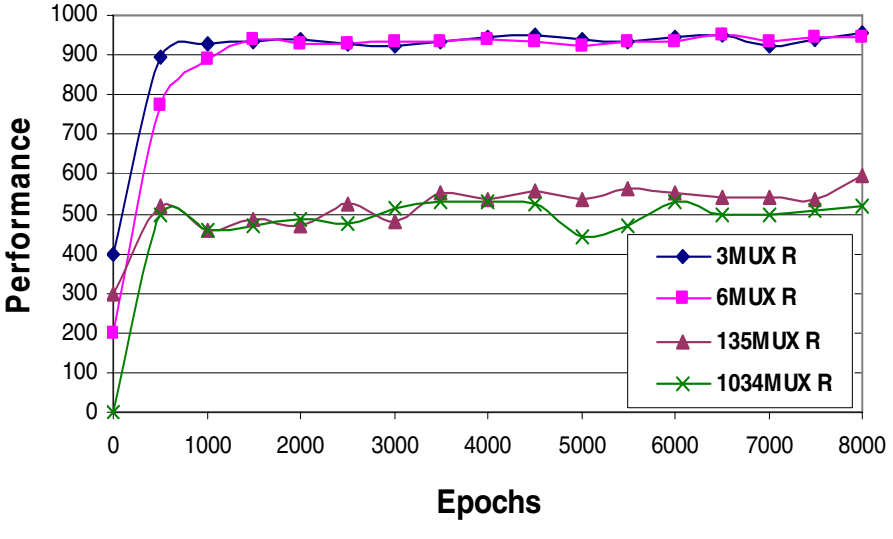


Fig. 7. S_XCS1 without AGS on 3-MUX, 6-MUX, 135-MUX & 1034-MUX problem, identified tailored functions

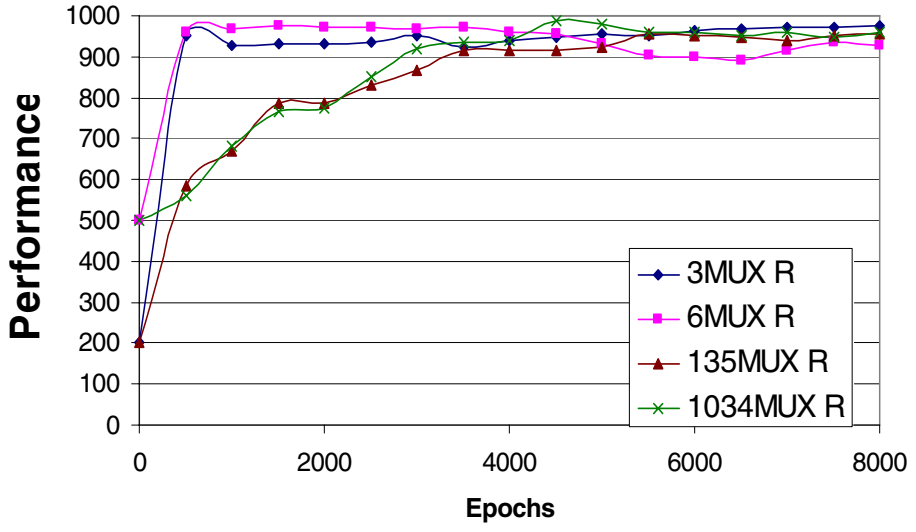


Fig. 8. S_XCS1 with known AGS on 3-MUX, 6-MUX, 135-MUX & 1034-MUX problem

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

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

5 Discussion

The S_XCS system (without tailored functions) does exhibit potential in developing solutions that resemble the Disjunctive or the Conjunctive Normal Form of the k-MUX problem, but it did not scale well. The reason for this was that few partially correct classifiers exist, hence no building blocks and thus the Genetic Algorithm was redundant. Coverage only could create a correct classifier. The values of environmental variables could effectively change at every epoch and the functions (AND, OR, NOT) were too granular to cover the change.

The S_XCS1 without abstraction system overcame the S_XCS problems and discovered optimal classifiers, including 135-MUX problem. Importantly, this system selected the most appropriate tailored functions and terminals for the domain, ignoring the majority of the 10 functions available. Including some poorly tailored functions did not incur population bloating or larger search times. One of the justifications for developing abstraction was to reduce the increased search space caused by more flexible alphabets, but the ability of XCS to select appropriate building blocks appears to negate this problem.

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

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

Redesigning subsumption deletion for S-Expressions is unnecessary for compacting the rule base as the coverage of each classifier is much greater, i.e. only one classifier is needed for the correct, accurate mapping. However, subsumption deletion may be needed for stabilising the performance close to optimum and potentially for the control of bloat. With many different functions, it is difficult to determine the coverage and hence overlap of each individual classifier. Thus the evolved population consists of many similar classifiers with similar fitness, which would require a tuned deletion operator to remove slightly suboptimal classifiers.

Utilising the same alphabet as Genetic Programming (GP) results in many similarities between GP and the techniques developed here, e.g. a single classifier solution instead of a cooperation of classifiers. However, a guided population based search is employed instead of a beam search, where the current best classifiers are evolved [10]. The online learning abilities together with the inherent generalisation pressure of XCS are still preserved. In order to prevent bloat the maximum depth of a classifier's condition required limiting (e.g., to 3 in 6-MUX problem), but was simple to implement. The need for a multistage approach to learning is demonstrated in the layered learning genetic programming approach [11].

Comparing the use of abstraction on the MUX problem (figures 7 and 8) showed that abstraction assists the S-Expression XCS to learn more compactly as unimportant aspects of the search space had already been removed by generalisation. In the domain tested, abstraction did not improve efficiency as the accurate, maximally general set had to be created initially. Similarly, abstraction did not improve effectiveness due to XCS being able to discover appropriate building blocks. Abstraction reduced the quantity of irrelevant functions in a classifier during the initial stages of training, which may prevent the system becoming trapped in local optima in other problem domains. Abstraction also scaled well on hypothesised AGSs, which were tested up to 1034MUX.

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

6 Conclusions

Positive results were achieved in that abstraction can compact a rule base using higher order functions once generalisation has occurred using a simple alphabet. Abstraction is also efficient and can scale well - once the initial rule base has been formed when possible.

However, abstraction was not needed to identify the most appropriate tailored functions as XCS has the ability to identify good building blocks from potentially useful tailored functions. The next step in the development of an abstraction algorithm for LCS is to generate the tailored functions autonomously from generalised rules without using domain knowledge.

References

1. Wilson, S.W.: ZCS: A Zeroth-level Classifier System. Evolutionary Computation 2(1), 1–18 (1994)
2. Wilson, S.W.: Classifier Fitness Based on Accuracy. Evolutionary Computation 3(2), 149–176 (1995)
3. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press (1975)
4. Wilson, S.W.: Generalization in the XCS Classifier System. In: Koza, J., et al. (eds.) Genetic Programming: Proceedings of the Third Annual Conference, pp. 665–674. Morgan Kaufmann, San Francisco (1998)
5. Kovacs, T.: Evolving Optimal Populations with XCS Classifier Systems. MSc Thesis, University of Birmingham. Also Technical Report CSR-96-17 and CSRP-96-17, School of Computer Science, University of Birmingham, Birmingham, UK (1996)
6. Kovacs, T.: XCS Classifier System Reliably Evolves Accurate, Complete, and Minimal Representations for Boolean Functions. In: Roy, Chawdhry, Pant. (eds.) Soft Computing in Engineering Design and Manufacturing, pp. 59–68. Springer, Heidelberg (1997)

7. Koza, J.: Genetic Programming. MIT Press, Cambridge (1992)
8. Lanzi, P.-L., Perrucci, A.: Extending the Representation of Classifier Conditions Part II: From Messy Coding to S-Expressions. In: Proceedings of the Genetic and Evolutionary Computation Conference, Orlando, Florida, pp. 345–352. Morgan Kaufmann, San Francisco (1999)
9. Butz, M.V.: Rule-Based Evolutionary Online Learning Systems: A Principled Approach to LCS Analysis and Design. Studies in Fuzziness and Soft Computing, vol. 191 (2006)
10. Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D.: Genetic Programming - An Introduction. Morgan Kaufmann Publishers, San Francisco and dpunkt.verlag, Heidelberg (1998)
11. Gustafson, S.M., Hsu, W.H.: Layered learning in genetic programming for a cooperative robot soccer problem. In: Miller, J., Tomassini, M., Lanzi, P.L., Ryan, C., Tetamanzi, A.G.B., Langdon, W.B. (eds.) EuroGP 2001. LNCS, vol. 2038, pp. 291–301. Springer, Heidelberg (2001)
12. Browne, W., Scott, D.: An abstraction algorithm for genetics-based reinforcement learning. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1875–1882. ACM Press, Washington (2005)

Evolving Fuzzy Rules with UCS: Preliminary Results

Albert Orriols-Puig[1], Jorge Casillas[2], and Ester Bernadó-Mansilla[1]

[1]Grup de Recerca en Sistemes Intel·ligents
Enginyeria i Arquitectura La Salle
Universitat Ramon Llull
Quatre Camins 2, 08022 Barcelona, Spain
{aorriols,esterb}@salle.url.edu
[2]Dept. Computer Science and Artificial Intelligence
University of Granada
18071, Granada, Spain
casillas@ugr.es

Abstract. This paper presents Fuzzy-UCS, a Michigan-style Learning Fuzzy-Classifier System designed for supervised learning tasks. Fuzzy-UCS combines the generalization capabilities of UCS with the good interpretability of fuzzy rules to evolve highly accurate and understandable rule sets. Fuzzy-UCS is tested on a large collection of real-world problems, and compared to UCS and three highly-used machine learning techniques: the decision tree C4.5, the support vector machine SMO, and the fuzzy boosting algorithm Fuzzy LogitBoost. The results show that Fuzzy-UCS is highly competitive with respect to the four learners in terms of performance, and that the fuzzy representation permits a much better understandability of the evolved knowledge. These promising results of the online architecture of Fuzzy-UCS allow for further research and application of the system to new challenging problems.

1 Introduction

Michigan-style Learning Classifier Systems (LCSs) [19] are online machine learning techniques that use Genetic Algorithms (GAs) [19,18] to evolve a rule-based knowledge. Among the different uses, several LCSs have been designed for performing supervised learning tasks [34,4,3]. Typically, LCSs deal with numerical attributes by means of evolving a set of interval-based rules that cooperate to predict the output of new unlabeled examples. Although the competence of LCSs in terms of accuracy has been widely shown, this excellence has been hindered by a poor interpretability of the evolved rule sets, which typically consist of large sets of overlapping interval-based rules that can hardly be read by human experts.

During the last decade, the interest in Fuzzy Rule-Based Systems (FRBSs) [11] has increased since they provide a robust, flexible, and powerful methodology to deal with *noisy*, *imprecise*, and *incomplete* data. Besides, the fuzzy representation allows for a better interpretability of the classification models. This has led

J. Bacardit et al. (Eds.): IWLCS 2006/2007, LNAI 4998, pp. 57–76, 2008.

to the first analyses and designs of *Learning Fuzzy-Classifier Systems* (LFCSs). Since the introduction of the first LFCS [30], several new architectures have been proposed [23,17,32,6,7,9], which have been mostly applied to reinforcement learning and control tasks. One of the first proposals of LFCS for pattern classification was presented in [20]. These first successful steps toward the design of competent LFCS warrants for further investigation, especially in the supervised learning paradigm.

In this paper, we address the problem of interpretability in LCSs, and propose Fuzzy-UCS, an online accuracy-based LFCS architecture that works under a supervised learning paradigm. We depart from the UCS classifier system, which has been shown to be highly competitive with respect to some of the most used machine learning techniques [3]. We introduce a linguistic fuzzy representation to UCS, and redesign most of its components to permit the system to deal with fuzzy rules. With the inclusion of fuzzy rules, we seek for a better interpretability of the evolved knowledge, as well as a reduction in the search space, while maintaining a performance similar to the one obtained with an interval-based representation. Moreover, we also prepare the system to be able to deal with vague and uncertain data.

The remaining of this paper is organized as follows. Section 2 deeply explains the proposed Fuzzy-UCS architecture, especially focusing on the differences from the original UCS. In Sect. 3, we analyze the behavior of Fuzzy-UCS on a large collection of real-world problems, and compare the performance an interpretability of the models evolved by Fuzzy-UCS to those created by UCS and three other machine learning techniques: C4.5, SMO, and Fuzzy LogitBoost. Finally, Sect. 4 concludes, highlights the differential traits of Fuzzy-UCS, and enumerates new opportunities that will be addressed as further work.

2 Description of Fuzzy-UCS

Figure 1 schematically shows the learning process of Fuzzy-UCS. The learner works in two different modes: training or *exploration* mode and testing or *exploitation* mode. During explore, Fuzzy-UCS evaluates online the quality of the rule-based knowledge, and evolves it by means of a GA. During test, Fuzzy-UCS uses the rules to infer the output of a given input instance. The different components of the system are detailed as follows.

2.1 Representation

Fuzzy-UCS evolves a *population* [P] of classifiers, where each classifier consists of a *linguistic fuzzy rule* and a set of parameters. The fuzzy rule is represented as follows:

$$\textbf{IF } x_1 \text{ is } \widetilde{A_1^k} \text{ and } \cdots \text{ and } x_n \text{ is } \widetilde{A_n^k} \textbf{ THEN } c^k \textbf{ WITH } F^k \tag{1}$$

where each input variable x_i is represented by a disjunction (T-conorm operator) of linguistic terms $\widetilde{A_i^k} = \{A_{i1} \vee ... \vee A_{in_i}\}$. In our experiments, all the input

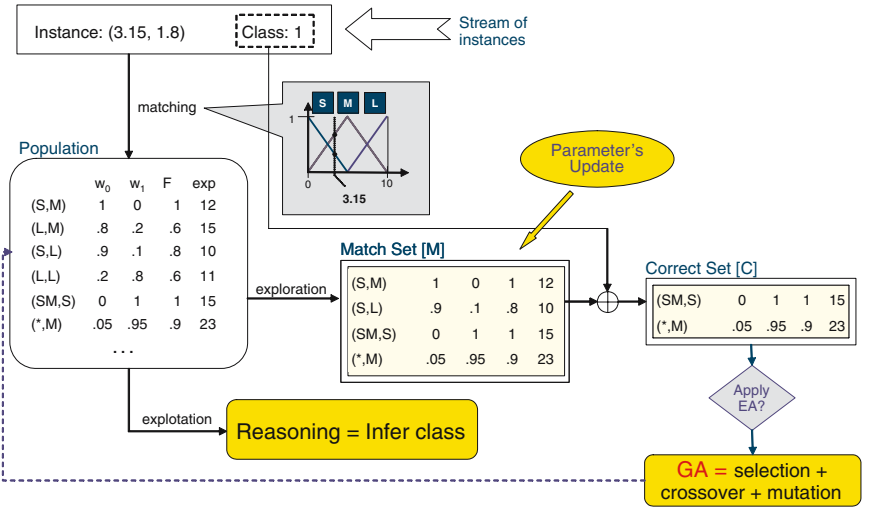


Fig. 1. Schematic illustration of Fuzzy-UCS

variables share the same semantics. The variables are defined by triangular-shaped fuzzy membership functions (see examples of these semantics with three and five linguistic terms per variable in Fig. 2).

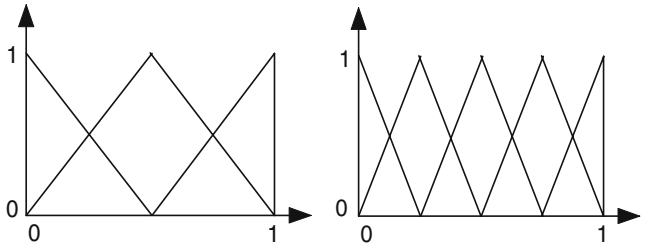
The consequent of the rule internally maintains one weight for each of the m classes $\{w_1^k, \cdots, w_m^k\}$. Each weight w_j^k indicates the soundness with which the rule k predicts class j for an example that fully matches this rule. These weights are incrementally updated with the learning interaction (see Sect. 2.3), and serve to calculate the class of the rule. That is, the class c^k predicted by the rule is the class that has associated the weight with maximum value.

Each classifier has four main parameters: a) the fitness F, which estimates the accuracy of the rule, b) the correct set size cs, which averages the sizes of the correct sets in which the classifier has participated, c) the experience exp, which reckons the contributions of the rule to classify the input instances, and d) the numerosity num, which counts the number of copies of the classifier in the population. All these parameters are updated online as specified in Sect. 2.3.

To implement this representation, we propose to use a binary coding for the antecedent of the rule. That is, a one-valued allele indicates that the corresponding linguistic term is used in this variable. The class predicted by the rule is codified as an integer, and the fitness as a float number. For instance, if we have three linguistic labels {S [small], M [medium], L [large]} for each input and two possible classes $\{c_1, c_2\}$, the fuzzy rule

$$\textbf{IF } x_1 \text{ is } S \text{ and } x_2 \text{ is } \{S \text{ or } L\} \textbf{ THEN } c_1 \textbf{ WITH } 0.8 \tag{2}$$

is encoded as: [100|101||c1|0.8].



(a) 3 linguistic labels (b) 5 linguistic labels

Fig. 2. Representation of a fuzzy partition for a variable with (a) three and (b) five triangular-shaped membership functions

2.2 Performance Component

The performance component of Fuzzy-UCS is inherited from UCS and adapted to deal with the new fuzzy representation. UCS learns under a supervised learning scheme. Given an input example e with its associated class c, UCS creates the *match set* [M] with all the classifiers in [P] that *match* the input instance. Then, [M] is used differently depending on whether the system is running on explore or on exploit mode. In explore mode, UCS forms the *correct set* [C], which consists of all the classifiers in [M] that advocate the class of the input example. If [C] is empty, *covering* is triggered. In exploit mode, the best action selected from the vote (weighted by fitness) of all classifiers in [M] is returned as the predicted output.

Fuzzy-UCS follows this process, but the role of the matching and the inference processes changes as they are adapted to deal with linguistic terms. In the following, the matching degree calculation and the phases followed in explore mode are detailed. The inference mechanism used during test is explained in Sect. 2.5.

Calculation of the matching degree. The *matching degree* $\mu_{A^k}(e)$ of a rule k with an example e is computed as follows. For each variable x_i of the rule, we compute the membership degree for each of its linguistic terms, and aggregate them by means of a T-conorm (disjunction). Then, the matching degree of the rule is determined by the T-norm (conjunction) of the matching degree of all the input variables. In our implementation, we used a *bounded sum* ($min\{1, a+b\}$) as T-conorm and the *product* ($a \cdot b$) as T-norm. Note that we used a bounded sum instead of other typical operators for the T-conorm to emulate the *don't care* used in the crisp representation. That is, with the bounded sum, a variable will have no influence if all its linguistic terms are set to '1'.

Creation of the match set. Given the input e, all the classifiers with a *matching degree* greater than zero form the *match set*.

Creation of the correct set. Next, the *correct set* [C] is created with all the classifiers in [M] that advocate the class c. If there is not any rule in [C] that matches e with the maximum matching degree, the *covering operator* is

triggered. We create the rule that matches e with maximum degree as follows. For each variable x_i, the linguistic term with maximum matching with e_i is activated. Then, the rule is generalized by setting each linguistic term to '1' with probability $P_{\#}$. The parameters of the classifier are initialized to: F=1, exp=0, num=1, and cs is set to the size of [C]. Finally, this rule is introduced in the population if there is not any rule in [C] with the same matching degree.

2.3 Parameters Update

At the end of each learning iteration, the parameters of all the classifiers that belong to the match set are updated according to their matching degree with the input example e of class c. First, the experience of each rule is incremented according to the current matching degree:

$$exp_{t+1}^k = exp_t^k + \mu_{A^k}(e) \tag{3}$$

Then, for each class j, we compute the sum of correct matchings cm_j of each classifier k:

$$cm_{j_{t+1}}^k = cm_{j_t}^k + m(k,j) \tag{4}$$

where

$$m(k,j) = \begin{cases} \mu_{A^k}(e) & \text{if j=c} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

That is, we compute separately the sum of matching degrees of every rule with the examples of different classes. Next, the weight of each class is computed as:

$$w_{j_{t+1}}^k = \frac{cm_{j_{t+1}}^k}{exp_{t+1}^k} \tag{6}$$

For example, if a rule k only matches examples of class j, the weight w_j^k will be 1 and the remaining weights 0. Rules that match instances of more than one classes will have the corresponding weights ranging from 0 to 1. In all cases, the sum of all the weights is 1.

Then, the fitness is computed from the class weights with the aim of favoring classifiers that match instances of only one class. For this purpose, we compute the fitness as follows:

$$F_{t+1}^k = w_{max_{t+1}}^k - \sum_{j|j \neq max} w_{j_{t+1}}^k \tag{7}$$

The equation selects the weight w_{max}^k with maximum value and substract the values of the other weights. Note that this formula can result in classifiers with zero or negative fitness (for example, if the class weights are equal). Finally, the correct set size cs^k is calculated as the arithmetic average of the sizes of all the correct sets to which the classifier has belonged.

2.4 Discovery Component

Similarly to UCS, Fuzzy-UCS uses a *genetic algorithm* for discovering new promising rules. The GA is applied on a correct set if the average time since its last application on the classifiers that form this correct set is greater than θ_{GA}. In this case, two parents are selected from [C] with probability proportional to their fitness and the matching degree of their fitness. That is,

$$p_{sel}^k = \frac{(F^k)^\nu \cdot \mu_{A^k}(e)}{\sum_{i\in[C]|F^i\geq 0}(F^i)^\nu \cdot \mu_{A^k}(e)} \tag{8}$$

where $\nu > 0$ is a constant that fixes the pressure toward maximally accurate rules (in our experiments, we set ν=10). Rules with negative fitness are not considered for selection. The fitness of young classifiers is decreased since they receive a minimum number of updates. That is, if $exp_k < \theta_{exp}$, the fitness of the classifier k is multiplied by $1/\theta_{exp}$.

Next, the parents are crossed and mutated with probabilities P_χ and P_μ respectively. The consequent of the rule and the parameters of the offspring are initialized as in covering.

The crossover operator crosses the rule antecedent by two points selected randomly. This could result in classifiers containing variables with no linguistic terms, which would indicate that the rule is not applicable. If this is the case, we copy a linguistic term from one of the parents. Crossover does not modify the consequent of the rule.

The mutation operator randomly decides if a variable has to be mutated. If a variable is selected, three types of mutation can be applied: *expansion*, *contraction*, or *shift*. Expansion chooses a linguistic term not represented in the corresponding variable and adds it to this variable; thus, it can only be applied on variables that do not have all the linguistic terms. Contraction is the opposite process: is removes a linguistic term from one variable; so, it can only be applied if the variable has more than one linguistic term. Shift changes a linguistic term for its immediately inferior or superior.

Finally, the new offspring are inserted into the population. First, each offspring is checked for subsumption with its parents. If either of the parents is enough experienced ($exp > \theta_{sub}$), highly accurate ($F > F_0$), and more general than the offspring, its numerosity is incremented. If the offspring cannot be subsumed by any of its parents, the same process is used to find a subsumer in [C]. If there is no subsumer, the offspring is inserted in the population. A classifier is deleted if the population is full; in this case, each classifier is given a deletion probability of

$$P_{del}^k = \frac{d^k}{\sum_{\forall j\in[P]} d_j} \tag{9}$$

where

$$d_k = \begin{cases} cs^k \cdot num^k \cdot \frac{\overline{F}}{F^k} & \text{if } exp^k > \theta_{del} \text{ and } (F^k)^\nu < \delta\overline{F} \\ cs^k \cdot num^k & \text{otherwise} \end{cases} \tag{10}$$

where $\overline{F}$ is the average fitness of classifiers in [P], and δ and θ_{del} are two parameters set by the user ($0 < \delta < 1$ and $\theta_{del} > 0$). Thus, this method gives higher deletion probabilities to numerous classifiers that belong to large correct sets; moreover, it also penalizes experienced classifiers with low fitness.

2.5 Fuzzy-UCS in Test Mode

Fuzzy-UCS aims at obtaining a highly accurate rule set of minimum size. To obtain high accuracy, we need to define an effective reasoning method that infers the output class from the final population. To obtain a reduced rule set, some reduction strategies may be applied to remove classifiers that are not important for the reasoning. In the following, we discuss two reasoning approaches which lead to two different rule set reduction mechanisms.

Class Inference. In test mode, given a new unlabeled instance e, several rules can match (with different degrees) this instance, each having a certain fitness F^k. Thus, a reasoning process needs to be applied to decide the output. Here, we propose two fuzzy-inference approaches:

1. *Weighted average inference.* In this approach, all rules vote to infer the output. Each rule k emits a vote v_k for class j it advocates, where

$$v_k = F^k \cdot \mu_{A^k}(e) \tag{11}$$

   The votes for each class j are added:

$$\forall j : vote_j = \sum_{k|c^k=j}^{N} v_k \tag{12}$$

   and the most-voted class is returned as the output. Note that this strategy is analogous to the inference scheme of UCS.
2. *Action winner inference.* This approach proposes to select the rule k that maximizes $F^k \cdot \mu_{A^k}(e)$, and choose the class of the rule as output [20]. Thus, the knowledge of overlapping rules is not considered in this inference scheme.

Ruleset Reduction. At the end of the learning, the population is reduced to obtain a minimum set of rules with the same training accuracy as the original rule set. The reduction strategy depends on the type of inference used.

1. *Reduction based on weighted average.* Under the weighted average inference, the final population is reduced by removing all the rules that a) are not experienced enough ($exp < \theta_{exploit}$) or b) have zero or negative fitness.
2. *Reduction based on action winner.* If action winner inference is used, it is only necessary to maintain the rules that maximize the prediction vote for each training example. Thus, after training, this reduction scheme infers the output for each training example. The rule that maximizes the vote v_j for each example is copied to the final population.

Next section compares Fuzzy-UCS with the two inference and reduction mechanisms to UCS and other machine learning techniques. For notation, these schemes will be addressed as: weighted average inference (wavg), and action winner inference (awin).

3 Experimentation

In this section, we analyze the competence of Fuzzy-UCS in classification tasks. The main goal of data classification is to obtain highly accurate models that provide comprehensible explanations for human experts. For this purpose, we compare the performance and rule set interpretability of Fuzzy-UCS with respect to UCS and three other well-known machine learning techniques: the decision tree C4.5 [25], the support vector machine SMO [24], and the boosting algorithm based on a fuzzy representation Fuzzy LogitBoost [22]. In the following, we first detail the methodology followed in the comparison and then present the results obtained.

3.1 Methodology

Experimentation Problems. We selected a collection of twenty real-world problems. This collection includes problems with different characteristics (see Table 1) which may pose particular challenges to the different learning techniques. All these problems were obtained from the UCI repository [5], except for *tao*, which was chosen from a local repository [4].

Machine Learning Techniques Included in the Comparison. We compared Fuzzy-UCS with the two inference mechanisms to four machine learning techniques: UCS, C4.5, SMO, and Fuzzy LogitBoost. UCS [3] is the learning classifier system from which Fuzzy-UCS was derived; so, we want to analyze whether the fuzzy representation permits to achieve similar performance and improves the interpretability of the rule sets evolved by UCS. C4.5 [25] is a decision tree that enhances ID3 by introducing methods to deal with continuous variables and missing values. C4.5 is one of the most used learners in the realm of pattern classification, since it usually results in accurate tree-based models which are quite interpretable by human experts. SMO [24] is a widely-used implementation of *support vector machines* [31]; SMO implements the *Sequential Minimization Algorithm* to solve the dual problem. We included both C4.5 and SMO into the analysis to compare Fuzzy-UCS to two top-notch machine learning techniques. Fuzzy LogitBoost [22] is a boosting algorithm that iteratively invokes a genetic algorithm to extract simple fuzzy rules that are combined to decide the output of new examples. We selected Fuzzy LogitBoost to be in the comparison since it is a modern method which relies on statistics theory, and so, it is a good representative of fuzzy learners. Table 2 summarizes the main characteristics of the learners.

C4.5 and SMO were run using WEKA [35]. For Fuzzy LogitBoost, we used KEEL [2]. For UCS, we ran our own code. All the open source methods were

Table 1. Properties of the data sets. The columns describe: the identifier of the data set (Id.), the name of the data set (dataset), the number of instances (#Inst), the total number of features (#Fea), the number of real features (#Re), the number of integer features (#In), the number of nominal features (#No), the number of classes (#Cl), and the proportion of instances with missing values (%Miss)

Id.	dataset	#Inst	#Fea	#Re	#In	#No	#Cl	%Miss
ann	Annealing	898	38	6	0	32	5	0
aut	Automobile	205	25	15	0	10	6	22.4
bal	Balance	625	4	4	0	0	3	0
bpa	Bupa	345	6	6	0	0	2	0
cmc	Contraceptive method choice	1473	9	2	0	7	3	0
col	Horse colic	368	22	7	0	15	2	98.1
gls	Glass	214	9	9	0	0	6	0
h-c	Heart-c	303	13	6	0	7	2	2.3
h-s	Heart-s	270	13	13	0	0	2	0
irs	Iris	150	4	4	0	0	3	0
pim	Pima	768	8	8	0	0	2	0
son	Sonar	208	60	60	0	0	2	0
tao	Tao	1888	2	2	0	0	2	0
thy	Thyroid	215	5	5	0	0	3	0
veh	Vehicle	846	18	18	0	0	4	0
wbcd	Wisc. breast-cancer	699	9	0	9	0	2	2.3
wdbc	Wisc. diagnose breast-cancer	569	30	30	0	0	2	0
wne	Wine	178	13	13	0	0	3	0
wpbc	Wisc. prognostic breast-cancer	198	33	33	0	0	2	2
zoo	Zoo	101	17	0	1	16	7	0

Table 2. Summary of the main characteristics of the learners included in the comparison: C4.5, SMO, UCS, and Fuzzy LogitBoost (Bst)

	Paradigm	Knowledge Rep. and Inference Method
C4.5	Decision-tree induction	Decision-tree. *Inference*: class given by the corresponding leaf.
SMO	Neural networks (support vector machines)	Weights of the support vector machines. *Inference:* The class is determined by the decision function represented by the SVM.
UCS	Michigan-style GBML	Population of intervalar rules with a fitness value. *Inference:* The output is the most voted class among the matching classifiers.
Bst	Statistical Learning Theory and GBML	Population of linguistic fuzzy rules with a strength per class. *Inference:* The output is the most voted class among the matching classifiers.

configured with the parameters values recommended by default, with the following exceptions. The model for SMO was selected by choosing the kernel that maximized the global accuracy with respect to the other learners. That is, we ran SMO with polynomial kernels of order 1, 3, 5, and 10, and with a Gaussian kernel. Then, we ranked the results obtained with the three polynomial kernels and chose the model that maximized the average rank: SMO with polynomial kernels of order 3. We followed a similar strategy to select the maximum population size of Fuzzy LogitBoost, for which we did not find a set of recommended values in the literature. We tried population sizes of N={8,25,50,100} for all the data sets, and provide the results of N=50 since they permitted to achieve, in general, higher performance ratios than N=8 and N=25, and did not significantly differ from the results obtained with N=100.

UCS was configured with the following parameters (see [3,21] for notation details): $numIter$=100,000, N=6400, $acc_0 = 0.99$, ν=10, $\{\theta_{GA}, \theta_{del}, \theta_{sub}\}$=50, P_χ=0.8, P_μ=0.04, δ=0.1, $P_\#$=0.6. These are typical configuration parameters for UCS. No extra experimentation was made to improve the performance results, so that the reported results could be even improved with a further tuning of configuration parameters. Similar configuration parameter values were chosen for Fuzzy-UCS, that is: $numIter$=100,000, N=6400, $F_0 = 0.99$, $\nu = 10$, $\{\theta_{GA}, \theta_{del}, \theta_{sub}\} = 50$, $\theta_{exploit} = 10$, $P_\chi = 0.8$, $P_\mu = 0.04$, δ=0.1, and $P_\# = 0.6$. Both fuzzy based learners, i.e., Fuzzy-UCS and Fuzzy LogitBoost, used five linguistic terms per variable defined by triangular-shaped membership functions.

Comparison Metrics. The data models built by each learner were evaluated in terms of performance and interpretability. We measured the performance of the method with the test accuracy, i.e., the proportion of correct classification on previously unseen examples. To obtain reliable estimates of test accuracies, we ran the experiments on a ten-fold cross validation [29]. For the stochastic methods, the results provided correspond to the average of ten runs with different random seeds.

The comparison of the interpretability of the models is more complicated since the methods included in the comparison use different knowledge representations. Whilst UCS, Fuzzy-UCS, and Fuzzy LogitBoost use a rule-based representation, C4.5 builds a decision tree, and SMO represents the data model with the weights (ranging from 0 to 1) of a support vector machine. For this reason, we gathered some indicators of the model sizes, i.e., number of rules for the rule-based systems, number of leaves for the trees, and number of weights for the support vector machines. In the next section, we qualitatively discuss the advantages and disadvantages of the different representations.

Statistical Analysis. We followed the methodology pointed in [12] to statistically analyze the differences in performance among learners. As suggested by the author, we avoided to use any parametric statistical test since they require that the data satisfy several strong conditions. Instead, all the statistical analysis is based on non-parametric tests.

We first applied a multi-comparison statistical procedure to test whether all the learning algorithms performed the same on average. Specifically, we used the Friedman test [15,16], the non-parametric equivalent to the analysis of variance test ANOVA [14]. As indicated in [12], if the Friedman test rejects the hypothesis that all the learners perform the same on average, several post-hoc tests can be used to detect significant differences between groups of learners. As our aim was to compare Fuzzy-UCS to the other learners (that is, one control learner against the others) we used the non-parametric Bonferroni-Dunn test [13]. The Bonferroni-Dunn test defines that one learner performs significantly differently than a control learner if the corresponding average rank differs by, at least, a critical difference CD, which is computed as

$$CD = q_\alpha \sqrt{\frac{n_\ell(n_\ell + 1)}{6n_{ds}}} \tag{13}$$

where n_ℓ is the number of learners in the comparison, n_{ds} is the number of data sets, and q_α is the critical value based on the Studentized range statistic [28]. We illustrate the results of this test by showing the group of learners that perform equivalently to the control learner.

The Bonferroni-Dunn test is said to be conservative, especially as the number of learners increases or the number of data sets decreases, so that it may not detect significant differences although they actually exist. Nonetheless, we use this test in the first stage of our analysis since it permits to detect groups of learners that truly perform differently from other learners. We latter apply pairwise comparisons to detect further significant differences between learners that belong to the same group, assuming the risk of increasing the error of rejecting null hypotheses when they are actually true. We used the non-parametric Wilcoxon signed-ranks test [33] for pairwise comparisons, and provide the approximative p-values computed as indicated in [28].

3.2 Results

Comparison of the Performance. Table 3 shows the average performance obtained with the six learners on the twenty real-world problems. The last two rows of the table supply the average rank and the position of each learner in the ranking. The procedure to calculate the ranks is the following. For each data set, we ranked the learning algorithms according to their performance; the method with the highest accuracy was the first in the ranking, while the learner with the poorest results holds the last position of the ranking. If a group of learners had the same accuracy, we assigned the average rank of the group to each of those learners.

The experimental results evidence the competitiveness of Fuzzy-UCS with respect to the other machine learning techniques, especially when all the rules are used in the inference process. That is, Fuzzy-UCS with weighted average inference is the third method of the ranking; it is only outperformed by SMO with polynomial kernels and UCS, the method from whom Fuzzy-UCS was inspired.

Table 3. Comparison of the accuracy rate of Fuzzy-UCS with weighted average (wavg) and action winner (awin) inference schemes to UCS, C4.5, SMO, and Fuzzy LogitBoost (LBoost). The two last rows of the table report the average rank of each learning algorithm (Rnk), and its position in the rank (Pos).

	UCS	**C4.5**	**SMO**	**LBoost**	**Fuzzy-UCS**	
					wavg	**awin**
ann	99.05	98.90	99.34	76.20	98.85	97.39
aut	77.41	80.94	78.09	32.63	74.42	67.42
bal	77.32	77.42	91.20	88.30	88.65	84.40
bpa	67.59	62.31	59.97	64.46	59.82	59.42
cmc	50.27	52.62	48.75	51.10	51.72	49.67
col	96.26	85.32	75.59	63.06	85.01	82.46
gls	70.04	66.15	66.15	68.18	60.65	57.21
h-c	79.72	78.45	78.59	62.09	84.39	82.62
h-s	74.63	79.26	78.89	59.33	81.33	80.78
irs	95.40	94.00	92.67	95.33	95.67	95.47
pim	74.61	74.23	76.70	71.84	74.88	74.11
son	76.49	71.07	85.52	53.38	80.78	73.71
tao	87.00	95.92	84.22	91.73	81.71	83.02
thy	95.13	94.91	88.91	97.08	88.18	89.49
veh	71.40	71.14	83.30	37.25	67.68	65.35
wbcd	96.28	94.99	96.42	94.12	96.01	95.73
wdbc	95.96	94.40	97.58	62.74	95.20	94.61
wne	96.13	93.89	97.75	85.02	94.12	94.86
wpbc	69.40	71.61	81.25	76.35	76.06	76.05
zoo	96.78	92.81	97.83	41.89	96.50	94.78
Rnk	***2.80***	***3.63***	***2.68***	***4.55***	***3.20***	***4.15***
Pos	***2***	***4***	***1***	***6***	***3***	***5***

These results indicate that, even though the granularity limitations that the linguistic fuzzy representation may impose compared to an interval-based representation, Fuzzy-UCS presents a similar performance than UCS. Besides, note that Fuzzy-UCS with weighted average achieves a better average rank than C4.5, one of the most used machine learning techniques. Similarly, Fuzzy-UCS with weighted average also outperforms Fuzzy LogitBoost, indicating that Fuzzy-UCS can evolve more accurate fuzzy rules than Fuzzy LogitBoost.

The average rank worsens when only the information of the best rules is used for inferring the class of new input instances. That is, Fuzzy-UCS with action winner inference holds the fifth position of the ranking. These results confirm the advantages of combining the information of all the fuzzy rules in the inference process [10]. However, the next section shows that Fuzzy-UCS with action winner results in much more reduced rule sets than Fuzzy-UCS with weighted average, which opens an accuracy-interpretability trade-off. Moreover, Fuzzy-UCS with action winner also outperforms the other fuzzy learner, Fuzzy LogitBoost.

We analyzed statistically the results to identify significant differences among learners. Friedman test for multiple comparisons rejected the null hypothesis

Table 4. Pairwise comparison of the performance of the six learners by means of a Wilcoxon signed-ranks test. The above diagonal contains the approximate p-values. The below diagonal shows a symbol $\oplus$ / $\ominus$ if the method in the row significantly outperforms/degrades the method in the column at a significance level of .05 and +/=/− if there is no significant difference and performs better/equal/worse.

	UCS	**C4.5**	**SMO**	**LogitBoost**	**wavg**	**awin**
UCS		.2043	.6542	.0072	.4330	.0674
C4.5	-		.4209	.0111	.7938	.3905
SMO	+	+		.0072	.1672	.0400
LogitBoost	$\ominus$	$\ominus$	$\ominus$		.0100	.0276
wavg	-	+	-	$\oplus$		.0032
awin	-	=	$\ominus$	$\oplus$	$\ominus$	

that the six learners performed the same on average with $p = 0.006$. To evaluate the differences between Fuzzy-UCS with the two inference schemes and the other learners, we applied the Bonferroni-Dunn test at $\alpha = 0.10$. Figure 3 places each method according to its average rank. Furthermore, it connects with a line the methods that perform equivalently to (1) Fuzzy-UCS with weighted average inference, and (2) Fuzzy-UCS with action winner inference. The statistical analysis indicates that Fuzzy-UCS with weighted average significantly outperforms Fuzzy LogitBoost. Besides, Fuzzy-UCS with action winner performs equivalently to all the methods except for SMO.

As the Bonferroni test is said to be conservative [28], so that it may not detect all the significant differences, we complemented the statistical study by compar-

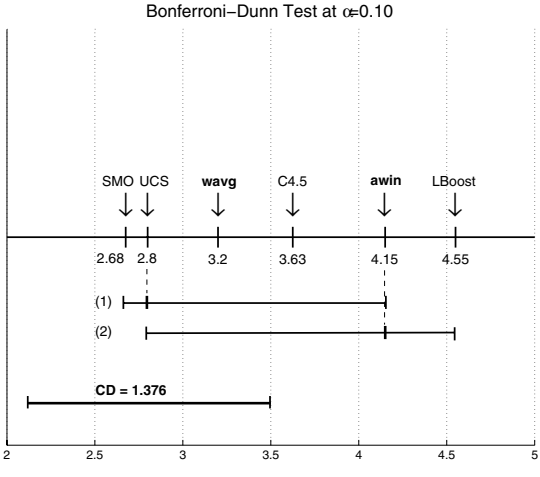


Fig. 3. Comparisons of one leaner against the others according to a Bonferroni-Dunn test at a significance level of 0.10. All the learners are compared to two different control groups: (1) Fuzzy-UCS with weighted average and (2) Fuzzy-UCS with action winner. The methods connected are those that perform equivalently to the control learner.

ing each pair of learners. Note that with this approach we are increasing the risk of rejecting hypothesis that are actually true [12]; nonetheless, we do not base the statistical analysis on pairwise comparisons, but use them to complement the conclusions drawn above. The above diagonal of Table 4 shows the approximate p-values of the pairwise comparison according to a Wilcoxon signed-ranks test. The below diagonal indicates with the symbols $\oplus$ and $\ominus$ that the method in the row significantly outperforms/degrades the performance obtained with the method in the column. Similarly, the symbols + and - are used to denote a non-significant improvement/degradation. The symbol = indicates that each method outperforms and degrades the other the same number of times.

The pairwise analysis confirms the conclusions pointed by the Bonferroni-Dunn test, and it detects additional significant differences among learners. The test supports that Fuzzy-UCS with weighted average inference does not degrade the performance of any other learner and significantly improves Fuzzy LogitBoost and Fuzzy-UCS with action winner inference. On the other hand, it indicates that Fuzzy-UCS with action winner inference outperforms Fuzzy LogitBoost; besides, it degrades the results obtained by SMO and Fuzzy-UCS with weighted average. Finally, note that Fuzzy-UCS with action winner inference is statistically equivalent to C4.5; that is, it outperforms and degrades the results of C4.5 in the same number of the data sets.

Comparison of the Interpretability. The analysis made in the previous section highlights that Fuzzy-UCS is able to evolve highly accurate models, especially when the weighted average inference is used. Herein, we analyze the readability of the models evolved by Fuzzy-UCS, and qualitatively compare them to the models built by the other learners. To evaluate the readability of the final model, we consider two aspects: (i) the intrinsic interpretability of the model, i.e., if the knowledge representation can be easily understood by human experts; and (ii) the size of the model.

First, we compare the type of rules evolved by Fuzzy-UCS to the other knowledge representations. Figure 4 depicts partial examples of the models created by each learner for the two-dimensional problem *tao*. Fuzzy-UCS evolves a set of linguistic fuzzy rules with a fitness value (see Fig. 4(a)). UCS creates a population of interval-based rules; the lower and upper bounds of the intervals can take any valid value for the attribute (see Fig. 4(b)). C4.5 builds a decision tree where each node establishes a decision over one variable (see Fig. 4(c)). SMO creates $\binom{n}{2}$ support vector machines (where n is the number of classes of the classification problem), each one consisting of a set of continuous weights ranging from 0 to 1 (see Fig. 4(d)). Fuzzy LogitBoost evolves a set of linguistic fuzzy rules (see Fig. 4(e)). These rules are similar to the ones of Fuzzy-UCS, with the following two exceptions. In Fuzzy-UCS, rule's variables can take a disjunction of linguistic terms; differently, in Fuzzy LogitBoost the variables are represented by a single linguistic term. Furthermore, the rules of Fuzzy-UCS predict a single class with a certain fitness, whilst the rules of Fuzzy LogitBoost maintain a weight for each class, and use these weights in the inference process.

if x is XL **then** blue **with** F=1.00
if x is XS **then** red **with** F=1.00
if x is {XS or S} **and** y is {XS or S} **then** red **with** F=0.87
...

(a) Fuzzy-UCS

if x is [-6.00, -0.81] **and** y is [-6.00, 0.40] **then** red **with** acc= 1.00
if x is [2.84, 6.00] **and** y is [-5.26, 4.91] **then** blue **with** acc =1.00
if x is [-6.00, -0.87] **and** y is [-6.00, 0.74] **then** red **with** acc =1.00
...

(b) UCS

```
x <= -2.75
|  x <= -3.25: red (308.0)
|  x > -3.25
|  |    y <= 1.75: red (55.0)
|  |    y > 1.75
|  |  |  x <= -3: red (11.0/1.0)
|  |  |  x > -3
|  |  |  |     y <= 4.25: blue (6.0)
|  |  |  |     y > 4.25: red (4.0)
                ...
```

(c) C4.5

```
-    1.000    * <0.229 0.875 > * X]
-    0.298    * <0.708 0.437 > * X]
                ...
```

(d) SMO

if x is L **and** y is L **then** blue **with** -5.42 **and** red **with** 0.0
if x is M **and** y is XS **then** blue **with** 2.21 **and** red **with** 0.0
if x is M **and** y is XL **then** blue **with** -2.25 **and** red **with** 0.0
...

(e) Fuzzy LogitBoost

Fig. 4. Examples of part of the models evolved by (a) Fuzzy-UCS, (b) UCS, (c) C4.5, (d) SMO, and (e) Fuzzy LogitBoost for the two-dimensional tao problem

This first analysis permits us to draw several observations concerning the readability of the representation itself. In the following, we distinguish between three types of learners depending on the inherent interpretability of their knowledge representation:

1. SMO presents the least readable knowledge representation, since the weights of the support vector machines provide poor information for human experts. Thus, SMO is not suitable when the interpretability of the models is important.
2. The trees created by C4.5 and the rules evolved by UCS can be easily interpreted by human experts given models of moderate size. However, note that both the decision nodes and the rules codify the decision over input variables with continuous numbers. This permits to precisely define the decision boundaries, but may hinder the interpretability of the models, especially when the number of nodes or rules increases.
3. Fuzzy-UCS and Fuzzy LogitBoost use a linguistic representation. As the input variables are represented by linguistic terms, individual rules can be read more easily than interval-based rules and decision trees. Nonetheless, linguistic terms imply a discretization of the feature space, which may result in a loss of precision to define the class boundaries.

Next, we analyze the sizes of the models evolved by the different learners. For this purpose, we used the following metrics. For rule-based systems, i.e., Fuzzy-UCS, UCS, and Fuzzy LogitBoost, we counted the number of rules evolved. For the tree-based system, i.e., C4.5, we reckoned the number of leaves, since each path from the root of the tree to one leaf can be regarded as a complex rule. Note that these measures are not directly comparable due to the differences in the knowledge representations. However, we use these metrics to qualitatively analyze the differences among learners.

Table 5 depicts the model sizes of the rule-based and tree-based systems. These results show that:

1. Fuzzy-UCS with weighted average inference evolves smaller rules sets than UCS in most of the problems; besides, the fuzzy rule representation is more readable. However, the still large amount of rules may hinder the interpretability of the final rule set.
2. Fuzzy-UCS with action winner inference creates a much smaller rule sets than UCS and Fuzzy-UCS with weighted average. These big differences permit to significantly improve the readability of the final rule set, at a cost of slightly decreasing the test performance as shown in the previous section.
3. Fuzzy LogitBoost creates rule sets of moderate size. Actually, the size of the rule sets is determined by a configuration parameter. In our experiments,

Table 5. Average sizes of the models built by UCS, C4.5, Fuzzy LogitBoost (LBoost), and Fuzzy-UCS with weighted average and action winner inference

	UCS	**C4.5**	**LBoost**	**Fuzzy-UCS**	
				wavg	**awin**
ann	4410	38	50	2769	75
aut	4064	44	50	3872	114
bal	1712	45	50	1212	114
bpa	2603	25	50	1440	73
cmc	3175	162	50	1881	430
col	3446	5	50	4135	154
gls	3013	24	50	2799	62
h-c	2893	29	50	3574	113
h-s	3499	17	50	3415	117
irs	634	5	50	480	18
pim	3225	19	50	2841	192
son	5999	14	50	3042	178
tao	609	36	50	111	19
thy	1283	8	50	1283	37
veh	4601	69	50	3732	332
wbcd	1799	12	50	3130	138
wdbc	5079	11	50	5412	276
wne	3413	5	50	3686	95
wpbc	5078	12	50	3772	156
zoo	1244	11	50	773	16

we set the maximum population size to 50 since it maximized the average performance rank of the algorithm. Smaller population sizes could be set for particular problems without loss of performance. However, we are interested in robust methods that do not present a high dependency on their configuration parameters. For this reason, we used the same parameters in all the runs, and did not search for the best configuration for each specific data set (in fact, we did not follow this approach for any learner).

The rule sets evolved by Fuzzy LogitBoost are slightly smaller than those evolved by Fuzzy UCS. However, note that the rules created by LogitBoost maintain a weight per each class (see Fig. 4(e)), whilst Fuzzy-UCS's rules only maintain a single fitness value (see Fig. 4(a)). Furthermore, Fuzzy-UCS evolves different number of rules depending on the intrinsic complexities of the domain, while Fuzzy LogitBoost needs to know beforehand the number of rules to be created.

4. C4.5 evolves trees of moderate size. The number of leaves is sligthly inferior than the number of rules of Fuzzy-UCS with action winner.

The whole analysis provided in this section showed the competitivity of Fuzzy-UCS with respect to the other machine learning techniques. In terms of performance, Fuzzy-UCS with weighted average inference was one of the best methods in the ranking. In terms of interpretability, we showed that Fuzzy-UCS with action winner inference resulted in rule sets much more reduced than those evolved by UCS—and Michigan-style Learning Classifier Systems in general—at the cost of slightly decreasing the test performance. This evidenced an accuracy-interpretability trade-off: the more information is used in the inference, the more accurate the prediction of the class of test examples is. The next section emphasizes the advantages of the online learning architecture and the many opportunities that Fuzzy-UCS offers, which are left for further work.

4 Conclusions and Further Work

In this paper, we proposed Fuzzy-UCS, a Michigan-style Learning Fuzzy-Classifier System for supervised learning. Fuzzy-UCS was derived from UCS with the aim of achieving more understandable models. For this reason, we replaced the interval-based rule representation of UCS with a linguistic fuzzy representation, and designed two inference mechanisms that offer different levels of rule set reduction.

We tested the system on a large collection of real-world problems, and compared the performance and interpretability of the models evolved by Fuzzy-UCS—with the two types of inference—to the models created by UCS, and three other machine learning techniques: the decision tree C4.5, the support vector machine SMO, and the statistic classifier based on a fuzzy representation Fuzzy LogitBoost. The results highlighted the competence of Fuzzy-UCS, which was able to evolve highly accurate and interpretable rule sets. Moreover, it showed an accuracy-interpretability trade-off. As expected, the more information is used for the inference process, the highest the accuracy of the models is; however, the

readability of these models also worsens, since there are more rules in the final populations. In further work, deeper research will be conducted to understand more carefully the actual limitations imposed by the linguistic representation and the inference scheme used, and fuzzy approximative representations will be studied.

Besides its good behavior, Fuzzy-UCS presents two differential traits with respect to the other learners in the comparison, which lead to several opportunities concerning different challenges of data mining. Fuzzy-UCS, and in general Michigan-style LCSs, evolve the rule set incrementally. This differs from many learners which go several times through all the data set to create the classification model. Due to the online architecture, Fuzzy-UCS could be applied to two topics of increasing interest: (i) learning from large data sets [8], and, (ii) learning from data streams [1]. Furthermore, the use of fuzzy logic allows Fuzzy-UCS to be adapted for learning from vague and uncertain data, which is frequent in real-world classification problems [26,27]. The research on these three topics is left as further work.

Acknowledgements

The authors thank the support of *Enginyeria i Arquitectura La Salle*, Ramon Llull University, as well as the support of *Ministerio de Educación y Ciencia* under projects TIN2005-08386-C05-01 and TIN2005-08386-C05-04, and *Generalitat de Catalunya* under grants 2005FI-00252 and 2005SGR-00302.

References

1. Aggarwal, C.: Data Streams: Models and Algorithms. Springer, Heidelberg (2007)
2. Alcalá-Fdez, J., del Jesus, M.J., Garrell, J.M., Herrera, F., Herbás, C., Sánchez, L.: Proyecto KEEL: Desarrollo de una herramienta para el análisis e implementación de algoritmos de extracción de conocimiento evolutivos. In: Aguilar, J.S., Giráldez, R., Riquelme, J.C. (eds.) Tendencias de la Minería de Datos en España, Red Española de Minería de Datos y Aprendizage, pp. 413–424 (2004)
3. Bernadó-Mansilla, E., Garrell, J.M.: Accuracy-Based Learning Classifier Systems: Models, Analysis and Applications to Classification Tasks. Evolutionary Computation 11(3), 209–238 (2003)
4. Bernadó-Mansilla, E., Llorà, X., Garrell, J.M.: XCS and GALE: a comparative study of two learning classifier systems on data mining. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 2001. LNCS (LNAI), vol. 2321, pp. 115–132. Springer, Heidelberg (2002)
5. Blake, C.L., Merz, C.J.: UCI Repository of ML Databases: University of California (1998), `http://www.ics.uc.edu/~mlearn/MLRepository.html`
6. Bonarini, A.: Evolutionary Learning of Fuzzy rules: competition and cooperation. In: Pedrycz, W. (ed.) Fuzzy Modelling: Paradigms and Practice, pp. 265–284. Kluwer Academic Press, Norwell (1996)
7. Bonarini, A., Trianni, V.: Learning fuzzy classifier systems for multi-agent coordination. Information Sciences: an International Journal 136(1-4), 215–239 (2001)

8. Bottou, L., Le Cun, Y.: On-line learning for very large data sets: Research Articles. Applied Stochastic Models in Business and Industry 21(2), 137–151 (2005)
9. Casillas, J., Carse, B., Bull, L.: Fuzzy-XCS: a michigan genetic fuzzy system. IEEE Transactions on Fuzzy Systems 15(4), 536–550 (2007)
10. Cordón, O., del Jesús, M.J., Herrera, F.: Analyzing the Reasoning Mechanisms in Fuzzy Rule Based Classification Systems. Mathware & Soft Computing 5, 321–332 (1998)
11. Cordón, O., Herrera, F., Hoffmann, F., Magdalena, L.: Genetic Fuzzy Systems: Evolutionary Tuning and Learning of Fuzzy Knowledge Bases. In: Advances in Fuzzy Systems—Applications and Theory, vol. 19, World Scientific, Singapore (2001)
12. Demšar, J.: Statistical Comparisons of Classifiers over Multiple Data Sets. Journal of Machine Learning Research 7, 1–30 (2006)
13. Dunn, O.J.: Multiple Comparisons among Means. Journal of the American Statistical Association 56, 52–64 (1961)
14. Fisher, R.A.: Statistical Methods and Scientific Inference, 2nd edn. Hafner Publishing Co, New York (1959)
15. Friedman, M.: The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance. Journal of the American Statistical Association 32, 675–701 (1937)
16. Friedman, M.: A Comparison of Alternative Tests of Significance for the Problem of m Rankings. Annals of Mathematical Statistics 11, 86–92 (1940)
17. Furuhashi, T., Nakaoka, K., Uchikawa, Y.: Suppression of excess fuzziness using multiple fuzzy classifier systems. In: Proceedings of the 3th IEEE International Conference on Fuzzy Systems, pp. 411–414. Morgan Kaufmann, San Francisco (1994)
18. Goldberg, D.E.: Genetic Algorithms in Search, Optimization & Machine Learning, 1st edn. Addison Wesley, Reading (1989)
19. Holland, J.H.: Adaptation in Natural and Artificial Systems. The University of Michigan Press (1975)
20. Ishibuchi, H., Nakashima, T., Murata, T.: Performance evaluation of fuzzy classifier systems for multidimensional pattern classification problems. IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics 29(5), 601–618 (1999)
21. Orriols-Puig, A., Bernadó-Mansilla, E.: A Further Look at UCS Classifier System. In: GECCO 2006: Genetic and Evolutionary Computation Conference workshop program, Seattle, W.A., USA, 08–12 July 2006. ACM Press, New York (2006)
22. Otero, J., Sánchez, L.: Induction of descriptive fuzzy classifiers with the logitboost algorithm. Soft Computing 10(9), 825–835 (2006)
23. Parodi, A., Bonelli, P.: A new approach to fuzzy classifier systems. In: Proceedings of the 5th International Conference on Genetic Algorithms, pp. 223–230. Morgan Kaufmann, San Francisco (1993)
24. Platt, J.: Fast Training of Support Vector Machines using Sequential Minimal Opt. In: Advances in Kernel Methods - Support Vector Lear. MIT Press, Cambridge (1998)
25. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, San Francisco (1995)
26. Sánchez, L., Couso, I.: Advocating the use of Imprecisely Observed Data in Genetic Fuzzy Systems. IEEE Transactions on Fuzzy Systems (2007)

27. Sánchez, L., Couso, I., Casillas, J.: Modeling vague data with genetic fuzzy systems under a combination of crisp and imprecise criteria. In: Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Multicriteria Decision Making, pp. 346–353 (2007)
28. Sheskin, D.J.: Handbook of Parametric and Nonparametric Statistical Procedures. Chapman & Hall, Boca Raton (2000)
29. Dietterich, T.G.: Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms. Neural Comp. 10(7), 1895–1924 (1998)
30. Valenzuela-Radón, M.: The Fuzzy Classifier System: A Classifier System for Continuously Varying Variables. In: 4th ICGA, pp. 346–353. Morgan Kaufmann, San Francisco (1991)
31. Vapnik, V.: The Nature of Statistical Learning Theory. Springer, Heidelberg (1995)
32. Velasco, J.: Genetic-based on-line learning for fuzzy process control. International Journal of Intelligent Systems 13, 891–903 (1998)
33. Wilcoxon, F.: Individual comparisons by ranking methods. Biometrics 1, 80–83 (1945)
34. Wilson, S.W.: Get Real! XCS with Continuous-Valued Inputs. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 1999. LNCS (LNAI), vol. 1813, pp. 209–219. Springer, Heidelberg (2000)
35. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques, 2nd edn. Morgan Kaufmann, San Francisco (2005)

A Principled Foundation for LCS

Jan Drugowitsch and Alwyn M. Barry

Department of Computer Science
University of Bath
Bath, BA2 7AY, UK
J.Drugowitsch@bath.ac.uk, A.M.Barry@bath.ac.uk

Abstract. In this paper we promote a new methodology for designing LCS that is based on first identifying their underlying model and then using standard machine learning methods to train this model. This leads to a clear identification of the LCS model and makes explicit the assumptions made about the data, as well as promises advances in the theoretical understanding of LCS through transferring the understanding of the applied machine learning methods to LCS. Additionally, it allows us, for the first time, to give a formal and general, that is, representation-independent, definition of the optimal set of classifiers that LCS aim at finding. To demonstrate the feasibility of the proposed methodology we design a Bayesian LCS model by borrowing concepts from the related Mixtures-of-Experts model. The quality of a set of classifiers and consequently also the optimal set of classifiers is defined by the application of Bayesian model selection, which turns finding this set into a principled optimisation task. Using a simple Pittsburgh-style LCS, a set of preliminary experiments demonstrate the feasibility of this approach.

1 Introduction

In this work we promote a model-based design approach for LCS that allows us to define formally and from first principles which set of classifiers LCS aim at learning, thus tackling the core question of LCS. The motivation behind acquiring the model-based approach is that, in essence, any machine learning method is based on some form of (sometimes implicit) model that determines its training and makes explicit the assumptions that are made about the data that it models. Thus, if the model underlying LCS is made explicit, we can use standard machine learning methods for its training and analysis, in addition to making explicit the assumptions about the data. The latter is important as it tells us how LCS differ from other approaches in machine learning, and in particular for which kind of data they might feature superior performance.

Up to now, the design of LCS is mostly handled in an ad-hoc way with an intuitive understanding of how the optimal set of classifiers might look like. This leads to algorithmic descriptions of their implementations and subsequent attempts of analysing their performance and inner working, such as, for example, the extensive analysis of XCS [1] by Butz et al. [2,3,4]. We propose a different approach: firstly, one should seek to make explicit the underlying model that

J. Bacardit et al. (Eds.): IWLCS 2006/2007, LNAI 4998, pp. 77–95, 2008.

LCS uses for a set of classifiers to represent the data. Having such a model, the definition of the optimal set of classifiers, that is, the set that LCS training should seek, becomes simple, yet powerful: the optimal set of classifiers is the one that models the data best.

Such a definition of the optimal set of classifiers, and with it the whole methodology, seemingly only applies to regression and classification tasks, and only if all data is known before the LCS is trained (that is, for batch training). However, we claim that the approach can be used to derive both batch and incremental training methods, based on interpreting incremental methods as approximations to solutions that can equally be found by a batch learning approach[1]. Also, the approach can be extended to sequential decision tasks by the application of reinforcement learning, and using LCS to approximate the value function, which is an incremental non-stationary univariate regression task [5].

Naturally, there is no single model that describes all possible LCS variants. Similarly, there is not only a single machine learning method that can be applied to tell us which model represent the data best. Thus, we want to emphasise that the regression model that forms the core of this paper — which is one that closely resembles XCS(F) [1,6] — and the methods that we have applied to define and find the best set of classifiers are only meant to illustrate that the methodology we promote does indeed lead to feasible LCS implementations. The presented model can be easily reformulated to a classification model that leads to a different LCS specialised on classification tasks. Alternatively, it might be reformulated to make different assumptions about the data, which leads to a different LCS variants.

Due to space constraints we mainly focus on formulating the model, which we derive from a generalisation of the well-known Mixtures-of-Experts model. As most of the LCS research is focused on model training rather than the model itself, we will also give an overview of how to train the presented model, and show preliminary experimental results. While we understand that the brevity of the presentation might make the details of the approach not immediately accessible, we feel that it needs to be presented as a whole to at least initially underline this holistic approach. Consequently, the experiments that we present cannot be reproduced by the reader, but we will make all necessary details available in forthcoming publications.

We start by giving a general description of LCS as a model that combines localised models (that is, the classifiers) to a global model. We then link such a structure to a generalised Mixture-of-Experts model, followed by discussing how to keep its training computationally tractable. Furthermore, we exploit the model by introducing a principled approach to the identification of the quality of a set of classifiers given the data, leading to a formal and general definition of the optimal set of classifiers. As this approach requires a Bayesian model, we fit priors to the probabilistic LCS model that make explicit the usually implicit

[1] A simple example is the application of Recursive Least Squares as an incremental approach to finding a solution that can equally be found be any Least Squares batch method.

prior assumptions that each model makes about the data. The quality metric on a set of classifiers turns the search for a good set of classifiers into an optimisation problem that can be handled by a genetic algorithm, which allows for the creation of a simple Pittsburgh-style LCS. Using such an implementation, we present some experimental results that show the applicability of the previously introduced concepts and conclude by pointing out the achievements and implications of this work.

2 Assembling an LCS Model

To create an adequate model for LCS we will firstly discuss the general structure of LCS models, based on characterising them as a member of the family of parametric models. This reveals that we can facilitate their similarity to the well-known Mixtures-of-Experts model to provide a probabilistic formulation for their model structure, describing a fixed set of classifiers.

2.1 A Bird's Eye View of the LCS Model

A parametric model family in ML can be characterised by the model structure $\mathcal{M}$ and the model parameters[2] $\boldsymbol{\theta}$. While the model structure is usually chosen before the model is trained, the model is fitted to the data by adjusting its parameters. For example, given the family of feed-forward neural networks, the number of hidden layers and nodes in each of the layers determines the model structure, and the model parameters are the weights of the connections between these nodes. Accordingly, the model structure commonly determines the number of model parameters that need adjustment.

While the same concepts apply to classification and reinforcement learning tasks, let us for now consider only regression tasks where the observations are assumed to be sampled from a target function f that maps the input space $\mathcal{X} = \mathcal{R}^{D_X}$ into the output space $\mathcal{Y} = \mathcal{R}$. In LCS, we have a set of K classifiers, each of which matches a subset of the input space. Considering classifier k, this classifier matches $\mathcal{X}_k \subseteq \mathcal{X}$ and provides a localised regression model $\hat{f} : \mathcal{X}_k \to \mathcal{Y}$, where the localisation is determined by $\mathcal{X}_k$ and is traditionally represented by the condition and action of a classifier. To provide a model of the full target function, the local models are *mixed* (that is, combined) in some way to provide the estimate $\hat{f} : \mathcal{X} \to \mathcal{Y}$, assuming that each element of $\mathcal{X}$ is matched by at least one classifier.

Leaving incremental training methods aside for now, this perspective reveals that the model structure $\mathcal{M}$ in LCS is in fact the number of classifiers in the population, and the parts of $\mathcal{X}$ that each of them matches. On the other hand, the model parameters $\boldsymbol{\theta}$ are the combined parameters of the regression model of

[2] While a *parameter* in LCS often refers to a constant that is set before training LCS and remains unchanged during training, we use it when referring to a variable of the model that is modified during training, and call a parameter in the LCS sense a *system parameter*.

each of the classifiers and the parameters of the model used to mix the classifier predictions. It also shows that LCS do not only aim at training a model $\mathcal{M}$ by adjusting the parameters of classifiers and mixing, but also tries to find an adequate model structure that fits (but does not overfit) the target function. While the second task is the more challenging one, let us for now concentrate on the first one, that is, how to adjust the model parameters for a given model structure, and come back to improving the model structure in Sect. 3. To do so, we need to define exactly the regression models underlying the classifiers and the model used for mixing their prediction.

Fortunately, Mixtures-of-Experts (MoE) [7,8] feature a similar model structure to LCS, and we can use this similarity to generalise MoE such that it corresponds to the model that underlies LCS. While we introduce the standard MoE model in the next section, we present the generalisations that make it similar to LCS in the section thereafter.

2.2 Mixtures of Experts

Mixture of Experts are probably most intuitively explained from the generative point-of-view: Let $\boldsymbol{X} = \{\boldsymbol{x}_n \in \mathcal{X}\}$ be the set of N inputs, and $\boldsymbol{Y} = \{y_n \in \mathcal{Y}\}$ the corresponding set of outputs, together giving the data $\mathcal{D} = \{\boldsymbol{X}, \boldsymbol{Y}\}$. For a set of K experts, each observation $\{\boldsymbol{x}, y\}$ is assumed to be generated by one and only one expert. We can model this by introducing the latent random vector $\boldsymbol{z} = (z_1, \ldots, z_K)^T$ of binary random variables, each of which corresponds to an expert. Given that expert $\bar{k}$ generated the observation, then $z_{\bar{k}} = 1$ and $z_k = 0$ for all $k \neq \bar{k}$. Hence, $\boldsymbol{z}$ has a 1-of-K structure, that is, it always contains one and only one element that is 1, with all other elements being 0.

Concentrating again on regression tasks, let the model of expert k be given by the conditional probability

$$p(y|\boldsymbol{x}, \boldsymbol{w}_k, \tau_k) = \mathcal{N}(y|\boldsymbol{w}_k^T \boldsymbol{x}, \tau_k^{-1}), \tag{1}$$

that is, by a univariate Gaussian with mean $\boldsymbol{w}_k^T \boldsymbol{x}$ and precision (that is, inverse variance) τ_k, where $\boldsymbol{w}_k$ is the weight parameter and τ_k is the precision parameter of expert k. This is a standard model for linear regression assuming constant noise variance over all observations and can easily be fitted by maximum likelihood, resulting in a linear least-squares problem.

As each observation is generated by one and only one expert, we can facilitate the 1-of-K structure of $\boldsymbol{z}$ to get the probability of y given $\boldsymbol{x}$ and all experts by

$$p(y|\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{\tau}, \boldsymbol{z}) = \prod_{k=1}^{K} p(y|\boldsymbol{x}, \boldsymbol{w}_k, \tau_k)^{z_k}, \tag{2}$$

where $\boldsymbol{W} = \{\boldsymbol{w}_k\}$, $\boldsymbol{\tau} = \{\tau_k\}$, and z_k are the elements of the latent variable $\boldsymbol{z}$ that corresponds to the observation $\{\boldsymbol{x}, y\}$.

If we know the values of $\boldsymbol{Z} = \{\boldsymbol{z}_n\}$, where $\boldsymbol{z}_n$ stands for the latent variable corresponding to observation $\{\boldsymbol{x}_n, y_n\}$, then we can train each expert independently

to fit only the observations that it generated. This can be seen by expanding the expression of the log-likelihood over the whole data

$$\ln p(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{W}, \boldsymbol{\tau}, \boldsymbol{Z}) = \ln \prod_{n=1}^{N} p(y_n|\boldsymbol{x}_n, \boldsymbol{W}, \boldsymbol{\tau}, \boldsymbol{z}_n)$$
$$= \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} \ln p(y_n|\boldsymbol{x}_n, \boldsymbol{w}_k, \tau_k),$$

where z_{nk} assigns the observations to the different experts. However, as $\boldsymbol{Z}$ is usually not known beforehand, we need to learn a model for it at the same time as training the experts. For that task MoE uses the multinomial logit model[3]; this is a standard model for categorical data and in the MoE context is known as the gating network, as it is responsible for associating observations and experts. It is defined by introducing another parameter vector $\boldsymbol{v}_k$ per expert that determines the probability of expert k having generated observation $\{\boldsymbol{x}_n, y_n\}$ by

$$g_k(\boldsymbol{x}_n) \equiv p(z_{nk} = 1|\boldsymbol{x}_n, \boldsymbol{v}_k) = \frac{\exp(\boldsymbol{v}_k^T \boldsymbol{x}_n)}{\sum_{j=1}^{K} \exp(\boldsymbol{v}_j^T \boldsymbol{x}_n)}. \tag{3}$$

This function is also known as the softmax function, and defines a soft linear partitioning over $\mathcal{X}$. The model emerges from the assumption that the relation between the probability of an expert k generating an observation $\{\boldsymbol{x}, y\}$ is log-linear in $\boldsymbol{x}$, that is $p(z_k = 1|\boldsymbol{x}, \boldsymbol{v}_k) \propto \exp(\boldsymbol{v}_k^T \boldsymbol{x})$.

Given the model structure $\mathcal{M}$ of MoE by the number of experts K, and having defined both the model for the experts and the gating network, the model parameters $\boldsymbol{\theta} = \{\boldsymbol{W}, \boldsymbol{\tau}, \boldsymbol{V}\}$ can be found by the EM-algorithm: in the E-step, the posteriors $p(z_{nk} = 1|\boldsymbol{x}_n, y_n, \boldsymbol{v}_k)$ are computed based on the current goodness-of-fit of the experts. The M-step uses these posteriors to adjust the expert and gating network parameters in order to maximise the likelihood of the data $\mathcal{D}$ and the latent variables $\boldsymbol{Z}$ [8]. This update has the effect that the gating network is adjusted according to the goodness-of-fit of the different experts, and the experts are trained according to how the gating network assigns the observations to the experts. In combination, this causes the input space to be partitioned by a soft linear partition, and each expert models the observations that fall in one of these partitions. Hence, the experts form localised models, where the localisation is determined by the gating network.

At this point the relation to LCS should be clear: The classifiers in LCS correspond to the experts in MoE, and the gating network has the same task as the mixing model in LCS. However, while the localisation of the classifiers in LCS is part of the model structure, the experts in MoE are localised by the interaction between the gating network and the experts. In the next section we show how an additional localisation layer in the MoE model can act as a generalisation to both the MoE model and the LCS model, and as such provides a strong probabilistic foundation for the LCS model.

[3] For details about the multinomial logit model and other generalised linear models see [9].

2.3 LCS as Generalised Mixtures of Experts

As a generalisation to the standard MoE model, we want to restrict the possibility of experts to generate observations to the inputs that the expert matches. Such matching is easily introduced by an additional binary random variable m_{nk} that is 1 if expert k matches input $\boldsymbol{x}_n$, and 0 otherwise. In contrast to the latent variables z_{nk}, m_{nk} does not need to obey the 1-of-K as several experts can match the same input. To enforce this matching, we define the probability of expert k generating the observation $\{\boldsymbol{x}_n, y_n\}$ to be

$$p(z_{nk} = 1 | m_{nk}, \boldsymbol{x}_n, \boldsymbol{v}_k) \propto \begin{cases} \exp(\boldsymbol{v}_k^T \vartheta(\boldsymbol{x}_n)) & \text{if } m_{nk} = 1, \\ 0 & \text{otherwise,} \end{cases} \tag{4}$$

where $\vartheta : \mathcal{X} \to \mathcal{R}^{D_V}$ is a transfer function over the input vectors, resulting in an additional generalisation over the MoE model, which uses $\vartheta(\boldsymbol{x}) = \boldsymbol{x}$. Therefore, if expert k matches input $\boldsymbol{x}_n$ then the probability of it generating the observation $\{\boldsymbol{x}_n, y_n\}$ is determined by a log-linear model as for the standard MoE model. If it does not match, however, then it could not have produced the observation either (that is, with probability 0). If we marginalise that probability over m_{nk}, we get

$$p(z_{nk} = 1 | \boldsymbol{x}_n, \boldsymbol{v}_k) \propto m_k(\boldsymbol{x}_n) \exp(\boldsymbol{v}_k^T \vartheta(\boldsymbol{x}_n)), \tag{5}$$

where we have defined the matching function $m_k(\boldsymbol{x}_n) \equiv p(m_{nk} = 1 | \boldsymbol{x}_n)$, giving the probability of expert k matching input $\boldsymbol{x}_n$. Adding the normalisation constant, we get the redefined gating network

$$g_k(\boldsymbol{x}_n) \equiv p(z_{nk} = 1 | \boldsymbol{x}_n, \boldsymbol{v}_k) = \frac{m_k(\boldsymbol{x}_n) \exp(\boldsymbol{v}_k^T \vartheta(\boldsymbol{x}_n)}{\sum_{j=1}^{K} m_j(\boldsymbol{x}_n) \exp(\boldsymbol{v}_j^T \vartheta(\boldsymbol{x}_n))}. \tag{6}$$

Comparing Eq. (6) to the gating network Eq. (3) of the standard MoE model, we can see the additional mediation by the matching functions. As matching is unchanged during the model fitting process, it is part of the model structure which is hence given by $\mathcal{M} = \{K, \boldsymbol{M}\}$, where $\boldsymbol{M} = \{m_k\}$ is the set of the expert's matching functions.

We do not need to modify the expert models, as by Eq. (4) an expert can only generate observations for a particular input if it matches that input, that is, $p(z_{nk} = 1 | \boldsymbol{x}_n, \boldsymbol{v}_k) > 0$ only if $m_k(x_n) > 0$. Thus, Eq. (2) still remains valid in the generalised MoE model.

To demonstrate that the model generalises over both MoE and LCS, we show how each of them can be recovered by fixing parts of the model structure: To get the standard MoE from our generalisation we simply need to assume a model structure where each expert matches all inputs. This structure is for some K given by the matching functions $m_k(\boldsymbol{x}_n) = 1$ for all n and k. Additionally, we have $\vartheta(\boldsymbol{x}) = \boldsymbol{x}$ as the gating network relies on the same inputs as the experts. LCS are not (yet?) as well defined as MoE and thus we could already claim that the generalised MoE by itself describes an LCS: A classifier corresponds to an expert with its matching function being specified by its condition/action

pair, that is, $m_k(\boldsymbol{x}) = 1$ if classifier k matches input $\boldsymbol{x}$, and $m_k(\boldsymbol{x}) = 0$ otherwise. Naturally, if the function ϑ is defined as something other than $\vartheta(\boldsymbol{x}) = 1$, then training the generalised MoE would cause the classifiers to be localised in regions of overlap beyond what is determined by their condition/action pair. While we have experimented with such a setup in [10], current commonly used LCS — such as XCS [1] and its derivates — perform mixing/gating based on an input-independent scalar, which can be modelled by setting $\vartheta(\boldsymbol{x}) = 1$ for all $\boldsymbol{x}$. Additionally, mixing is usually performed by heuristics (such as the normalised fitness in XCS), but having a better probabilistic justification like the multinomial logit model is certainly an advantage.

2.4 Training the Classifiers Independently

While the generalised MoE can be trained just like the standard MoE by using the EM-algorithm, its training comes with the same disadvantages: As the objective function for MoE is highly multi-modal, we will easily get stuck in local optima [11]. This problem is usually addressed by random restarts when training MoE, which still does not guarantee finding the optimal solution. In LCS, fitting a model to the data (that is, tuning its model parameters) is necessary to evaluate a certain model structure, but that needs to be performed many-fold when searching the space of possible model structures to find a good set of classifiers. As this space is potentially huge and very complex, we need to quickly be able to evaluate a single model structure, which is certainly not possible when utilising a random restart strategy.

Fortunately we do not need to look very far to solve this problem: XCS addresses it implicitly by not considering the interaction between classifiers and mixing. In fact, the multitude of local optima in the MoE model stem from the interdependence of expert and gating network training. Note that this interdependence is required to perform the necessary localisation of the experts. However, in our generalisation of the MoE model there is a second layer of localisation that is defined by the matching functions. Hence, for training the classifiers we can assume that the localisation of the different classifiers is fully defined by the matching function, which makes it independent of how their predictions are mixed. This has the advantages that i) classifiers can be trained by a single pass over the data; and ii) classifiers with the same associated condition/action always have the same parameter values, independent of the other classifiers in the population. The mixing parameters can now be either determined heuristically or, alternatively, trained in a single pass based on the goodness-of-fit of the different classifiers. On the downside, removing the link between classifier training and how they are mixed reduces the goodness-of-fit of the whole model, which needs to be counterbalanced by a more powerful model structure search.

Note that the modified training schema moves the model away from MoE towards ensemble learning where independently trained models are combined to form a single model. While this link has also been established independently

in [12], it is clearly beyond the scope of this paper to elaborate on its implication. Let us just point out that while interdependent classifier/mixing training assumes that each observation is generated by one and only one classifier, training the classifiers independently changes the assumptions about the data such that each observation is assumed to be a mixture of different generative processes, each of which is modelled by a different classifier.

3 Finding a Good Set of Classifiers

Probably the most important part of LCS is to find a good set of classifiers that fit the data. But what is a good quality metric when we want to evaluate the "goodness" of a set of classifiers? Its error when modelling the data is certainly an important component. However, given a set of observations, the model error is minimal if each observation is modelled by a single classifier — a not very satisfactory solution, given that it does not provide any more information about the data than the data itself. An alternative is to seek for the smallest set of classifiers that still results in an error-free fit of the data. Although better than one classifier per observation, this method would not fare well in the light of noisy data. XCS handles this problem by considering classifiers as accurate up to a user-defined error threshold and thus provides some form of accuracy/generalisation tradeoff. However, the solution does not give guidelines on how to set the error threshold and thus can overfit the data arbitrarily.

3.1 Applying Bayesian Model Selection

As already alluded to in the introduction, we approach the problem of defining the quality of a set of classifiers by assessing how well the model structure it represents explains the given data. Fortunately, this problem is well-known in machine learning and is handled by the field of *model selection*. The essential problem that model selection deals with is to find a model structure that does, on one hand, correctly identify all pattern within the data (within the realm of the model family) but avoids modelling randomness, essentially identifying a good tradeoff between generalisation and goodness-of-fit of a model. This is a difficult problem and different philosophical assumptions about the nature of randomness leads to different results, such as the Minimal Description Length [13] method or Statistical Learning Theory [14].

Bayesian model selection is a model selection method founded in Bayesian statistics which has fortunately already been applied to the standard MoE model [11,15]. It is based on the idea that the probability of a model structure given the data can be evaluated by

$$p(\mathcal{M}|\mathcal{D}) \propto p(\mathcal{D}|\mathcal{M})p(\mathcal{M}), \tag{7}$$

where $p(\mathcal{D}|\mathcal{M})$ is the goodness-of-fit of the data given a certain model structure, and $p(\mathcal{M})$ is the prior probability for that model structure[4]. Hence, given that we have a fully Bayesian model, the problem of finding a good model structure becomes as simple as finding one that maximises the model structure posterior $p(\mathcal{M}|\mathcal{D})$.

3.2 A Bayesian LCS Model

To apply Bayesian model selection we require a Bayesian LCS model, which we introduce by extending the probabilistic LCS model by adequate priors on its model parameters. The priors that we introduce are similar to the ones used by Waterhouse et al. for the Bayesian MoE model [17,18]. They are conjugate priors[5] where possible, except for the gating network parameters, where Laplace approximation (for example, [19]) is required to keep the evaluation of the parameter posteriors analytically tractable.

We also give recommendations on the prior parameters that cause them to be very uninformative; that is, in the light of some evidence the influence of the prior on the posterior is negligible. This makes the model selection criterion and subsequently also our definition of the optimal set of classifiers almost completely independent of the choice of priors.

Classifier Model Priors. For the univariate linear classifier model Eq. (1) we acquire the conjugate normal inverse-gamma prior

$$\begin{aligned} p(\boldsymbol{w}_k, \tau_k|\alpha_k) &= \mathcal{N}(\boldsymbol{w}_k|\mathbf{0}, (\alpha_k\tau_k)^{-1}\boldsymbol{I})\mathrm{Gam}(\tau_k|a_\tau, b_\tau) \\ &= \left(\frac{\alpha_k\tau_k}{2\pi}\right)^{D\mathcal{X}/2} \frac{b_\tau^{a_\tau}\tau_k^{(a_\tau-1)}}{\Gamma(a_\tau)} \exp\left(-\frac{\alpha_k\tau_k}{2}\boldsymbol{w}_k^T\boldsymbol{w}_k - a_\tau\tau_k\right), \end{aligned} \quad (8)$$

where $\mathrm{Gam}(\cdot|a_\tau, b_\tau)$ denotes the gamma distribution with scale a_τ and shape b_τ, and $\Gamma(\cdot)$ is the gamma function. This prior expresses that we expect the elements of the weight vector $\boldsymbol{w}_k$ to be independently distributed by a zero-mean Gaussian with precision (that is, inverse variance) $\alpha_k\tau_k$. In other words, we expect the weight elements to be small, which is a realistic assumption, given that the target function that a classifier aims at modelling is expected to be smooth. The noise precision τ_k is distributed according to a gamma distribution which we will parameterise as in [11] by $a_\tau = 10^{-2}$ and $b_\tau = 10^{-4}$ to keep the prior sufficiently broad an uninformative.

[4] Bayesian model selection differs from the approach of maximum likelihood in that, assuming a uniform model structure prior $p(\mathcal{M})$, Bayesian model selection aims at finding the $\mathcal{M}$ that maximises $p(\mathcal{D}|\mathcal{M})$, whereas maximum likelihood aims at finding the parameters $\boldsymbol{\theta}$ that maximise $p(\mathcal{D}|\boldsymbol{\theta}, \mathcal{M})$. The $p(\mathcal{D}|\mathcal{M})$ is found by marginalising over $p(\mathcal{D}|\boldsymbol{\theta}, \mathcal{M})$ (see Eq. (14)) which implicitly penalises the complexity of the parameter space $\{\boldsymbol{\theta}\}$ and hence protects from overfitting [16].

[5] Conjugate priors are priors that, when conditioned on the data likelihood, result in a posterior of the same distribution family as the prior.

Even though we could specify α_k directly as an additional prior parameter, we rather assign it a hyperprior as in [11] to allow it to automatically adjust to the data. We specify this hyperprior by the gamma distribution

$$p(\alpha_k|a_\alpha, b_\alpha) = \text{Gam}(\alpha_k|a_\alpha, b_\alpha) = \frac{b_\alpha^{a_\alpha} \alpha_k^{(a_\alpha - 1)}}{\Gamma(a_\alpha)} \exp(-a_\alpha \alpha_k), \tag{9}$$

with parameters $a_\alpha = 10^{-2}$ and $b_\alpha = 10^{-4}$ to keep it uninformative.

Gating Network Priors. In the prospect of applying Laplace approximation to the gating network model Eq. (6), resulting in a Gaussian, we apply conjugate Gaussian priors to the gating weights, given by

$$\begin{aligned} p(\boldsymbol{v}_k|\beta_k) &= \mathcal{N}(\boldsymbol{v}_k|\boldsymbol{0}, \beta_k^{-1}\boldsymbol{I}) \\ &= \left(\frac{\beta_k}{2\pi}\right)^{D_V/2} \exp\left(-\frac{\beta_k}{2}\boldsymbol{v}_k^T \boldsymbol{v}_k\right). \end{aligned} \tag{10}$$

This again corresponds to the assumption of having gating weight vectors with small and independent elements. β_k is again modelled by a hyperprior, given by the gamma distribution

$$p(\beta_k|a_\beta, b_\beta) = \text{Gam}(\beta_k|a_\beta, b_\beta) = \frac{b_\beta^{a_\beta} \beta_k^{(a_\beta - 1)}}{\Gamma(a_\beta)} \exp(-a_\beta \beta_k), \tag{11}$$

which parameter values $a_\beta = 10^{-2}$ and $b_\beta = 10^{-4}$ to keep the hyperprior uninformative.

Model Structure Prior. Recalling that $\mathcal{M} = \{K, \boldsymbol{M}\}$ is the number of classifiers K and their matching functions $\boldsymbol{M}$, the model structure prior $p(\mathcal{M})$ in Eq. (7) lets us specify any prior belief we have about the number of classifiers and their matching functions, such as that the number of classifiers is certainly not infinite. At first thought it might seem reasonable to specify it to express that every possible unique model structure is equally likely to represent the data. One should note, however, that the number of possible unique model structures for a fixed number of classifiers K grows exponentially with K. Thus, putting a uniform prior on the model structure space will put an implicit bias on model structures with a large number of classifiers.

The contrary, which is to put a uniform prior on the number of classifiers rather than the unique model structures, leads to a bias *against* particular model structures with higher numbers of classifiers, as for those there exist more possible model structures. Thus, how to appropriately define $p(\mathcal{M})$ is a topic of further investigation, but for now we have chosen to specify it by

$$p(\mathcal{M}) \propto \frac{1}{K!}, \tag{12}$$

to capture that in most LCS implementations the same model structure with K classifiers can be specified in $K!$ different ways by simply reordering the classifiers.

A summary of the full Bayesian model including its priors and hyperpriors is given in App. A. The variable dependency structure that shows the different random variables depend on each other is shown in Fig. 1.

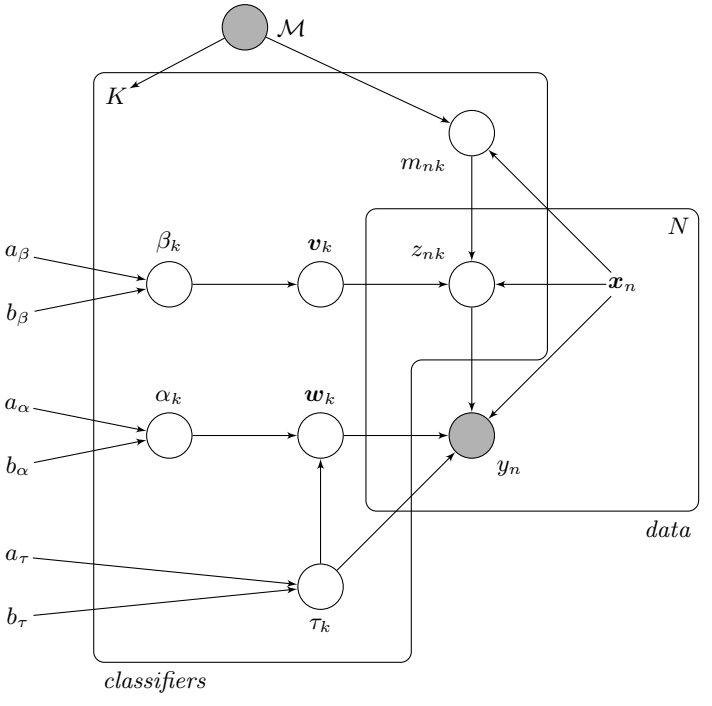


Fig. 1. Directed graphical model of the Bayesian LCS model. The circular nodes are random variables, which are observed when shaded. Labels without nodes are either constants or adjustable parameters. The boxes are "plates", comprising replicas of the entities inside them. Note that to train the model we assume the data $\mathcal{D}$ and the model structure $\mathcal{M}$ to be given. Hence, the y_n's and $\mathcal{M}$ are observed random variables, and the $\boldsymbol{x}_n$'s are constants.

3.3 Evaluating Posterior and Model Evidence

In order to evaluate the model structure posterior $p(\mathcal{M}|\mathcal{D})$ by Eq. (7), one needs to know $p(\mathcal{M})$ and $p(\mathcal{D}|\mathcal{M})$. In fact, as the posterior density is only used to compare different model structures, we can equally use the unnormalised log-density, that, by using Eq. (12), is given by

$$\ln p(\mathcal{M}|\mathcal{D}) = \ln p(\mathcal{D}|\mathcal{M}) - \ln K! + \text{const.}, \tag{13}$$

where the constant term represents the logarithm of the normalisation constant. To get the *model evidence* $p(\mathcal{D}|\mathcal{M})$ we need to evaluate

$$p(\mathcal{D}|\mathcal{M}) = \int_\theta p(\mathcal{D}|\boldsymbol{\theta}, \mathcal{M}) p(\boldsymbol{\theta}|\mathcal{M}) \mathrm{d}\boldsymbol{\theta}, \tag{14}$$

where $\boldsymbol{\theta} = \{\boldsymbol{W}, \boldsymbol{\tau}, \boldsymbol{V}, \boldsymbol{\alpha}, \boldsymbol{\beta}\}$ denotes the parameters of a model with structure $\mathcal{M}$, and $p(\boldsymbol{\theta}|\mathcal{M})$ represents their priors, as we have specified above. Unfortunately, there is no closed-form solution to Eq. (14) and so we have to either apply sampling techniques to sample from the posterior or resort to approximating it. We have decided for the latter, as sampling techniques are slower and would prohibit the quick evaluation of a large number of model structure, which is essential for LCS.

Variation Bayesian Inference with Factorial Distributions. Our goal is, on one hand, to find a variational distribution $p(\boldsymbol{U})$ that approximates the true parameter posterior $p(\boldsymbol{U}|\mathcal{D}, \mathcal{M})$ and, on the other hand, to get the model evidence $p(\mathcal{D}|\mathcal{M}) \equiv p(\boldsymbol{Y}|\boldsymbol{X}, \mathcal{M})$, where $\boldsymbol{U} = \boldsymbol{\theta} \cup \{\boldsymbol{Z}\}$ denotes all hidden variables. Variational Bayesian inference is based on the decomposition [20,19]

$$\ln p(\boldsymbol{Y}|\boldsymbol{X}, \mathcal{M}) = \mathcal{L}(q) + \mathrm{KL}(q||p), \tag{15}$$

$$\mathcal{L}(q) = \int q(\boldsymbol{U}) \ln \frac{p(\boldsymbol{U}, \boldsymbol{Y}|\boldsymbol{X}, \mathcal{M})}{q(\boldsymbol{U})} \mathrm{d}\boldsymbol{U}, \tag{16}$$

$$\mathrm{KL}(q||p) = -\int q(\boldsymbol{U}) \ln \frac{p(\boldsymbol{U}|\boldsymbol{X}, \boldsymbol{Y}, \mathcal{M})}{q(\boldsymbol{U})} \mathrm{d}\boldsymbol{U}, \tag{17}$$

which holds for any choice of the variational distribution q. As the Kullback-Leibler divergence $\mathrm{KL}(q\|p)$ is always non-negative, and zero if and only if $q(\boldsymbol{U}) = p(\boldsymbol{U}|\boldsymbol{X}, \boldsymbol{Y}, \mathcal{M})$, the variational bound $\mathcal{L}(q)$ is a lower bound on $\ln p(\boldsymbol{Y}|\boldsymbol{X}, \mathcal{M})$ and only equivalent to the latter if $q(\boldsymbol{U})$ is the true posterior $p(\boldsymbol{U}|\boldsymbol{X}, \boldsymbol{Y}, \mathcal{M})$. Hence, we can approximate the posterior by maximising the lower bound $\mathcal{L}(q)$, which brings the variational distribution closer to the true posterior and at the same time gives us an approximation of the model evidence by $\mathcal{L}(q) \leq \ln p(\boldsymbol{Y}|\boldsymbol{X}, \mathcal{M})$.

To make this approach tractable, we need to choose a family of distributions $q(\boldsymbol{U})$ that gives an analytical solution. A frequently used approach (for example, [17,11]) that is sufficiently flexible to give a good approximation to the true posterior is to use the set of distributions that factorises with respect to disjoint groups $\boldsymbol{U}_i$ of variables $q(\boldsymbol{U}) = \prod_i q_i(\boldsymbol{U}_i)$, which allows us to maximise $\mathcal{L}(q)$ with respect to each group of hidden variables while keeping the other ones fixed. This results in

$$\ln q_i^*(\boldsymbol{U}_i) = \mathbb{E}_{i \neq j}(\ln p(\boldsymbol{U}, \boldsymbol{Y}|\boldsymbol{X}, \mathcal{M})) + \text{const..}, \tag{18}$$

when maximising with respect to $\boldsymbol{U}_i$, where the expectation is taken with respect to all hidden variables except for $\boldsymbol{U}_i$, and the constant term is the logarithm of the normalisation constant of q_i^* [19].

Applying variational Bayesian inference to the Bayesian LCS model is a long-winded and complex process that we will not illustrate here, but is described in more detail in [21]. An analytical solution is only acquired if the model has a conjugate-exponential structure, which is violated by the gating network. Thus, we have applied Laplace approximation to its distribution, of which the details

can also be found in [21]. Overall, the procedure results in a closed-form expression for the variational bound $\mathcal{L}(q)$ that can be improved incrementally and can replace the model evidence $p(\mathcal{D}|\mathcal{M})$ in Eq. (7) to approximate the model structure posterior.

3.4 Summarising the Approach

Our starting point was to apply model selection to find the set of classifiers that model the data best, that is, without overfitting and underfitting. Approaching the problem by Bayesian model selection, the problem becomes the one of finding a set of classifiers $\mathcal{M}$ that maximises $p(\mathcal{M}|\mathcal{D})$. Given an efficient method to evaluate $p(\mathcal{M}|\mathcal{D})$ for any $\mathcal{M}$, any global optimisation method can be applied to search the space of possible sets of classifiers to maximise $p(\mathcal{M}|\mathcal{D})$.

To get an expression for $p(\mathcal{M}|\mathcal{D})$, or more specifically for the unnormalised $\ln p(\mathcal{M}|\mathcal{D})$ that can equally be used to compare different $\mathcal{M}$, we have applied variational Bayesian inference. This results in an approximation $\mathcal{L}(q)$ to $\ln p(\mathcal{D}|\mathcal{M})$ which can be used in Eq. (13) to approximate $\ln p(\mathcal{M}|\mathcal{D})$. Overall, this is sufficient to implement simple algorithms that search for the optimal set of classifiers for some data, with respect to the previously defined LCS model.

4 But..., Does it Work?

To illustrate that the introduced LCS design methodology leads to useful LCS implementations, we demonstrate the performance of the introduced LCS model and its training on two simple one-dimensional regression tasks. We understand that the brevity of the presentation does not allow the results to be replicated, but a forthcoming publication will, on one hand, give all the details that are required for replication and, on the other hand, present further results.

In all experiments we have used linear classifiers with input vectors $\boldsymbol{x} = (1, x)^T$ for input x, causing the model to by represented by a straight line.

4.1 Model Structure Search

To search the space of possible sets of classifiers we have applied a genetic algorithm (GA) to create a Pittsburgh-style LCS. The individuals represent model structures, and their fitness is the approximated model structure posterior, evaluated by Eq. (13). Thus, by searching for the fittest individual, the GA aims at finding the set of classifiers that maximises $p(\mathcal{M}|\mathcal{D})$.

We have used a rather small population of 20 individuals, and tournament selection with a tournament size of 5 individuals. The individuals are of variable length, and uniform crossover is performed by random sampling without replacement from the combined set of matching functions of two selected individuals. The mutation operator is dependent on the representation of the matching function and will not be detailed. In both experiments, the best individual of any of the 500 training epochs is reported.

To demonstrate that any global optimisation method can be used to find adequate model structures we have performed similar experiments using Monte Carlo Markov Chain (MCMC) methods, analogous to how it was applied by Chipman et al. in [22]. The found model structures were about the same as when applying the GA for model structure search, but we expect MCMC to perform worse in more complex problems where the GA can be expected to exploit building blocks in the solution structure. More details on applying MCMC to model structure search are given in [21].

4.2 Approximating a Generated Function

To test if the method correctly identifies the model structure when the data was generated in conformation to the LCS model assumptions, we have generated such data by combining the models of three localised classifiers with added Gaussian noise. To demonstrate the LCS model's ability to perform matching by degree, we use radial basis matching function, given by

$$m_k(x) = \exp\left(-(2\sigma_k^2)^{-1}(x-\mu_k)^2\right), \tag{19}$$

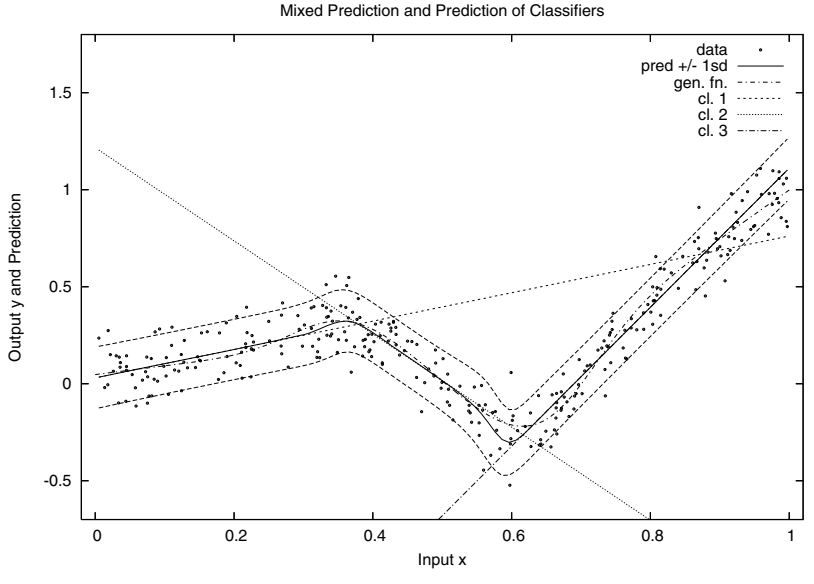


Fig. 2. The plot shows the mean of the data-generating function and the training data itself. Additionally, it shows the linear models of the 3 identified classifiers and the prediction of the full model with confidence intervals that represent one standard deviation to either side of the prediction. As can be seen, the method identified a model structure that fits the data well and is close in its parameter to the model that generated the data.

which is an unnormalised Gaussian that is centred on μ_k and has variance σ_k^2. Thus the classifier matches input $x = \mu_k$ with probability 1, and all other inputs with a probability that decreases in proportion to the distance from μ_k.

The best model structure found in a single run of the GA is shown in Fig. 2, illustrating the data, the classifiers in the model structure, and the mean prediction with confidence intervals. The latter is an additional feature of the model-based approach: due to the probabilistic model, we can make clear statements about the confidence of the prediction. As can be seen, the found model structure represents the data well. Additionally, the model structure parameters are fairly close to the ones that were used to generate the data, and we do not expect the search to find a perfect match as the model structure space is infinite.

4.3 Variable Measurement Noise

XCS seeks for classifiers that feature a mean absolute error close to a preset minimum error ϵ_0, leading to classifier models with approximately equal variances. The introduced LCS model is more flexible by allowing the classifier models to have different variances, depending on the given data. To test if this feature can be exploited we generate data where the level of noise varies over the input range. More specifically, the target function is for $-1 \leq x \leq 1$ given by $f(x) = -1-2x+\mathcal{N}(0, 0.6)$ if $x < 0$, and is $f(x) = -1+2x+\mathcal{N}(0, 0.1)$ otherwise, resulting in a V-shaped function with two different noise levels.

To let the classifiers match distinct areas of the input space we use interval matching with soft boundaries to indicate that in the light of finite data we can never be certain about where the interval boundaries lie. Given that classifier k matches the interval $[l_k, u_k]$, its matching function is given by

$$m_k(x) = \begin{cases} \exp\left(-\frac{1}{2\sigma_k^2}(x - l'_k)^2\right) & \text{if } x < l'_k, \\ \exp\left(-\frac{1}{2\sigma_k^2}(x - u'_k)^2\right) & \text{if } x > u'_k, \\ 1 & \text{otherwise,} \end{cases} \tag{20}$$

where $l'_k \approx l_k+0.0566b_k$, $u'_k \approx u_k+0.0566b_k$, $\sigma_k \approx 0.0662b_k$, and b_k is the interval width $b_k = u_k - l_k$. This causes the classifier to match the interval $[l'_k, u'_k]$ with probability 1, with unnormalised Gaussian boundaries that are with one standard deviation inside $[l_k, u_k]$ and otherwise outside of the interval. Additionally, 95% of the area underneath $m_k(x)$ is inside $[l_k, u_k]$.

The training data and best model structure found in a single training run with 500 epochs is shown in Fig. 3 and clearly shows by the width of its confidence intervals that the identified model features different noise levels in different areas of the input space. This illustrates that the method does not only protect from overfitting at the model structure level, but also at the classifier level by correctly separating the underlying pattern from the data.

The results of both experiments suggest that the method we have derived works as expected. Still, we want to emphasise that the developed method is only an example that show that the proposed model-based design methodology

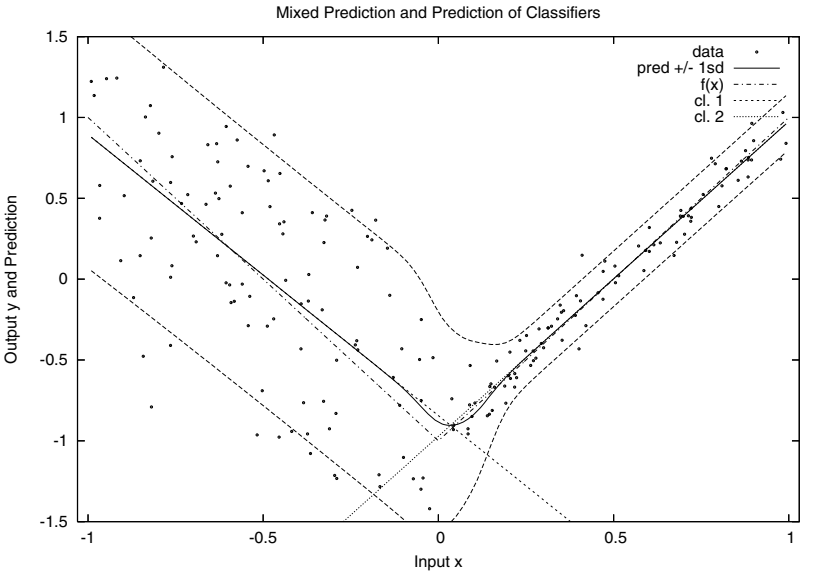


Fig. 3. The plot shows the mean of the data-generating function and the training data itself. In addition, it shows the linear models of the 2 identified classifiers and the prediction of the full model with confidence intervals that represent one standard deviation to either side of the prediction. These confidence intervals clearly show that the model is able to handle and model data whose noise level varies over the input space.

not only provides us with a good understanding of the model underlying different LCS variants, but also leads to useful and well-understood implementations.

5 Summary and Conclusions

We have proposed a new methodology for the design of LCS that is based on first specifying the model underlying a set of classifiers and then applying standard machine learning methods to train this model and identify a good set of classifiers. The advantages of this approach are manifold, such as i) having a formal definition of what it means for a set of classifiers to be the optimal set with respect to some data; ii) conceptually separating the LCS model from its training; iii) making explicit the assumptions the LCS model makes about the data; iv) establishing strong links to other machine learning methods through the application of standard machine learning methods for LCS training; and v) advancing the theoretical understanding of LCS by transferring the understanding of the applied machine learning methods to LCS. Furthermore, the methodology promises a high degree of flexibility in designing new LCS through changing the structure of the LCS model, or applying different machine learning methods for their training.

To demonstrate that the methodology is feasible, we have introduced a model for LCS and have described how it can be trained. In this work we have particularly

focused on the design of the model that is — closely related to XCS — specified as combining a set of independently trained localised models to form a global prediction over the whole input space. This characterisation allows it to be linked to a generalisation of the well-known Mixtures-of-Experts model, which puts it on a strong probabilistic foundation. For the identification of good model structures, that is, good sets of classifiers, we have used Bayesian model selection that results in the aim of maximising the probability of the model structure given the data. For training we have use the variational Bayesian method to find the model structure posterior, and have used a GA to search the space of possible model structures, resulting in a Pittsburgh-style LCS. To illustrate that the sets of classifiers identified that way indeed represent the data well, we have shown the methods performance on two simple regression tasks.

The work has wide implications and it is opening up significant future research directions, amongst which are i) to create a suitable LCS model specialised on classification by changing the classifier models from regression to classification models; ii) to compare and contrast LCS that train their classifiers independently to those that do not; iii) to design suitable methods, eventually using the additional probabilistic information that is available through the model, to apply the same methodology to design Michigan-style LCS; iv) analysing the suitability of the regression model for approximating the value function of reinforcement learning tasks.

Acknowledgements

We would like to thank Christopher M. Bishop, Markus Svensén and Matthew Beal for useful comments to our questions regarding the variational Bayesian approach.

References

1. Wilson, S.W.: Classifier Fitness Based on Accuracy. Evolutionary Computation 3(2), 149–175 (1995)
2. Butz, M.V., Pelikan, M.: Analyzing the Evolutionary Pressures in XCS. In: [23], pp. 935–942.
3. Butz, M.V., Goldberg, D.E., Tharakunnel, K.: Analysis and Improvement of Fitness Exploitation in XCS: Bounding Models, Tournament Selection and Bilateral Accuracy. Evolutionary Computation 11, 239–277 (2003)
4. Butz, M.V., Kovacs, T., Lanzi, P.L., Wilson, S.: Toward a Theory of Generalization and Learning in XCS. IEEE Transaction on Evolutionary Computation 8, 28–46 (2004)
5. Drugowitsch, J., Barry, A.M.: A Formal Framework for Reinforcement Learning with Function Approximation in Learning Classifier Systems. Technical Report 2006–02, University of Bath, U.K (January 2006)
6. Wilson, S.W.: Function Approximation with a Classifier System. In: [23], pp. 974–981.

7. Jacobs, R.A., Jordan, M.I., Nowlan, S., Hinton, G.E.: Adaptive mixtures of local experts. Neural Computation 3, 1–12 (1991)
8. Jordan, M.I., Jacobs, R.A.: Hierarchical mixtures of experts and the EM algorithm. Neural Computation 6, 181–214 (1994)
9. McCullach, P., Nelder, J.A.: Generalized Linear Models. Monographs on Statistics and Applied Probability. Chapman and Hall, Boca Raton (1983)
10. Drugowitsch, J., Barry, A.M.: Mixing Independent Classifiers. In: [24]
11. Bishop, C.M., Svensén, M.: Bayesian Hierarchical Mixtures of Experts. In: Proceedings of the 19th Annual Conference on Uncertainty in Artificial Intelligence (UAI 2003), pp. 57–64. Morgan Kaufmann, San Francisco (2003)
12. Brown, G., Kovacs, T., Marshall, J.: UCSPv: Principled Voting in UCS Rule Populations. In: [24], pp. 1774–1782.
13. Grünwald, P.D.: A tutorial introduction to the minimum description length. In: Grünwald, P., Myung, J., Pitt, M.A. (eds.) Advances in Minimum Description Length Theory and Applications. Information Processing Series, pp. 3–79. MIT Press, Cambridge (2005)
14. Vapnik, V.N.: An Overview of Statistical Learning Theory. IEEE Transactions on Neural Networks 10(5), 988–999 (1999)
15. Ueda, N., Ghahramani, Z.: Bayesian model search for mixture models based on optimizing variational bounds. Neural Networks 15, 1223–1241 (2002)
16. MacKay, D.J.C.: Bayesian interpolation. Neural Computation 4(3), 415–447 (1992)
17. Waterhouse, S., MacKay, D., Robinson, T.: Bayesian Methods for Mixtures of Experts. In: Touretzky, D.S.T., Mozer, M.C., Hasselmo, M.E. (eds.) Advances in Neural Information Processing Systems 8, pp. 351–357. MIT Press, Cambridge (1996)
18. Waterhouse, S.: Classification and Regression using Mixtures of Experts. PhD thesis, Department of Engineering, University of Cambridge (1997)
19. Bishop, C.M.: Pattern Recognition and Machine Learning. Information Science and Statistics. Springer, Heidelberg (2006)
20. Jaakkola, T.S.: Tutorial on variational approximation methods. In: Opper, M., Saad, D. (eds.) Advanced Mean Field Methods, pp. 129–160. MIT Press, Cambridge (2001)
21. Drugowitsch, J., Barry, A.M.: Generalised Mixtures of Experts, Independent Expert Training, and Learning Classifier Systems. Technical Report 2007–02, University of Bath, U.K (2007)
22. Chipman, H.A., George, E.I., McCulloch, R.E.: Bayesian CART Model Search. Journal of the American Statistical Association 93(443), 935–948 (1998)
23. Spector, L., Goodman, E.D., Wu, A., Langdon, W.B., Voigt, H.M., Gen, M., Sen, S., Dorigo, M., Pezeshk, S., Garzon, M.H., Burke, E. (eds.): GECCO-2001: Proceedings of the Genetic and Evolutionary Computation Conference. Morgan Kaufmann, San Francisco (2001)
24. Thierens, D., Beyer, H.G., Birattari, M., Bongard, J., Branke, J., Clark, J.A., Cliff, D., Congdon, C.B., Deb, K., Doerr, B., Kovacs, T., Kumar, S., Miller, J.F., Moore, J., Neumann, F., Pelikan, M., Poli, R., Sastry, K., Stanley, K.O., Stützle, T., Watson, R.A., Wegener, I. (eds.): GECCO-2007: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation Conference 2007, vol. 2. ACM Press, New York (2007)

A The Bayesian LCS Model

The following table gives an overview over the Bayesian LCS model with all its components.

Data, Model Structure, and Likelihood	
	N observations $\{(\boldsymbol{x}_n, y_n)\}$, $\boldsymbol{x}_n \in \mathcal{X} = \mathcal{R}^{D_\mathcal{X}}$, $y_n \in \mathcal{Y} = \mathcal{R}$
	Model structure $\mathcal{M} = \{K, \boldsymbol{M}\}, k = 1, \ldots, K$
	K classifiers
	Matching functions $\boldsymbol{M} = \{m_k : \mathcal{X} \to [0, 1]\}$
Likelihood	$p(\boldsymbol{Y}\|\boldsymbol{X}, \boldsymbol{W}, \boldsymbol{\tau}, \boldsymbol{Z}) = \prod_{n=1}^{N} \prod_{k=1}^{K} p(y_n\|\boldsymbol{x}_n, \boldsymbol{w}_k, \tau_k)^{z_{nk}}$
Classifiers	
Variables	Weight matrices $\boldsymbol{W} = \{\boldsymbol{w}_k\}, \boldsymbol{w}_k \in \mathcal{R}^{D_\mathcal{X}}$
	Noise precisions $\boldsymbol{\tau} = \{\tau_k\}$
	Weight shrinkage priors $\boldsymbol{\alpha} = \{\alpha_k\}$
	Noise precision prior parameters a_τ, b_τ
	α-hyperprior parameters a_α, b_α
Model	$p(y\|\boldsymbol{x}, \boldsymbol{w}_k, \tau_k) = \mathcal{N}(y\|\boldsymbol{w}_k^T \boldsymbol{x}, \tau_k^{-1})$
Priors	$p(\boldsymbol{w}_k, \tau_k\|\alpha_k) = \mathcal{N}(\boldsymbol{w}_k\|\boldsymbol{0}, (\alpha_k \tau_k)^{-1} \boldsymbol{I}) \mathrm{Gam}(\tau_k\|a_\tau, b_\tau)$
	$p(\alpha_k) = \mathrm{Gam}(\alpha_k\|a_\alpha, b_\alpha)$
Gating	
Variables	Latent variables $\boldsymbol{Z} = \{\boldsymbol{z}_n\}$, $\boldsymbol{z}_n = (z_{n1}, \ldots, z_{nK})^T \in \{0, 1\}^K$, 1-of-$K$
	Gating weight vectors $\boldsymbol{V} = \{\boldsymbol{v}_k\}$, $\boldsymbol{v}_k \in \mathcal{R}^{D_V}$
	Gating weight shrinkage priors $\boldsymbol{\beta} = \{\beta_k\}$
	β-hyperprior parameters a_β, b_β
Model	$p(\boldsymbol{Z}\|\boldsymbol{X}, \boldsymbol{V}, \boldsymbol{M}) = \prod_{n=1}^{N} \prod_{k=1}^{K} g_k(\boldsymbol{x}_n)^{z_{nk}}$
	$g_k(\boldsymbol{x}) \equiv p(z_k = 1\|\boldsymbol{x}, \boldsymbol{v}_k, m_k) = \frac{m_k(\boldsymbol{x}) \exp(\boldsymbol{v}_k^T \vartheta(\boldsymbol{x}))}{\sum_{j=1}^{K} m_j(\boldsymbol{x}) \exp(\boldsymbol{v}_j^T \vartheta(\boldsymbol{x}))}$
Priors	$p(\boldsymbol{v}_k\|\beta_k) = \mathcal{N}(\boldsymbol{v}_k\|\boldsymbol{0}, \beta_k^{-1} \boldsymbol{I})$
	$p(\beta_k) = \mathrm{Gam}(\beta_k\|a_\beta, b_\beta)$

Revisiting UCS: Description, Fitness Sharing, and Comparison with XCS

Albert Orriols-Puig and Ester Bernadó-Mansilla

Grup de Recerca en Sistemes Intel·ligents
Enginyeria i Arquitectura La Salle
Universitat Ramon Llull
Quatre Camins 2, 08022 Barcelona (Spain)
{aorriols,esterb}@salle.url.edu

Abstract. This paper provides a deep insight into the learning mechanisms of UCS, a learning classifier system (LCS) derived from XCS that works under a supervised learning scheme. A complete description of the system is given with the aim of being useful as an implementation guide. Besides, we review the fitness computation, based on the individual accuracy of each rule, and introduce a fitness sharing scheme to UCS. We analyze the dynamics of UCS both with fitness sharing and without fitness sharing over five binary-input problems widely used in the LCSs framework. Also XCS is included in the comparison to analyze the differences in behavior between both systems. Results show the benefits of fitness sharing in all the tested problems, specially those with class imbalances. Comparison with XCS highlights the dynamics differences between both systems.

1 Introduction

UCS [1] is a learning classifier system (LCS) derived from XCS [22,23] that works under a supervised learning scheme. UCS inherits the main components and structure of XCS, which are adapted for supervised learning. The main differences between both systems are related to 1) classifier's parameters and their update, and to 2) the lack of a prediction array in UCS. UCS's fitness is based on accuracy, computed as the percentage of correct classifications. This leads UCS to explore the consistently correct classifiers and thus evolve only best action maps.

Previous studies on artificial problems showed that UCS could overcome the fitness dilemma [6] that appeared in some problem categories in XCS, whose effect is a misleading pressure that tends to guide search in the wrong direction. Also evolution of best action maps was proved to be an efficient way of searching the search space, specially in large search spaces. UCS also showed to be competitive in real-world problems with XCS, as well as with some non-evolutionary learners such as C4.5 [19] and SMO [21].

Although UCS's results seem promising, there are still some open issues to be addressed. In [1], UCS's lack of fitness sharing was identified as a potential

J. Bacardit et al. (Eds.): IWLCS 2006/2007, LNAI 4998, pp. 96–116, 2008.

weakness. Fitness sharing is attributed to allow for better resource distribution among the different niches. Without fitness sharing, premature good rules can overtake the population, preventing potential niches from being explored and maintained in the population. Although this effect was not strongly observed in previous analyses, we acknowledge that, as current challenges such as unbalanced class problems are posed to UCS [18], the effect may become important. The literature has also proved that fitness sharing is beneficial for LCSs [3,7]. Other advances of the GA and LCS fields such as tournament selection [4] are also introduced in UCS to further optimize performance.

The aim of this paper is two-fold. Firstly, we want to provide a deeper description of all UCS components, detailed enough to be used as an implementation guide. The description also assembles other operators from XCS such as specify and includes new improvements of LCSs such as tournament selection. Also, a fitness sharing scheme is designed for UCS.

Secondly, the paper makes a comparison of UCS without fitness sharing, UCS with fitness sharing, and XCS. We use a testbed consisting of five binary-input classification problems that gather different complexity factors. Three of them were already used in [1]: a) the parity, b) the decoder, and c) the position problem. They represent respectively two cases of strong specialization with two classes (a) and multiple classes (b), and a case with multiple classes and non-uniform distribution of examples per class (c). We further add the imbalanced multiplexer problem and the multiplexer problem with noise. The designed testbed provides representative conditions of real-world problems for analyzing whether fitness sharing improves performance. By comparing UCS with XCS we update and enhance the previous comparison performed in [1] to the current settings. We seek to understand the dynamics of each approach and analyze conditions where each approach is better suited. The paper aims at enhancing our current comprehension of LCSs, and UCS in particular, and at providing a framework for further UCS system investigations and developments.

The remainder of this paper is organized as follows. Section 2 briefly introduces XCS. Next, UCS is described in detail, including UCS's fitness sharing version. Section 4 analyzes UCS with both fitness sharing and non-sharing, also compared to XCS. Finally, Section 5 summarizes the main differences observed, and Section 6 provides the main conclusions of our work. Appendix A provides a full explanation of all problems used in the comparison.

2 XCS in a Nutshell

This section provides a brief introduction to the XCS classifier system. The reader is referred to [22,23] for a detailed explanation and to [8] for an algorithmic description. Although XCS is applicable to several domains, such as reinforcement learning tasks [15] and function approximations [24], we will restrict our description to XCS as a pure classifier system.

XCS is an accuracy-based learning classifier system introduced by S.W. Wilson [22]. The main difference between XCS and previous strength-based LCSs is that

XCS bases fitness on the accuracy of the reward prediction instead of on the reward itself. This led XCS to avoid the presence of strong overgenerals that appeared in strength-based LCS [11]. The accuracy-based approach makes XCS evolve a complete action map (denoted as [O]) of the environment, evolving not only high-rewarded rules (i.e., consistently correct rules), but also consistently incorrect rules (i.e., rules with zero prediction and low error.).

XCS works as an online learner. For each input example, XCS forms the match set [M] consisting of all classifiers with matching condition. If not enough classes are covered in [M] XCS triggers the covering operator, which creates new classifiers with uncovered classes. Under pure exploration, a class is selected randomly, and all classifiers predicting that class form the action set [A]. The class is sent to the environment and the received reward is used to update the parameters of the classifiers in [A]. Eventually, the genetic algorithm is triggered inside the action set [A], and subsumption may be applied to favor accurate and general classifiers. Under exploit mode, XCS predicts the most voted class for each input example. The class vote is computed as a fitness weighted average of the predictions of all classifiers advocating that class.

3 Description of UCS

3.1 UCS Components

UCS is an accuracy-based learning classifier system introduced in [1]. It inherits the features of XCS, but specializes them for supervised learning tasks. UCS mainly differs from XCS in two respects. Firstly, the performance component is adjusted to a supervised learning scheme. As the class is provided with each new example, UCS only explores the class of the input examples. This implies that UCS only evolves high-rewarded classifiers, that is, the best action map [B]. Secondly, accuracy is computed differently in both systems. UCS computes accuracy as the percentage of correct classifications instead of computing it from the prediction error.

In the following, we give a deeper insight into UCS by explaining each component of the system.

Classifier's Parameters. In UCS, classifier's parameters are the following: a) accuracy acc; b) fitness F; c) correct set size cs; d) numerosity num; and e) experience exp. Accuracy and fitness are measures of the quality of the classifier. The correct set size is the estimated average size of all the correct sets where the classifier participates. Numerosity is the number of copies of the classifier, and experience is the number of times a classifier has participated in a match set.

Performance Component. UCS is an online learner that receives a new input example $x = (x_1, ..., x_n)$ at each learning iteration. As it works under a supervised learning scheme, also the class c of the example is provided. Then, the system creates the *match set* [M], which contains all classifiers in the population [P] whose condition matches x. Next, all the classifiers in [M] that predict the

class c form the *correct set* [C]. If [C] is empty, the covering operator is activated, creating a new classifier with a generalized condition matching x, and predicting class c. The remaining classifiers form the incorrect set ![C].

In test mode, a new input example x is provided, and UCS must predict the associated class. To do that, the match set [M] is created. All classifiers in [M] emit a vote, weighted by their fitness, for the class they predict. The vote of young classifiers (i.e., $exp < \theta_{del}$) is decreased by multiplying its vote by exp/θ_{del} to avoid that poorly evaluated classifiers emit a strong vote if more experienced classifiers exist. The most-voted class is chosen. New inference schemes have been proposed in [2]. Under test mode, the population of UCS does not undergo any change. All update and search mechanisms are disabled.

Parameter Updates. Each time a classifier participates in a match set, its experience, accuracy and fitness are updated. Firstly, the experience is increased. Then, the accuracy is computed as the percentage of correct classifications:

$$acc = \frac{number\ of\ correct\ classifications}{experience} \tag{1}$$

Thus, accuracy is a cumulative average of correct classifications over all matches of the classifier. Next, fitness is updated according to the following formula:

$$F_{micro} = (acc)^{\nu} \tag{2}$$

where ν is a constant set by the user that determines the strength pressure toward accurate classifiers (a common value is 10). Thus, fitness is calculated individually for each microclassifier, and it is not shared. The fitness of a macroclassifier F_{macro} is obtained by:

$$F_{macro} = F_{micro} \cdot num \tag{3}$$

Finally, each time the classifier participates in [C], the correct set size cs is updated. cs is computed as the arithmetic average of all sizes of the correct sets in which the classifier has taken part.

Discovery Component. The genetic algorithm (GA) is used as the primary search mechanism to discover new promising rules. The GA is applied to [C], following the same procedure as in XCS. It selects two parents from [C] with a probability that depends on classifier's fitness. The same selection schemes applied in XCS can be used in UCS, such as proportional selection or tournament selection. The two parents are copied, creating two new children, which are recombined and mutated with probabilities χ and μ respectively.

Finally, both children are introduced into the population. First, each offspring is checked for subsumption with its parents. The subsumption mechanism is adapted from XCS as follows: if one of the parents is sufficiently experienced ($exp > \theta_{sub}$), accurate ($acc > acc_0$) and more general than the offspring, then the offspring is not introduced into the population and the numerosity of this parent is increased. If the offspring cannot be subsumed for any of its parents, it is inserted in the population, deleting another classifier if the population is full.

Specify. The specify operator [14] is also adapted from XCS. At each learning iteration, the accuracy of the correct set $acc_{[C]}$ is compared to the average accuracy of the whole population $acc_{[P]}$. If $acc_{[C]} \leq \frac{acc_{[P]}}{2}$ and the average experience of the classifiers in [C] is greater than N_{Sp} ($exp_{[C]} \geq N_{Sp}$), a classifier is randomly selected from [C], with probability inversely proportional to its accuracy. Then, the classifier is copied, and its *don't care* symbols are specified —to the value of the input example— with probability P_{Sp}. Finally, the new classifier is introduced in the population, removing potentially poor classifiers if the population is full.

Deletion. The deletion probability of each rule is calculated as:

$$p_{del} = \begin{cases} \frac{cs \cdot num \cdot F_{[P]}}{F_{micro}} & \text{if } exp > \theta_{del} \text{ and } F_{micro} < \delta F_{[P]} \\ cs \cdot num & \text{otherwise} \end{cases} \quad (4)$$

where δ and θ_{del} are parameters set by the user, and $F_{[P]}$ is the average fitness of the population. In this way, deletion will tend to balance resources among the different correct sets, while removing low-fitness classifiers from the population. As fitness is computed from the percentage of correct classifications of a classifier, classifiers that predict wrong classes are not maintained in the population, and so, only the best action map evolves.

Classifier Parameters Initialization. UCS is very robust to parameter initialization since the initial value of most of the parameters is lost the first time that the classifier participates in a match set. When a classifier is created by covering, its parameters are set to: $exp = 1$, $num = 1$, $cs = 1$, $acc = 1$ and $F = 1$. If a classifier is created by the GA, its parameters are initialized to: $exp = 1$, $num = 1$, $cs = (cs_{p_1} + cs_{p_2})/2$ (where p_1 and p_2 denote each of the parents), $acc = 1$ and $F = 1$.

3.2 Why Should We Not Share Fitness?

UCS showed to perform competitively with other machine learning techniques in both artificial and real-world domains [1], even without fitness sharing. However, there are some analyses in the literature that demonstrate the advantages of sharing fitness in learning classifier systems [3] and, in general, in genetic algorithms [9].

Thus, we introduce a new fitness computation scheme that shares fitness, similarly to XCS, with the aim of comparing its advantages and disadvantages with a non sharing scheme. In the remainder of the paper, UCS without sharing is referred as UCSns, and UCS with sharing as UCSs.

Parameters update with fitness sharing works as follows. Experience, correct set size and accuracy are computed as in UCSns. However, fitness is shared among all classifiers in [M]. Firstly, a new accuracy k is calculated, which discriminates between accurate and inaccurate classifiers. For classifiers belonging to ![C], $k_{cl \in ![C]} = 0$. For classifiers belonging to [C], k is computed as follows:

$$k_{cl \in [C]} = \begin{cases} 1 & if \quad acc > acc_0 \\ \\ \alpha(acc/acc_0)^{\nu} & otherwise \end{cases}$$

Then, a relative accuracy k' is calculated:

$$k'_{cl} = \frac{k_{cl} \cdot num_{cl}}{\sum_{cl_i \in [M]} k_{cl_i} \cdot num_{cl_i}} \tag{5}$$

And fitness is updated from k':

$$F = F + \beta \cdot (k' - F) \tag{6}$$

Let's note that, under this scheme, the computed fitness corresponds to the macroclassifier's fitness, as numerosities are involved in the formulas. Whenever the microclassifier's fitness is needed (e.g., in the deletion algorithm), we use the relation of formula 3.

4 XCS and UCS in Binary-Input Problems

4.1 Methodology

The aim of this section is to analyze different aspects of UCS and XCS learning behavior. We base our analysis on five artificial problems that gather some complexity factors said to affect the performance of LCSs [12,1]: a) a binary-class problem; b) a multiclass problem; c) an imbalanced binary-class problem; d) an imbalanced multiclass problem; and e) a noisy problem. For a detailed description of each problem the reader is referred to appendix A. Each problem was run with UCSns, UCSs and XCS.

If not stated differently, we used the following *standard configuration* for XCS: $P_{\#} = 0.6$, $\beta = 0.2$, $\alpha = 0.1$, $\nu = 5$, $\theta_{GA} = 25$, $selection$ = tournament, $\chi = 0.8$, $\mu = 0.04$, $\theta_{del} = 20$, $\delta = 0.1$, GA_{sub} = true, $[A]_{sub}$ = false, $\theta_{sub} = 20$. When specify was enabled, $N_s = 0.5$ and $P_s = 0.5$. Parameters for UCS had the same values as in XCS, with acc_0=0.999 and ν=10.

The maximum population size of XCS and UCS was configured depending on the size of the optimal population that the systems were expected to evolve. XCS evolves a complete action map [O] [12], which consist of both highly rewarded rules[1] and poorly rewarded rules[2] with low error. On the other hand, UCS evolves the best action map [B] [1], which includes only highly rewarded rules. As proposed in [1], we set population sizes to $N = 25 \cdot |[O]|$ in XCS, and to $N = 25 \cdot |[B]|$ in UCS For each problem, we sought to find the parameters' configuration that permitted to obtain the best results. Thus, any changes on the parameter settings will be properly stated.

Different metrics were used to evaluate performance. We discarded accuracy as the measure of performance, since it does not provide enough evidence of effective genetic search, as pointed out in [13]. Instead of accuracy, the proportion

[1] Rules with high prediction and low error.
[2] Rules with low prediction and low error.

of the optimal action map achieved %[O] [12] was proposed as being a better indicator of the progress of the genetic search [13]. However, UCS and XCS evolve different optimal populations: XCS creates the complete action map [O], whereas UCS represents a best action map [B]. To allow a fair comparison, we only consider the proportion of best action map %[B] achieved by each system. That is, we only count the percentage of consistently correct rules. Besides, the proportion of correct classifications of the minority class (TP rate) is also used in imbalanced binary-class problems.

4.2 Binary-Class Problem: Parity

We tested the LCSs on the parity problem. The parity problem is a binary-class problem, whose class is 1 if the number of ones in the input is odd and 0 otherwise. The problem does not allow any generalization in the condition rules. Please, see appendix A.1 for the details. We ran the parity problem with condition lengths from 3 to 9 and the standard configuration. To stress specificity, we enabled the specify operator in XCS and UCS.

Figure 1 depicts the proportion of the best action map %[B] achieved by UCSns, UCSs and XCS. Curves are averages of ten runs. The plots show that UCSs converges earlier than UCSns, which shows up the advantages of fitness sharing. This is specially observed for larger input lengths, such as for 8 and 9 inputs. UCSs also improves XCS, and in turn XCS slightly improves UCSns. Next, we discuss these results in more detail.

Table 1. Accuracy and fitness of UCSns's classifiers along the generality-specifity dimension, depicted for the parity problem with $\ell = 4$

	Condition	Class	Accuracy	Fitness
1	####	0	0.5	0.00097
2	0###	0	0.5	0.00097
3	00##	0	0.5	0.00097
4	000#	0	0.5	0.00097
5	0000	0	1	1

a) Why is the Parity Problem Difficult for Both Systems?
The parity problem appears to be a hard problem for both learners. Its main difficulty can be attributed to the lack of *fitness guidance* toward optimal classifiers. As some generalization is introduced by the covering operator, XCS and UCS have to drive the population from generality to specificity. However, the fitness pressure does not correctly lead to specificity. That is, specifying one bit of an overgeneral classifier does not increase its accuracy unless all bits are specified. Table 1 shows the evolution that would suffer the most overgeneral classifier to become an optimal classifier in UCSns in the parity problem with ℓ=4. At each step, one of the *don't care* bits is specified. Note that accuracy, and so, fitness, remain constant during all the process until the optimal classifier is achieved. However, we would expect that the specification of one bit should result in a fitter classifier, since it

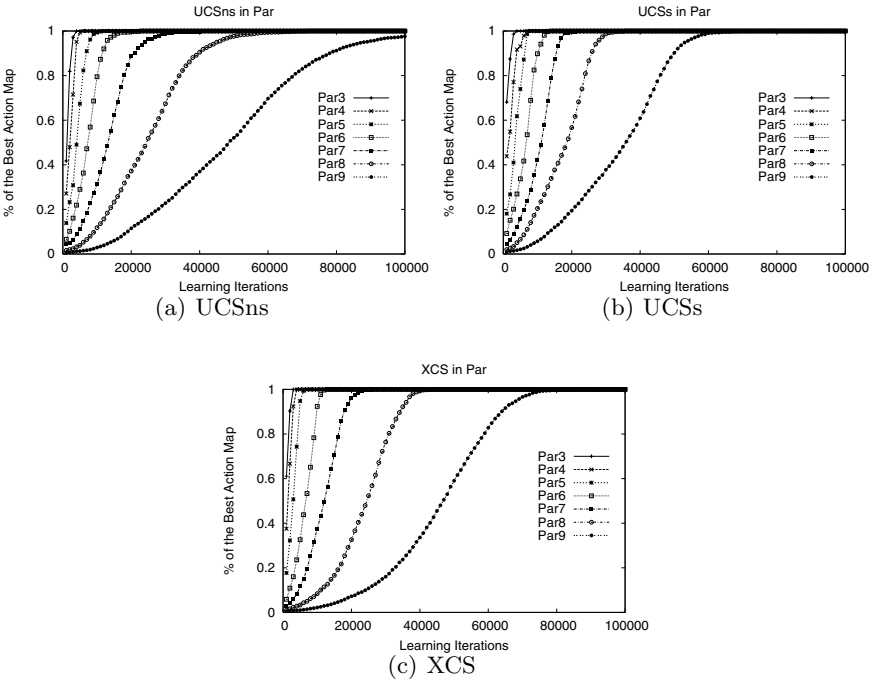


Fig. 1. Proportion of the best action map achieved by UCSns (a), UCSs (b) and XCS (c) in the parity problem with condition lengths from ℓ=3 to ℓ=9

approaches the optimum classifier (in which all the bits are specified). Therefore, the fitness is not guiding toward optimal optimal solutions, and so, an optimal classifier can only be obtained randomly. This example is also generalizable to XCS, for which fitness guidance is carefully analyzed in [6].

b) Exploring Only High-Rewarded Niches

Under a pure explore regime, XCS uniformly explores niches containing high-rewarded rules and niches with consistently incorrect rules (provided that examples come uniformly). Actually, the system maintains a complete action map. On the other hand, UCS explores only the high rewarded niches, i.e., the best action map.

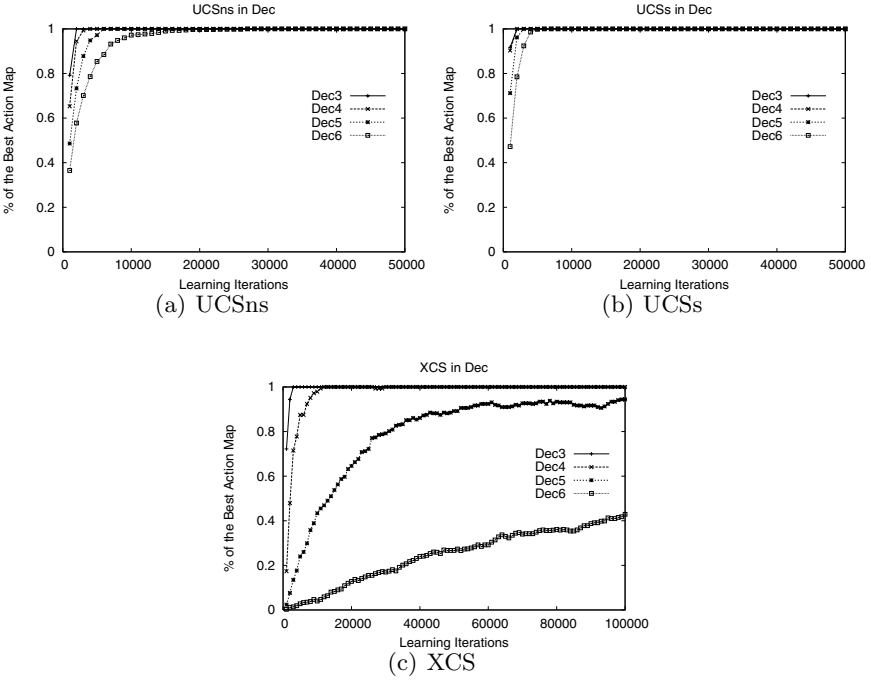
Thus, UCS is expected to evolve the best action map in half the time XCS does. However, results show that both systems need a similar number of learning iterations to converge. Our hypothesis is that the exploration of consistently incorrect rules may help XCS to discover consistently correct rules. For example, if XCS evolves a consistently incorrect classifier such as `0001:0`, XCS may discover the consistently correct classifier `0001:1` by mutation of the class bit. Thus, XCS can discover parts of [B] while exploring low-rewarded niches.

c) The Advantages of Sharing Fitness

The results show that the convergence curves in UCSs and XCS are steeper than in UCSns. After some initial iterations, the systems start to evolve some optimal classifiers. In the case of UCSs and XCS, once the first optimal classifiers are discovered, the convergence curves raise quickly. This effect is due to fitness sharing. Under fitness sharing, the discovery of a new optimal classifier makes the fitness of overgeneral classifiers that participate in the same action/correct set decrease quickly. This produces a) a higher pressure toward overgeneral deletion and b) a higher selective pressure toward specific classifiers in the GA. Thus, the GA is likely to produce new specific, and so, optimal classifiers. Without fitness sharing, overgeneral classifiers maintain the same fitness along the whole learning process. This effect is specially strong in imbalanced data sets, where overgeneral classifiers have higher accuracy.

4.3 Multiclass Problem: Decoder

In this section we aim at analyzing the behavior of XCS and UCS on multiclass problems. For this purpose, we use the decoder problem, which has $\ell + 1$ classes, being ℓ the string length. The best action map contains 2^ℓ specific classifiers, while the complete action map has $(\ell+1)\cdot 2^\ell$ classifiers. See appendix A.2 for the details.



(a) UCSns (b) UCSs

(c) XCS

Fig. 2. Proportion of the best action map achieved by UCSns (a), UCSs (b) and XCS (c) in the decoder problem with condition lengths from ℓ=3 to ℓ=6. Note that UCS is shown for 50,000 explore trials, while XCS is shown for 100,000 trials.

Figure 2 depicts the proportion of the best action maps achieved by UCSns, UCSs and XCS. The results evidence a much better performance of UCS, specially for higher condition lengths. With a condition length ℓ=6, UCSs needs up to 5,000 learning iterations to discover the complete best action map, whereas XCS discovers only 40% of it in 100,000 iterations. This huge difference between XCS and UCS could mainly be explained by a) the explore regime and b) the accuracy guidance toward the optimal population.

a) Explore Regime

The first aspect that hinders the performance of XCS is the explore regime used. As XCS explores uniformly each class, only 1 of each $\ell + 1$ explores will be made on the class of the input instance. The other ℓ explores will be focused on classifiers that predict wrong classes. This issue, which appeared to have low effect in the parity problem, now takes importance since the number of classes increases. Moreover, the hamming distance between consistently incorrect classifiers and correct classifiers is bigger than in the parity problem. So, in the decoder problem, it is more difficult to create a high-rewarded classifier while exploring a low-rewarded niche.

b) Accuracy Guidance Toward Optimal Population

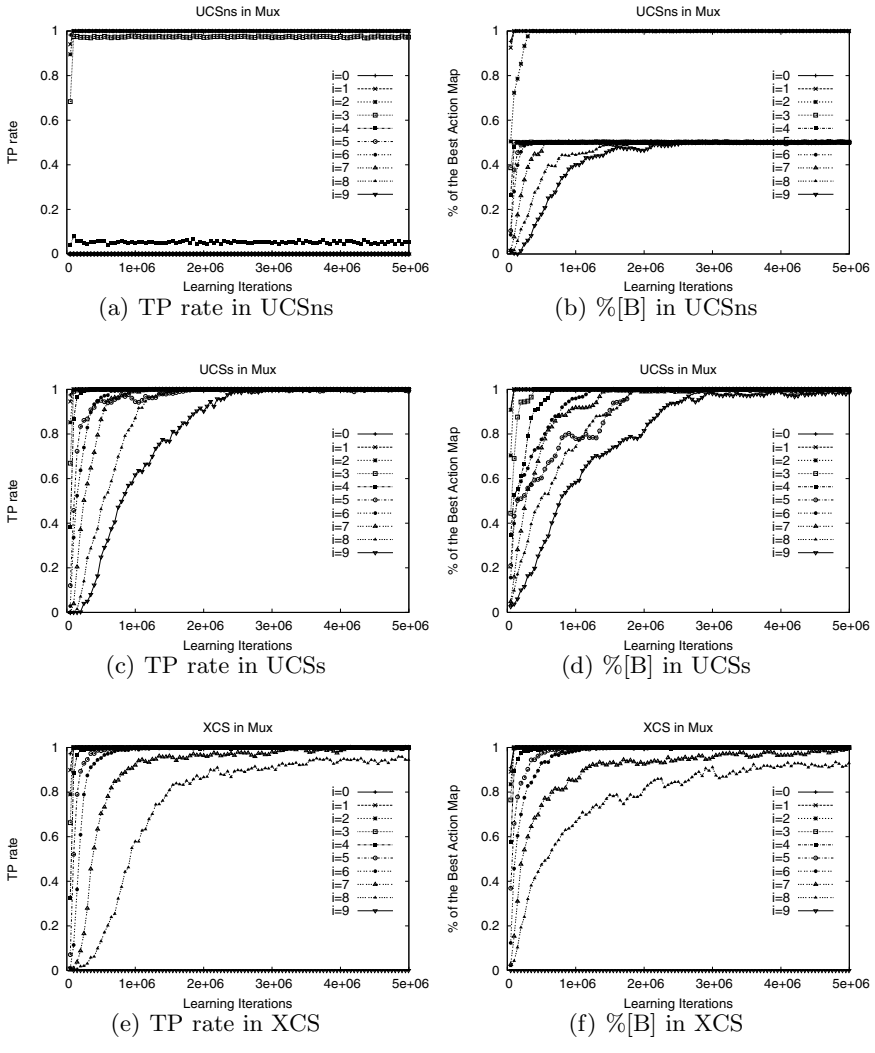
The second point that influences the convergence of XCS is the way in which fitness guides to the solution. This issue was initially termed by Butz et al. as the *fitness dilemma* [6]. Let's suppose we have the overgeneral classifier cl_1 `1###:8`, whose estimated parameters are P=125 and ϵ = 218.75 [6]. XCS is expected to drive cl_1 to the classifier `1000:8`. Imagine now that, either mutation or specify generates the classifier cl_2 `10##:8`, whose parameter estimates are P=250 and ϵ = 375. That is, as the classifier approaches the optimal one, the error increases. As long as the classifier moves towards the right direction, it gets smaller fitness, which consequently means fewer genetic event opportunities and higher deletion probabilities. Thus, there is no effective fitness pressure towards optimal classifiers.

UCS overcomes the *fitness dilemma*, because the accuracy is calculated as the percentage of correct classifications. Therefore, classifier cl_2 would have higher accuracy than cl_1. In that way, UCS's accuracy guidance does not mislead the search.

Finally, it can be observed that UCSs converges slightly quicker than UCSns. As in the parity problem, we ascribe this behavior to the fact that fitness sharing makes a stronger pressure toward the optimal classifiers as long as they are discovered.

4.4 Imbalanced Binary-Class Problem: Imbalanced Multiplexer

Real-world problems often have smaller proportion of examples of one class than the others. It has been widely accepted that this issue, addressed as the class imbalance problem [10], could hinder the performance of some well-known learners. Recent studies [16,18] have shown that UCSns suffers from class imbalances,



(a) TP rate in UCSns (b) %[B] in UCSns

(c) TP rate in UCSs (d) %[B] in UCSs

(e) TP rate in XCS (f) %[B] in XCS

Fig. 3. TP rate and % of [B] achieved by UCSns, UCSs and XCS in the 11-bit multiplexer problem with imbalance levels from i=0 to i=9

while different mechanisms acting at the classifier level or at the sampling level can alleviate the effect. On the other hand, XCS has shown to be highly robust to class imbalances if its parameters are set appropriately (see [17]). In this section we compare the behavior of XCS and UCS in a controlled imbalanced problem, and besides, analyze the influence of fitness sharing in such conditions.

We used the imbalanced 11-bit multiplexer (see appendix A.4), where minority class instances were sampled with a proportion P_{accept}. In the remainder of the paper we use i to refer to the imbalance level, that is, $P_{accept} = \frac{1}{2^i}$.

We ran XCS and UCS with the 11-bit multiplexer and imbalance levels from i=0 to i=9. Parameters were set as suggested in [17]: for high imbalance levels, β should be increased to allow more reliable parameter estimates, while θ_{GA} should be decreased to allow more uniform genetic opportunities among the different niches. Moreover, we only apply subsumption when the involved classifiers have received, at least, 2^i parameter updates. Thus, for XCS we used the standard configuration but setting $\beta = \{0.04, 0.02, 0.01, 0.005\}$, $\theta_{GA} = \{200, 400, 800, 1600\}$, and $\theta_{sub} = \{64, 128, 256, 512\}$ for i=$\{6, 7, 8, 9\}$ respectively. As UCS appeared to be less sensitive to parameters' settings in previous experiments (not reported here), we maintained the standard configuration. Only for i$\geq$6, $\theta_{GA} = 50$ and $\beta = 0.02$.

Figure 3 shows the proportion of correct classifications of the minority class (TP rate) and the proportion of the best action map achieved by each system. We do not show the proportion of correct classifications of the majority class since it raises to 100% in few iterations. The results show that UCSns fails at low imbalance levels (it only converges for i$\leq$3), whereas XCS solves the problem up to i=8, and UCSs up to i=9.

Let's analyze the behavior of UCSns. Figure 3(b) shows that, for i$\geq$4, UCSns evolves only half of the best action map. Looking at the populations evolved (see table 2 for $i = 6$) we observed that UCSns discovered all optimal classifiers predicting the minority class (class 1). However, optimal classifiers predicting the majority class were replaced by the most overgeneral classifier cl_{ovg}: `###########:0`. This behavior is related to the accuracy computation and the lack of fitness sharing in UCSns. At any imbalance level, the accuracy expected for cl_{ovg} is $acc_{cl_{ovg}} = 1 - P_{accept}$, and the fitness is computed as a power of the accuracy. For i=6, $acc_{cl_{ovg}} = 0.9843$, which gives a high fitness value. The classifier's accuracy is still lower than acc_0 (recall that acc_0 is set to 0.999); thus, the classifier would be considered as inaccurate for subsumption purposes. However, since cl_{ovg} participates in $1 - P_{accept}$ of all correct sets, the classifier receives many genetic opportunities, and finally overtakes the population. Increasing ν

Table 2. Most numerous rules evolved in a single run of UCSns with the 11-bit imbalanced multiplexer for i=6. Cond. is the classifier's condition, C. the class it predicts, and Acc., F., and Num. are the accuracy, fitness and numerosity of the classifier.

Cond.	C.	Acc.	F.	Num.
`###########`	0	0.98	0.86	142
`0001#######`	1	1.00	1.00	55
`001#1######`	1	1.00	1.00	74
`010##1#####`	1	1.00	1.00	60
`011###1####`	1	1.00	1.00	63
`100####1###`	1	1.00	1.00	57
`101#####1##`	1	1.00	1.00	69
`110######1#`	1	1.00	1.00	57
`111#######1`	1	1.00	1.00	68
...				

to make a stronger pressure toward accuracy does not significantly improve the results for high imbalance levels.

This effect does not appear in XCS and UCSs with appropriate parameter settings. As both systems share fitness, the fitness of overgeneral classifiers would considerably diminish when an optimal classifier is discovered. Figures 3(c)-3(f) show the improvement obtained with UCSs and XCS. UCSs shows to perform slightly better at high imbalance levels, being able to solve the 11-bit multiplexer even for i=9, in which XCS fails. At these regimes of such low supply of minority class examples, exploring only the correct class is crucial to maximally benefit from exploration. In fact, as XCS explores half of the correct actions for a given number of iterations (with respect to UCS), one could expect that UCSs solves the multiplexer up to one imbalance level greater than XCS does.

4.5 Imbalanced Multiclass Problem: Position

After analyzing the effects of class imbalance on an artificially imbalanced problem, we introduce the position problem, which intrinsically has unequal distribution of examples per class, and a number of classes that increases linearly with the condition length (see appendix A.3 for a description of the problem).

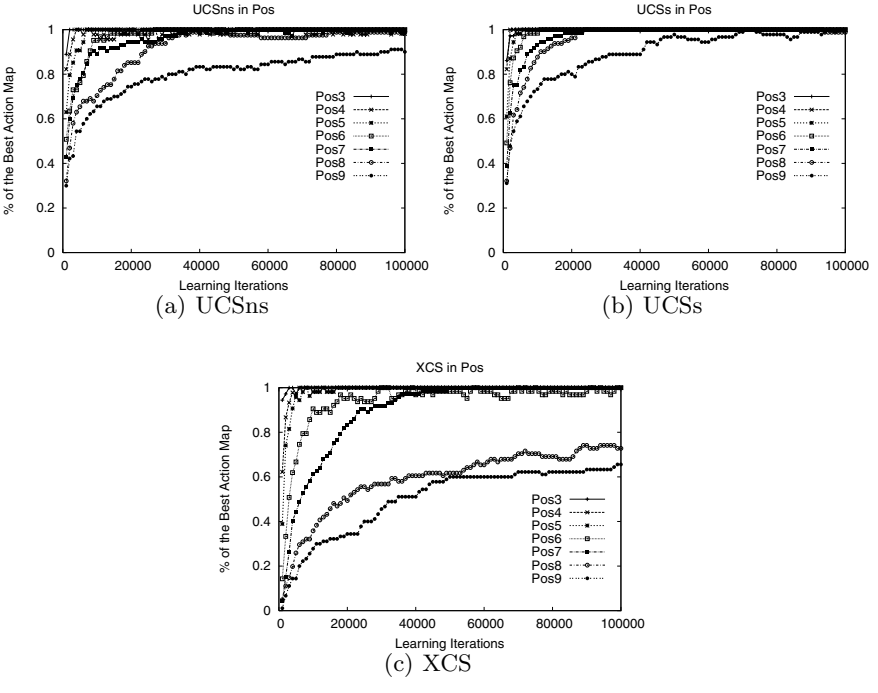


Fig. 4. Proportion of the best action map achieved by UCSns (a), UCSs (b) and XCS (c) in the position problem with condition lengths from l=3 to l=9

We ran UCSns, UCSs and XCS with the position problem and the standard configuration. Specify was disabled to introduce more generalization pressure. Figure 4 shows the proportion of the best action map achieved.

The position problem combines the effects of class imbalances and fitness dilemma in a single problem. The best action map consists of classifiers with different levels of generality. Thus, the most specific rules are expected to suffer from fewer genetic opportunities, since they are activated less often [1]. The results obtained support this hypothesis. In all cases, the shape of the curves have a steep increase at the beginning —where the most general rules are discovered—, and afterward, the curves improve slowly. Examining the populations (not shown for brevity), we confirmed that the most specific rules were difficult to discover.

In addition, the results obtained with XCS are poorer than with UCS. This can be again ascribed to the fitness dilemma, as in the decoder problem. That is, there is a misleading pressure toward optimal rules. Figure 5 shows an example of how the prediction error increases when driving the overgeneral rule $\#\#\ldots\# : 0$ to the optimal classifier $00\ldots : 0$ for different input lengths ℓ. This, coupled together with the few genetic opportunities that the most specific classifiers receive, makes the discovery of these classifiers very hard.

Finally, UCSs slightly outperforms UCSns. For position with $\ell = 9$, after 100,000 iterations, UCSs is able to discover all the best map while UCSns discovers 90% of the best action map. Again, fitness sharing helps to discover the most specific classifiers.

4.6 Noisy Problem: Multiplexer with Alternating Noise

To conclude the study, we analyze the effects of training XCS and UCS on noisy environments. To do that, we used the 20-bit multiplexer introducing alternating noise (denoted as mux_{an}). That is, the class of a input instance was flipped with probability P_x, as proposed in [5].

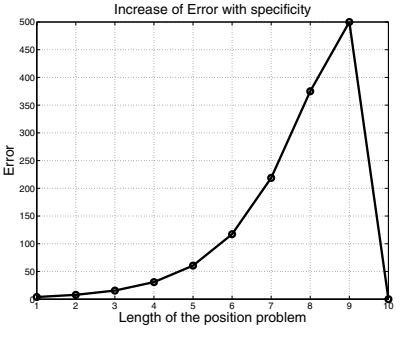
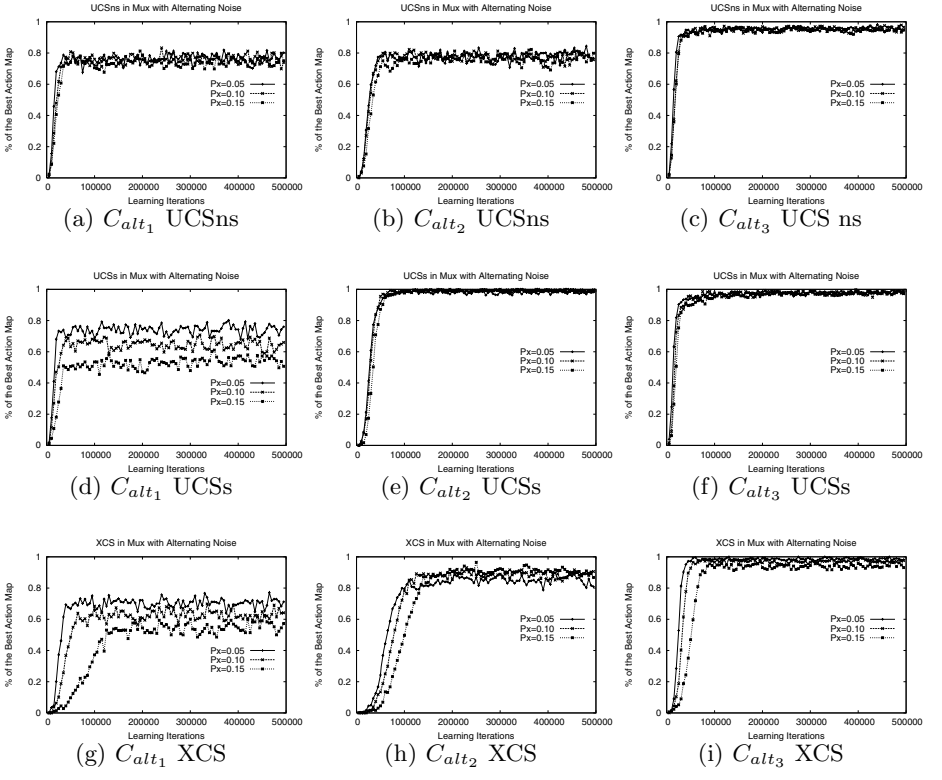


Fig. 5. Error of XCS's classifiers along the generality-specificity dimension. The curve depicts how the error of the most overgeneral classifier $\#\#\ldots\# : 0$ evolves until obtaining the maximally accurate rule $00\ldots : 0$.



(a) C_{alt_1} UCSns (b) C_{alt_2} UCSns (c) C_{alt_3} UCS ns

(d) C_{alt_1} UCSs (e) C_{alt_2} UCSs (f) C_{alt_3} UCSs

(g) C_{alt_1} XCS (h) C_{alt_2} XCS (i) C_{alt_3} XCS

Fig. 6. Proportion of the best action map achieved by UCSns, UCSs, and XCS in the 20-bit multiplexer with alternating noise and with configurations C_{alt_1}, C_{alt_2}, and C_{alt_3}

The effect of noise may be related to the effect of undersampled classes to some extent. The system receives wrongly labeled instances in a percentage P_x of the explore trials. The key difference here is that each instance is correctly labeled in a high proportion of samples (1 - P_x), and only in small proportion of cases (P_x), the instance comes with the wrong class. Thus, we aim at analyzing the ability of each system to obviate noisy instances in favor of non-noisy ones.

We ran UCSns, UCSs and XCS in the 20-bit multiplexer with alternating noises $P_x = \{0.05, 0.10, 0.15\}$ and three different parameter configurations C_{alt_1}, C_{alt_2} and C_{alt_3}. C_{alt_1} is the standard configuration with specify disabled. C_{alt_2} sets β=0.01 and $\theta_{GA} = 100$ to have more reliable parameters estimates before the GA triggers. Finally, C_{alt_3} is based on C_{alt_1}, setting $acc_0 = 1 - P_x$ and $\epsilon_0 = P_x * R_{max}$ to let the system admit the noise of the data. In all runs, the population size was set to 2,000 in both UCS and XCS[3]. Figure 6 shows the percentage of optimal population achieved for each setting.

[3] UCS population size is equal to XCS, because the effect of wrongly labeled instances makes UCS try to evolve consistently incorrect rules to fit to noise.

With C_{alt_1} setting, UCSns gets higher performance than UCSs and XCS, specially for the highest levels of noise (see figures 6(a), 6(d) and 6(g)). This behavior can be attributed to the way that classifier parameters are estimated. Both XCS and UCSs compute a windowed average of fitness by means of the learning parameter β. In noisy environments, the parameter averages oscillate and cannot stabilize properly. So, high levels of noise require low values of β. As UCSns computes fitness as a power of the accuracy, parameter values of experienced classifiers remain steady.

Figures 6(b), 6(e) and 6(h) show the proportion of the best action map achieved with C_{alt_2} setting, which is configured to lead to more stable parameters estimates. The results show a clear improvement of UCSs and XCS with respect to configuration C_{alt_1}. This proves our hypothesis that higher levels of noise require lower β values to allow for better parameter stabilization, coupled with higher θ_{GA} to let the genetic algorithm operate with better estimates. UCSs specially benefits from that, nearly reaching 100% of the best action map in few iterations. Results of UCSns are the same as those obtained with C_{alt_1} setting, as it does not use β in the fitness estimate.

Finally, figures 6(c), 6(f) and 6(i) show the experiments with C_{alt_3} setting, which fits acc_0 and ϵ_0 to consider noise as negligible. This setting allows optimal classifiers, which include a certain percentage of noise, to be considered as accurate. This lets optimal classifiers have higher accuracy estimates (in the case of UCSs and XCS). Besides, subsumption would be applied toward optimal classifiers increasing the pressure toward them. Results appear better to those obtained with C_{alt_1} in all LCSs. With respect to C_{alt_2}, we obtained better results than with C_{alt_3} in UCSns and XCS. Combining C_{alt_2} and C_{alt_3} benefits from the advantages of each approach (not shown for brevity).

5 Summing Up

In this section, we briefly sum up the differences observed when sharing is introduced in UCS, as well as the different dynamics between UCS and XCS.

Explore Regime. Exploring only the class of the input instance (as UCS does) appeared to be beneficial in problems with high number of classes, e.g., decoder and position. Moreover, it helped to solve the imbalanced multiplexer up to one imbalance level higher than XCS, in an extreme low supply of minority class instances. Finally, the effects were nearly imperceptible in the parity problem, since the exploration of the low-rewarded niches could lead to discover high-rewarded optimal classifiers.

We used a pure explore regime in XCS, where each available class is uniformly explored for each possible input. Since UCS proves that a more directed search toward the correct class generally speeds up convergence, we could also think of other explore regimes in XCS. Training in XCS could be based on an exploration regime that gradually changes from pure exploration toward increasing exploitation, similarly to schemes such as softmax [20]. This remains an open issue for further work.

Accuracy Guidance. The results of XCS in some domains showed the lack of fitness guidance toward accurate classifiers. This problem, already observed in previous studies [1,6], was termed the fitness dilemma in [6]. The problem does not exist in UCS since accuracy is computed directly as the percentage of correct classifications. We showed that XCS strongly suffers from the fitness dilemma in the decoder, and to a lower degree, in the position. In these cases, UCS clearly outperformed XCS. To alleviate this effect, bilateral accuracy was proposed for XCS [6]. As a future work, we aim to investigate how this approach compares with UCS.

Fitness Sharing. Fitness sharing speeds up the convergence in all problems tested. Specially, it appears to be crucial in highly imbalanced data sets to deter overgeneral classifiers from overtaking the population. UCSns only gave better performance when parameter estimates were unstable in UCSs due to an inappropriate setting of the learning parameter β. Anyway, this effect cannot be attributed to the presence or absence of fitness sharing, but rather to the way in which fitness is estimated. Recall that UCSns computes fitness as a power of accuracy, while UCSs computes fitness as a weighted windowed average with learning parameter β.

Population Size. In the tested problems, UCS evolved best action maps with less learning iterations. Also smaller population sizes were used in UCS in all the tested problems, except for the noisy problem. The population evolved by XCS is generally larger, but comparable to that of UCS in terms of legibility. In fact, by removing low-rewarded classifiers from XCS's final population, we get a set of rules similar to that of UCS (not shown for brevity). Thus, the advantage of having smaller populations in UCS lies in the reduction of computational resources.

6 Conclusions

This paper provided insight into the UCS learning classifier system. We improved the original UCS system as introduced in [1] by including tournament selection and fitness sharing. Robustness of the modified UCS was proved across different artificial domains. Specially, fitness sharing was necessary in the imbalanced multiplexer problem. We suspect that this behavior can be also generalizable to other imbalanced problems, where overgeneral classifiers can easily become strong. Using sharing, we allow overgenerals until optimal classifiers start to evolve. When this happens, fitness of overgeneral classifiers decreases fast by the effect of sharing fitness with better competing solutions.

The comparison with XCS allowed for a better understanding of the differences between the two approaches of accuracy-based classifier systems. UCS has an architecture specifically designed for supervised learning problems. XCS is more general, and can be applied to multi-step problems, learning only from the feedback about action consequences. Thus, it is reasonable that XCS does not perform as well as UCS in supervised environments.

Two key differences provide UCS with better results in some classification domains: exploration focuses on best action maps and correct fitness pressures towards accuracy. Suggestions to improve XCS's convergence are made, such as using search regimes with more exploitation guidance. Some methods such as those based on ϵ-greedy action-selection or softmax action-selection [20] have already been tested on reinforcement learners. Their introduction to XCS could lead to performance similar to UCS. To avoid the effects of the fitness dilemma in XCS, the use of bilateral accuracy is proposed, as suggested in [6].

The experiments on a noisy problem showed high performance for the LCSs. Both UCS and XCS could achieve optimal populations with a level of noise up to 15%. Future work could enhance this study by analyzing LCSs' tolerance to increasing levels of noise. Our experiments performed in noisy and imbalanced problems showed that UCS was less sensitive to parameter settings. In XCS, two parameters became critical for optimal performance: the learning rate β and the GA triggering threshold θ_{GA}.

Although our results and conclusions are limited to artificial problems, our experimental testbed contained many complexity factors present in real-world problems: multiple classes, noisy instances and imbalanced classes, among others. In this sense, the paper provided some guidelines for further improving learning classifier systems in increasingly challenging problems.

Acknowledgements

The authors thank the support of *Enginyeria i Arquitectura La Salle*, Ramon Llull University, as well as the support of *Ministerio de Ciencia y Tecnología* under project TIN2005-08386-C05-04, and *Generalitat de Catalunya* under Grants 2005FI-00252 and 2005SGR-00302.

References

1. Bernadó-Mansilla, E., Garrell, J.M.: Accuracy-Based Learning Classifier Systems: Models, Analysis and Applications to Classification Tasks. Evolutionary Computation 11(3), 209–238 (2003)
2. Brown, G., Kovacs, T., Marshall, J.A.R.: UCSpv: principled voting in UCS rule populations. In: GECCO 2007, pp. 1774–1781. ACM Press, New York (2007)
3. Bull, L., Hurst, J.: ZCS Redux. Evolutionary Computation 10(2), 185–205 (2002)
4. Butz, M.V., Sastry, K., Goldberg, D.E.: Strong, stable, and reliable fitness pressure in XCS due to tournament selection. Genetic Programming and Evolvable Machines 6(1), 53–77 (2005)
5. Butz, M.V.: Rule-Based Evolutionary Online Learning Systems: A Principled Approach to LCS Analysis and Design. In: Studies in Fuzziness and Soft Computing, vol. 109. Springer, Heidelberg (2006)
6. Butz, M.V., Goldberg, D., Tharankunnel, K.: Analysis and improvement of fitness exploration in XCS: Bounding models, tournament selection, and bilateral accuracy. Evolutionary Computation 11(3), 239–277 (2003)

7. Butz, M.V., Goldberg, D.E., Lanzi, P.L.: Effect of pure error-based fitness in XCS. In: Kovacs, T., Llorà, X., Takadama, K., Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 2003. LNCS (LNAI), vol. 4399, pp. 104–114. Springer, Heidelberg (2007)
8. Butz, M.V., Wilson, S.W.: An algorithmic description of XCS. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 2000. LNCS (LNAI), vol. 1996, pp. 253–272. Springer, Heidelberg (2001)
9. Harik, G.: Finding Multiple Solutions in Problems of Bounded Difficulty. Technical report, IlliGAL Report No. 94002, Urbana-Champaign IL 61801, USA (May 1994)
10. Japkowicz, N., Stephen, S.: The Class Imbalance Problem: A Systematic Study. Intelligent Data Analisis, 6(5), 429–450 (2002)
11. Kovacs, T.: Strength or Accuracy? Fitness Calculation for Classifier Systems. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 1999. LNCS (LNAI), vol. 1813, pp. 143–160. Springer, Heidelberg (2000)
12. Kovacs, T., Kerber, M.: What makes a problem hard for XCS. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 2000. LNCS (LNAI), vol. 1996, pp. 80–99. Springer, Heidelberg (2001)
13. Kovacs, T., Kerber, M.: High Classification Accuracy does not Imply Effective Genetic Search. In: Deb, K., et al. (eds.) GECCO 2004. LNCS, vol. 3103, pp. 785–796. Springer, Heidelberg (2004)
14. Lanzi, P.L.: A Study of the Generalization Capabilities of XCS. In: Bäck, T. (ed.) Proc. of the Seventh Int. Conf. on Genetic Algorithms, pp. 418–425. Morgan Kaufmann, San Francisco (1997)
15. Lanzi, P.L.: Learning Classifier Systems: A Reinforcement Learning Perspective. In: Studies in Fuzziness and Soft Computing, vol. 183, pp. 267–284. Springer, Heidelberg (2005)
16. Orriols-Puig, A., Bernadó-Mansilla, E.: The Class Imbalance Problem in UCS Classifier System: Fitness Adaptation. In: Congress on Evolutionary Computation, Edinburgh, UK, 2-5 September 2005, vol. 1, pp. 604–611. IEEE, Los Alamitos (2005)
17. Orriols-Puig, A., Bernadó-Mansilla, E.: Bounding XCS Parameters for Unbalanced Datasets. In: GECCO 2006, pp. 1561–1568. ACM Press, New York (2006)
18. Orriols-Puig, A., Bernadó-Mansilla, E.: The Class Imbalance Problem in UCS Classifier System: A Preliminary Study. In: Kovacs, T., Llorà, X., Takadama, K., Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 2003. LNCS (LNAI), vol. 4399, pp. 161–180. Springer, Heidelberg (2007)
19. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, San Mateo (1995)
20. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT Press, Cambridge (1998)
21. Vapnik, V.: The Nature of Statistical Learning Theory. Springer, New York (1995)
22. Wilson, S.W.: Classifier Fitness Based on Accuracy. Evolutionary Computation 3(2), 149–175 (1995)
23. Wilson, S.W.: Generalization in the XCS Classifier System. In: Genetic Programming: Proceedings of the Third Annual Conference, pp. 665–674. Morgan Kaufmann, San Francisco (1998)
24. Wilson, S.W.: Classifiers that approximate functions. Journal of Natural Computing 1(2), 211–234 (2002)

A Problem Definitions

In this section, we introduce the problems used in the paper. For each one, a short explanation of its characteristics and the sizes of the best action map $\%[B]$ and the complete action map $\%[O]$ are provided.

A.1 Parity

The parity is a problem that has widely been used as a benchmark in LCS since it was originally introduced in [12] to show the dependence of XCS's performance on the optimal population size. Given a binary string of length ℓ, the number of ones modulo two determines the output. Thus, the problem does not permit any generalization.

The best action map size consists of all the specific rules predicting the correct class, that is, $|[B]| = 2^\ell$. The complete action map doubles the best action map, as it also maintains specific rules predicting the wrong class. Then $|[O]| = 2^{\ell+1}$.

A.2 Decoder

The decoder problem is an artificial problem with binary inputs and multiple classes. Given an input of length ℓ, the output is determined by the decimal value of the binary input. The number of classes increases exponentially with the condition length — $num_{classes} = 2^\ell$. The best action map consists of all possible binary inputs $|[B]| = 2^\ell$ with their corresponding decimal value as output. The complete action map adds ℓ consistently incorrect rules per each consistently correct rule of the best action map. Thus, $|[O]| = 2^\ell \cdot (\ell + 1)$.

A.3 Position

Position is an imbalanced multiclass problem defined as follows. Given a binary-input instance of length ℓ, the output is the position of the left-most one-valued bit.

The best action map consists of $\ell + 1$ rules with different levels of generalization. The complete action map needs to maitain a set of wrong-labeled rules. The size of this set depends on the level of generalization of each class. Table 3 shows the best and the complete action map for position with $\ell = 6$.

A.4 Multiplexer

The multiplexer problem is one of the most used benchmarks in accuracy-based learning classifier systems [22]. The multiplexer is defined for binary strings of size ℓ, where $\ell = k + 2^k$. The first k bits of the conditions are the *position bits*. The output of the multiplexer is the value of the bit referred by the position bits.

Table 3. Best action map (first column) and complete action map (all columns) of position with ℓ=6

000000:0	1#####:0	#1####:0	##1###:0	###1##:0	####1#:0	#####1:0
000001:1	1#####:1	#1####:1	##1###:1	###1##:1	####1#:1	#####0:1
00001#:2	1#####:2	#1####:2	##1###:2	###1##:2	####1#:2	
0001##:3	1#####:3	#1####:3	##1###:3	###0##:3		
001###:4	1#####:4	#1####:4	##0###:4			
01####:5	1#####:5	#0####:5				
1#####:6	0#####:6					

Imbalanced Multiplexer. The imbalanced multiplexer was introduced in [17] to analyze the effects of undersampled classes in XCS. Departing from the original multiplexer problem, the imbalanced multiplexer undersamples one of the classes —labeled as the minority class— in the following way. When required, a new input example is selected randomly. If the example belongs to the class labeled as the majority class, it is given to the system. Otherwise, it is accepted with probability P_{accept}. If it is discarded, a new input example is chosen, which undergoes the same process. Regarding the notation used in the paper, $P_{accept} = \frac{1}{2^i}$.

Multiplexer with Alternating Noise. The multiplexer with alternating noise was firstly used in [5] to show that XCS with tournament selection is able to handle data sets with inconsistent data. The problem is as follows. When a new input instance corresponding to the multiplexer problem is sampled, its action is flipped with probability P_x.

Analysis and Improvements of the Classifier Error Estimate in XCSF

Daniele Loiacono[1], Jan Drugowitsch[2], Alwyn Barry[2], and Pier Luca Lanzi[1,3]

[1] Artificial Intelligence and Robotics Laboratory (AIRLab),
Politecnico di Milano. P.za L. da Vinci 32, I-20133, Milano, Italy
[2] Department of Computer Science, University of Bath, UK
[3] Illinois Genetic Algorithm Laboratory (IlliGAL),
University of Illinois at Urbana Champaign,
Urbana, IL 61801, USA
loiacono@elet.polimi.it, J.Drugowitsch@bath.ac.uk,
A.M.Barry@bath.ac.uk, lanzi@elet.polimi.it

Abstract. The estimation of the classifier error plays a key role in accuracy-based learning classifier systems. In this paper we study the current definition of the classifier error in XCSF and discuss the limitations of the algorithm that is currently used to compute the classifier error estimate from online experience. Subsequently, we introduce a new definition for the classifier error and apply the Bayes Linear Analysis framework to find a more accurate and reliable error estimate. This results in two incremental error estimate update algorithms that we compare empirically to the performance of the currently applied approach. Our results suggest that the new estimation algorithms can improve the generalization capabilities of XCSF, especially when the action-set subsumption operator is used.

1 Introduction

XCS with computed prediction, namely XCSF [12], is a major advance in the field of learning classifier systems. It extends the typical idea of a classifier by replacing the classifier prediction parameter with a prediction function $p(s_t, \mathbf{w})$, that is used to compute the classifier prediction based on the current state s_t and a parameter vector $\mathbf{w}$ associated with each classifier. Since the introduction of XCSF, several studies focused on the classifier weight vector update rule [7,6] and on extending the form of the prediction function p (e.g. see [8]). However, very little work (e.g. see [1]) has concentrated on the classifier error estimate in XCSF, despite its important role in all accuracy-based learning classifier systems. In XCSF the classifier error is usually computed in the same way as in XCS [11]: it is defined as the estimate of the mean absolute prediction error and is updated by the Widrow-Hoff rule (also known as *Least Mean Squared* algorithm). In this paper we suggest the re-definition the classifier error as an estimate of the root mean squared prediction error and propose application of the Bayes Linear Analysis framework for computing the optimal classifier weight vector and the

J. Bacardit et al. (Eds.): IWLCS 2006/2007, LNAI 4998, pp. 117–135, 2008.

classifier error estimate simultaneously. Within such a framework we provide an accurate and reliable estimate of the classifier error. At the same time, we can also provide a better insight into the relationship between the expected error and the statistical knowledge we have about the problem at hand. We prove that the proposed approach, when applied to updating the classifiers in XCSF, (i) computes exactly the same weight vector as the Least Squares update rule introduced in [7], and (ii) provides an unbiased estimate of the root mean squared prediction error. Additionally, by exploiting the similarities to Least Squares, we also introduce a convenient recursive approach for updating the classifiers that combines the Recursive Least Squares weights update [7] and the error tracking algorithm derived in [1]. Finally, we provide an empirical comparison of the classifier error update rules presented in this paper. The experiments have been performed by applying XCSF to the approximation of several commonly used target functions [4]. Our experimental results suggest that the novel classifier error update rules are able to find a more accurate and reliable estimate. In particular, they may improve the generalization capabilities of the XCSF system and allows for using the action-set subsumption operator while preventing overgeneral solutions from taking over the evolved population.

2 The XCSF Classifier System

When compared to XCS, XCSF replaces the classifier scalar prediction parameter by a prediction function $p(\phi(\mathbf{s}_t), \mathbf{w})$ that is parameterised by a parameter vector $\mathbf{w}$. This function computes the prediction as a function of the feature vector $\phi(\mathbf{s}_t)$, extracted from the current sensory input $\mathbf{s}_t$, and the classifier parameter vector $\mathbf{w}$ that replaces the usual scalar prediction parameter; to keep the notation uncluttered, for the rest of this paper we denote $\boldsymbol{\phi}_t \equiv \phi(\mathbf{s}_t)$ as the feature vector that corresponds to the sensory input $\mathbf{s}_t$, and $p(\boldsymbol{\phi}_t) \equiv p(\boldsymbol{\phi}_t, \mathbf{w})$ as the classifier prediction for $\mathbf{s}_t$. Usually, $p(\boldsymbol{\phi}_t, \mathbf{w})$ is computed by the linear combination $p(\boldsymbol{\phi}_t, \mathbf{w}) = \mathbf{w}^T \boldsymbol{\phi}_t$, where the feature vector is given by $\boldsymbol{\phi}^T = [x_0, \mathbf{s}_t(1), \ldots, \mathbf{s}_t(n-1)]^T$, x_0 is a fixed parameter (that is, a constant term), and $n-1$ is the size of the sensory input vectors $\mathbf{s}_t$, so that the feature vectors $\boldsymbol{\phi}_t$ are of size n. Even though it is possible to use non-linear functions to compute the prediction in XCSF [5], in this paper we will exclusively consider the just introduced linear function.

To update the classifiers, at each time step t, XCSF builds a *match set* [M] containing the classifiers in the population [P] whose condition matches the current sensory input $\mathbf{s}_t$. For each action a_i in [M], XCSF computes the *system prediction*, as the fitness-weighted average of the predictions computed by all classifiers in [M] that promote this action. Next, XCSF selects an action to perform. The classifiers in [M] that advocate the selected action form the current *action set* [A]; the selected action is sent to the environment and a reward r_t is returned to the system together with the next input. When XCSF is used as a pure function approximator (like in [12] and in this paper), there is only one dummy action which has no actual effect and the expected payoff is computed

by $P_t = r_t$. The expected payoff P_t is then used to update the weight vector $\mathbf{w}$ of the classifiers in [A] using the Widrow-Hoff rule, also known as the *modified delta rule* [10]. The weight vector, $\mathbf{w}$, of each classifier in [A] is adjusted as follows:

$$\mathbf{w} \leftarrow \mathbf{w} + \frac{\eta \boldsymbol{\phi}_t}{\|\boldsymbol{\phi}_t\|^2} \left(P_t - \mathbf{w}^T \boldsymbol{\phi}_t\right), \tag{1}$$

where η is the correction rate and $\|\boldsymbol{\phi}_t\|^2$ is the squared Euclidean norm of the input vector $\boldsymbol{\phi}_t$ [12]. Then the prediction error, ε, is updated by

$$\varepsilon \leftarrow \varepsilon + \beta \left(|\mathbf{w}^T \boldsymbol{\phi}_t - P_t| - \varepsilon\right). \tag{2}$$

Finally, the classifier fitness is updated as usual [11] and the discovery component is applied as in XCS.

3 Squared Error or Absolute Error?

In XCSF the classifier weight vector is adjusted to minimise the mean squared prediction error (MSE), while the classifier's error is an estimate of the mean absolute prediction error (MAE). Before discussing the consequences of this inconsistency, let us firstly show that this claim is actually correct.

For the rest of this paper we will consider a single classifier and will assume the sequence $t = 1, 2, \ldots$ to represent each time step in which this classifier participates in the action set (making it equivalent to the classifier's *experience*) and thus will be updated.

3.1 Re-deriving the XCSF Weight Vector and Error Update

Let us assume that we have, after t classifier updates, the inputs $\{\mathbf{s}_i\}_{i=1}^t$ and their associated payoffs $\{P_i\}_{i=1}^t$, and that we want to estimate the classifier's weight vector $\mathbf{w}$ that minimises the MSE, given by

$$f_t(\mathbf{w}) = \frac{1}{t} \sum_{i=1}^{t} \left(\mathbf{w}^T \boldsymbol{\phi}_i - P_i\right)^2, \tag{3}$$

where we have again used $\boldsymbol{\phi}_i \equiv \phi(\mathbf{s}_i)$. Applying the modified delta rule (also known as the *Normalised Least Mean Squared* algorithm) to minimise $f_t(\mathbf{w})$ results in the weight vector update equation

$$\mathbf{w}_t = \mathbf{w}_{t-1} + \frac{\eta \boldsymbol{\phi}_t}{\|\boldsymbol{\phi}_t\|^2} \left(P_t - \mathbf{w}_{t-1}^T \boldsymbol{\phi}_t\right), \tag{4}$$

which is equivalent to the Eq. 1 and thus confirms that the XCSF weight vector update indeed aims at minimising the MSE $f_t(\mathbf{w})$.

To get the prediction error, on the other hand, let us assume that we want to estimate the MAE, given by

$$\varepsilon_t = \frac{1}{t} \sum_{i=1}^{t} \left|\mathbf{w}^T \boldsymbol{\phi}_i - P_i\right|. \tag{5}$$

This estimation problem can be reformulated as a least squares problem that minimises

$$g_t^{\mathrm{MAE}}(\mathbf{w}) = \frac{1}{t}\sum_{i=1}^{t}\left(\varepsilon_t - \left|\mathbf{w}^T\boldsymbol{\phi}_i - P_i\right|\right)^2 \tag{6}$$

with respect to ε_t. Solving $\partial g_t^{\mathrm{MAE}}(\mathbf{w})/\partial\varepsilon_t = 0$ for ε_t results in Eq. 5, which confirms that we can indeed estimate ε_t by minimising $g_t^{\mathrm{MAE}}(\mathbf{w})$. Applying the delta rule (also known as the *Least Mean Squared* algorithm) to minimising $g_t^{\mathrm{MAE}}(\mathbf{w})$ results in the prediction error update

$$\varepsilon_t = \varepsilon_{t-1} + \beta\left(\left|\mathbf{w}_t^T\boldsymbol{\phi}_t - P_t\right| - \varepsilon_{t-1}\right), \tag{7}$$

where we have approximated the weight vector by its current estimate $\mathbf{w} \approx \mathbf{w}_t$. This update equation is equivalent to Eq. 2, which shows that XCSF estimates the mean absolute prediction error rather than the mean squared prediction error.

Consequently, the performance component of XCSF that estimates the weight vector aims at minimising the MSE, while the discovery component judges the prediction quality of classifiers based on the MAE. This inconsistency is usually not a serious issue because an optimal solution with respect to the MSE is also nearly optimal with respect to the MAE. Moreover, the MAE is superior to the MSE in terms of human readability; that is, while a threshold on the MAE of the classifier prediction can be easily related to the expected accuracy of the evolved approximation, a threshold on the MSE is not easily related to the final approximation accuracy. Unfortunately, it is rather difficult to find estimators that minimise the MAE (for an XCSF-related example see [8]), whilst there are numerous techniques in the literature that provide accurate estimates that minimise the MSE. This is a concern for XCSF, where the prediction error estimate should reflect the actual prediction error. Thus, we propose replacing the MAE estimate by the MSE estimate, as shown in the following section.

3.2 Estimating the Root Mean Squared Error

We can estimate the MSE (Eq. 3) in the same way as the MAE (Eq. 5) by reformulating its estimation as a least squares problem that minimises

$$g_t^{\mathrm{MSE}}(\mathbf{w}) = \frac{1}{t}\sum_{i=1}^{t}\left(\varepsilon_t^2 - (\mathbf{w}^T\boldsymbol{\phi}_i - P_i)^2\right)^2 \tag{8}$$

with respect to ε_t^2, which denotes the estimate of the classifier squared error at time t. Applying the delta rule by again approximating $\mathbf{w}$ by its estimate $\mathbf{w}_t$ gives the update equation

$$\varepsilon_t^2 = \varepsilon_{t-1}^2 + \beta\left((\mathbf{w}_t^T\boldsymbol{\phi}_t - P_t)^2 - \varepsilon_{t-1}^2\right), \tag{9}$$

from which we compute the classifier error by

$$\varepsilon_t = \sqrt{\varepsilon_t^2}. \tag{10}$$

Thus, it is given by the estimated root mean squared error (RMSE) of the prediction. We use the RMSE instead of the MSE because (i) the RMSE is a standard error measure in the machine learning literature and (ii) the RMSE has the same value range as the MAE that is usually used in XCSF.

Relying on the MSE instead of the MAE has the additional advantage that we do not need to estimate it by the delta rule, as in Eq. 9, but can track the solution to $f_t(\mathbf{w})$ directly, as we will show in the following section.

4 Improving the Error Estimate

In previous studies [6,1] the problem of computing the classifier prediction has been presented as a problem of incremental parameter estimation. In the following sections we show that both the optimal classifier weights and the estimate of the classifier error can be computed by applying the Bayes Linear Analysis [3] framework. Within this framework we are not only able to provide a very accurate and reliable estimate of a classifier's squared error but also give additional insight into the relationship between the expected prediction error and the statistical properties of the target payoff and the feature vector. Furthermore, we show how the proposed theoretical framework can be used in practice to extend the classifier update in XCSF. Finally, we discuss how the proposed extension is related to the least squares approach introduced in the literature for computing the classifier weights [7] and for incrementally estimating the classifier error [1].

4.1 The Bayes Linear Analysis

In XCSF, we need to predict, with the highest accuracy possible, the value of the target payoff P on the basis of the observed features vector $\boldsymbol{\phi}$. Assuming a full knowledge on the probability distribution of both P and $\boldsymbol{\phi}$ we might derive the conditional density $p(P|\boldsymbol{\phi})$ of P given $\boldsymbol{\phi}$ and thus compute the classifier prediction as the conditional expectation $E[P|\boldsymbol{\phi}]$. Unfortunately, we usually do not have such knowledge and therefore cannot derive the conditional probability distribution. In XCSF, however, we limit our search for a suitable prediction model to the linear function $\mathbf{w}^T\boldsymbol{\phi}$ (see Section 2). This assumption allows us to apply Bayes Linear Analysis [3] to compute the classifier weights and errors. Accordingly, the classifier weight vector $\mathbf{w}$ is considered optimal if it solves the following minimisation problem,

$$\min_{\mathbf{w}} E[(P - \mathbf{w}^T\boldsymbol{\phi}(\mathbf{s}))^2], \tag{11}$$

which corresponds (see Appendix A for the derivation) to the classifier weight vector $\mathbf{w}$, defined as,

$$\mathbf{w} = E\left[\boldsymbol{\phi}\boldsymbol{\phi}^T\right]^{-1} E\left[P\boldsymbol{\phi}\right]. \tag{12}$$

By substituting $\mathbf{w}$ by Eq. 12 into the minimisation objective Eq. 11, we get the following classifier squared prediction error estimate (see Appendix A for the derivation):

$$\varepsilon^2 = E\left[(P - \mathbf{w}^T\boldsymbol{\phi})^2\right] = E\left[P^2\right] - E\left[P\boldsymbol{\phi}^T\right] E\left[\boldsymbol{\phi}\boldsymbol{\phi}^T\right]^{-1} E\left[P\boldsymbol{\phi}\right]. \tag{13}$$

Before showing how Eqs. 12 and 13 can be used in practice to update the classifiers in XCSF, it is worthwhile to discuss in more details the consequences of Eq. 13. First of all, given that in XCSF we have $\boldsymbol{\phi}^T = [x_0 \quad \mathbf{s}^T]$, Eq. 13 can be rewritten (see Appendix A for the derivation) as follows,

$$\varepsilon^2 = \mathrm{cov}(P, P)(1 - \rho^2), \tag{14}$$

where $\mathrm{cov}(P, P) = E\left[(P - E\left[P\right])^2\right]$ and ρ^2 is the squared correlation coefficient between P and $\mathbf{s}$, given by,

$$\rho^2 = \frac{\mathrm{cov}(P, \mathbf{s})^T \mathrm{cov}(\mathbf{s}, \mathbf{s})^{-1} \mathrm{cov}(P, \mathbf{s})}{\mathrm{cov}(P, P)} \tag{15}$$

where we have

$$\begin{aligned} \mathrm{cov}(P, \mathbf{s}) &= E\left[(P - E\left[P\right])(\mathbf{s} - E\left[\mathbf{s}\right])\right], \\ \mathrm{cov}(\mathbf{s}, \mathbf{s}) &= E\left[(\mathbf{s} - E\left[\mathbf{s}\right])(\mathbf{s} - E\left[\mathbf{s}\right])^T\right]. \end{aligned}$$

Equation 14 offers an interesting insight on the expected classifier prediction error. When P and $\mathbf{s}$ are completely uncorrelated, i.e. $\rho^2 = 0$, it is not possible to provide any prediction of the target payoff better than its expected value; therefore the expected square prediction error is equal to the variance of P. On the other hand, when P and $\mathbf{s}$ are maximally correlated, i.e. $\rho^2 = 1$, the target payoff can be predicted without error; therefore the expected square prediction error is equal to 0. In all the other cases, the higher the correlation between P and $\mathbf{s}$, the more accurate is the target payoff prediction and, therefore, the lower is the expected square prediction error.

4.2 A Sample-Based Implementation and Its Relation to Least Squares

So far we have assumed knowledge of $E\left[PP\right]$, $E\left[P\boldsymbol{\phi}\right]$ and $E\left[\boldsymbol{\phi}\boldsymbol{\phi}\right]$. Unfortunately such knowledge is not available and thus we cannot directly use Eqs. 12 and 13 in XCSF. For this reason we propose to replace the true expectations with their sample-based estimators, computed at each time step t as,

$$E_{PP} \approx \hat{E}_{PP,t} = \frac{1}{t}\sum_{i=1}^{t} P_i^2 = \hat{E}_{PP,t-1} + \frac{1}{t}(P_t^2 - \hat{E}_{PP,t-1}), \tag{16}$$

$$E_{\boldsymbol{\phi}\boldsymbol{\phi}} \approx \hat{E}_{\boldsymbol{\phi}\boldsymbol{\phi},t} = \frac{1}{t}\sum_{i=1}^{t} \boldsymbol{\phi}_i\boldsymbol{\phi}_i^T = \hat{E}_{\boldsymbol{\phi}\boldsymbol{\phi},t-1} + \frac{1}{t}(\boldsymbol{\phi}_t\boldsymbol{\phi}_t^T - \hat{E}_{\boldsymbol{\phi}\boldsymbol{\phi},t-1}), \tag{17}$$

$$E_{P\boldsymbol{\phi}} \approx \hat{E}_{P\boldsymbol{\phi},t} = \frac{1}{t}\sum_{i=1}^{t} P_i\boldsymbol{\phi}_i = \hat{E}_{P\boldsymbol{\phi},t-1} + \frac{1}{t}(P_i\boldsymbol{\phi}_t - \hat{E}_{P\boldsymbol{\phi},t-1}). \tag{18}$$

Using the above approximations in Eqs. 12 and 13, we obtain the following update rules that can be used for computing the classifier weights and error in XCSF:

$$\mathbf{w}_t = \hat{E}^{-1}_{\boldsymbol{\phi}\boldsymbol{\phi},t}\hat{E}_{P\boldsymbol{\phi},t}, \tag{19}$$

$$\varepsilon_t^2 = \hat{E}_{PP,t} - \hat{E}^T_{P\boldsymbol{\phi},t}\hat{E}^{-1}_{\boldsymbol{\phi}\boldsymbol{\phi},t}\hat{E}^T_{P\boldsymbol{\phi},t}. \tag{20}$$

Notice that the above update rules are more accurate than the usual Widrow-Hoff rule (Eqs. 1 and 9) in finding the optimal classifier weights and error estimates. On the downside, they are computationally more expensive: due to the computation of matrix $\hat{E}^{-1}_{\boldsymbol{\phi}\boldsymbol{\phi}}$ both update rules have a time complexity of $O(n^3)$, whilst the Widrow-Hoff has a time complexity of $O(n)$. Furthermore, it is necessary to store, for each classifier, the sample-based estimators used in Eqs. 19 and 20 with an additionally memory overhead of $O(n^2)$.

To reduce the time complexity of the above update equations it is useful to note that, using the sample-based estimators introduced before, it can be shown (see Appendix B) that Eq. 12 is equivalent to the Least Squares update rule that was introduced in [7] for computing the classifier weights. Additionally, the classifier error computed by Eq. 20 is equivalent to the sample-based estimator of the mean square prediction error (see Appendix B), given by $f_t(\mathbf{w})$ in Eq. 3. In the following section we show how this knowledge can be exploited to derive more efficient update equations.

4.3 Recursive Least Squares and Error Tracking

We have shown that by applying Bayes Linear Analysis we can effectively compute both the classifier weight vector and the classifier error. Unfortunately, as already mentioned before, Eqs. 19 and 20 are computationally expensive. This is a serious drawback because in XCSF the classifiers are updated incrementally and frequently. However this is a well known limitation of the Least Squares update, that is computed by Eq. 19. Thus, following the same procedure as in [7], we can instead use the less costly Recursive Least Squares algorithms (see Appendix C for more details) to incrementally update the classifier weights by

$$\beta^{\mathrm{RLS}} = 1 + \boldsymbol{\phi}_t^T\mathbf{V}_{t-1}\boldsymbol{\phi}_t, \tag{21}$$

$$\mathbf{V}_t = \mathbf{V}_{t-1} - \frac{1}{\beta^{\mathrm{RLS}}}\mathbf{V}_{t-1}\boldsymbol{\phi}_t^T\boldsymbol{\phi}_t\mathbf{V}_{t-1}, \tag{22}$$

$$\mathbf{w}_t = \mathbf{w}_{t-1} + \mathbf{V}_t\boldsymbol{\phi}_t(P - \mathbf{w}_{t-1}^T\boldsymbol{\phi}_t), \tag{23}$$

where $\mathbf{V}_t$ is an estimate of the feature vector autocorrelation matrix. Note that the above update equations avoid the computation of the inverse matrix at each time and therefore have the lower computational complexity of $O(n^2)$. On the other hand, each classifier still need to store the actual $\mathbf{V}$ matrix with an additional memory overhead of $O(n^2)$.

Table 1. Target functions used to compare the performance of XCSFrls, XCSFb and XCSFrb ($x \in [0, 1]$)

$$
\begin{aligned}
f_p(x) &= 1 + x + x^2 + x^3, \\
f_{abs}(x) &= |\sin(2\pi x) + |\cos(2\pi x)||\,, \\
f_{s3}(x) &= \sin(2\pi x) + \sin(4\pi x) + \sin(6\pi x), \\
f_{s4}(x) &= \sin(2\pi x) + \sin(4\pi x) + \sin(6\pi x) + sin(8\pi x).
\end{aligned}
$$

Regarding the classifier error update, the solution to Eq. 3 can be tracked by using the following recursive update (see [1] and Appendix D for more details):

$$\varepsilon_t^2 = \varepsilon_{t-1}^2 + \frac{1}{t}\left((P_t - \mathbf{w}_{t-1}^T \boldsymbol{\phi}_t)(P_t - \mathbf{w}_t^T \boldsymbol{\phi}_t) - \varepsilon_{t-1}^2\right), \tag{24}$$

where $\mathbf{w}_{t-1}$ and $\mathbf{w}_t$ are respectively the classifier weight vectors before and after the update. Note that Eq. 24 is as accurate as Eq. 13 in tracking the classifier's squared prediction error estimate, but has the same complexity as Eq. 9, that is $O(n)$ in time.

5 Experimental Design

All the experiments discussed in this paper aim at comparing the performance of the different classifier error updates introduced in the previous sections. For this purpose we use three different versions of XCSF: (i) XCS with RLS prediction, briefly XCSFrls, that uses RLS (Eqs. 21, 22 and 23) to update classifier weights, and the commonly applied Widrow-Hoff (Eq. 9) to update the classifier error estimate; (ii) XCS with Bayes Linear prediction, briefly XCSFb that updates the classifier weight and error estimate by Eqs. 19 and 20; (iii) XCS with recursive Bayes Linear prediction, briefly XCSFrb, that applies the RLS algorithm (as XCSFrls) to update the classifier weights, and uses Eq. 24 to track the classifier error estimate. Note that in all the XCSF variants the classifier error is defined as an estimate of the RMSE of the prediction, for the reasons discussed in Section 3.

The experimental analysis has been performed on several function approximation tasks, following the standard design used in the literature [12]. As target functions we used the four functions reported in Table 1 that are a real valued version of the ones used in [4].

In all the experiments performed in this work the feature vector is defined as $\boldsymbol{\phi} = [x_0 \quad x]^T$, with x_0 set to 1. The performance is measured as the accuracy of the evolved approximation $\hat{f}(x)$ with respect to the target function $f(x)$, evaluated, in each experiment, as the *root mean square error* (RMSE) given by

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(f(x_i) - \hat{f}(x_i))^2},$$

where $\{x_1, \cdots, x_N\}$ are the N input samples used to evaluate the approximation error. In practice we considered the average RMSE, dubbed $\overline{RMSE}$, over all

experimental runs. To measure the generalization capabilities we considered both the number of macroclassifiers evolved and the fitness-weighted generality of the evolved populations, where each classifier generality is computed as the expected fraction of inputs matched according to [4].

Statistical Analysis. To analyze the results reported in this paper, we performed an analysis of variance (ANOVA) [2] on the resulting performances, evolved population size and generality. For each experiment and for each setting, we analyzed the final performance, the number of macroclassifiers evolved, and their fitness-weighted average generality for the different versions of XCSF; we applied the analysis of variance to test whether there was some statistically significant difference; in addition, we applied four *post-hoc tests* [2], Tukey HSD, Scheffé, Bonferroni, and Student-Neumann-Keuls, to find which XCSF variants performed significantly different.

6 Experimental Results

The experimental analysis is organized as follows. At first we study the different update rules using a single classifiers for approximating a target function. Then we compare XCSFrls, XCSFrb, and XCSFb on several function approximation problems without using the action-set subsumption operator (Section 6.2) and using it (Section 6.3).

6.1 Single Classifier Error

In the first experiment we compare the classifier error updates in XCSFrls, XCSFb and in XCSFrb. For this purpose we focus on the error updates of a single classifier approximating f_{abs} for $x \in [0.4, 0.6]$. Figure 1 shows the classifier error estimate of a single run of the same classifier, as computed by XCSFrls, XCSFb, and XCSFrb when applied to f_{abs}. As reference, we also report the true error of the classifier, computed at the end of the experiment. Note that, although all the three error estimates are about the same on the average, the estimate computed by XCSFrls is very noisy and therefore not particularly reliable. On the other hand, both XCSFb and XCSFrb compute a very reliable and accurate classifier error estimate. Also, the estimates of XCSFrb and XCSFb initially differ slightly due to the bias induced by the initialization of $\mathbf{V}$ in XCSFrb (see Appendices C and D), but they converge very quickly to the same estimate. In conclusion, notice that the reliability of the error estimate of XCSFrls might be improved using a smaller value of the learing rate β; on the other hand, the smaller β the slower the convergence of the error estimate toward the true error. In all the experiments in the rest of the paper we always set $\beta = 0.2$ because tuning the value of β is tricky and the best value is, in general, problem dependent.

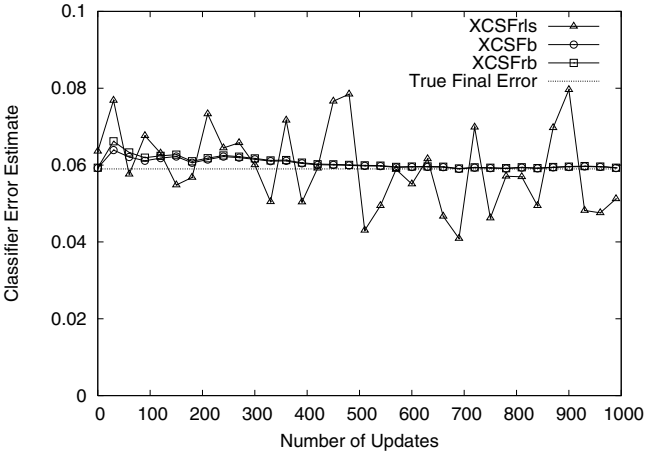


Fig. 1. Comparison of the error updates of a single run of XCSFrls, XCSFb, and XCSFrb applied to f_{abs}. The reported error estimates are that of a single classifier that matches all the inputs in the range $[0.4, 0.6]$.

6.2 Analysis of Generalization

In the second set of experiments we apply XCSFrls, XCSFb, and XCSFrb to all four functions in Table 1, using the following parameters setting: $N = 400$; $\beta = 0.2$; $\alpha = 0.1$; $\nu = 5$; $\chi = 0.8$, $\mu = 0.04$, $\theta_{del} = 25$, $\theta_{GA} = 25$, and $\delta = 0.1$; GA-subsumption is on with $\theta_{GAsub} = 25$; action-set subsumption is not used; $m_0 = 0.2$, $r_0 = 0.1$ [12]; in XCSFrls and XCSFrb we set $\delta_{rls} = 10000$ [7]; the value of ϵ_0 is set to either 0.05, 0.1 or 0.2. Table 2 reports the performance of XCSFrls, XCSFb, and XCSFrb, measured by the average RMSE of the evolved solutions

Table 2. Performance of XCSFrls, XCSFb, and XCSFrb applied to f_p, f_{s3}, f_{s4}, and f_{abs}. The action-set subsumption is not used. Statistics are averages over 50 runs.

$f(x)$	$\epsilon 0$	`XCSFrls`	`XCSFrb`	`XCSFb`
f_p	0.05	0.0111 ± 0.0014	0.0173 ± 0.0024	0.0172 ± 0.0027
f_p	0.10	0.0206 ± 0.0026	0.0289 ± 0.0032	0.0300 ± 0.0032
f_p	0.20	0.0543 ± 0.0070	0.0869 ± 0.0125	0.0869 ± 0.0125
f_{s3}	0.05	0.0300 ± 0.0083	0.0353 ± 0.0027	0.0347 ± 0.0025
f_{s3}	0.10	0.0510 ± 0.0045	0.0633 ± 0.0043	0.0618 ± 0.0032
f_{s3}	0.20	0.0831 ± 0.0075	0.1013 ± 0.0070	0.1024 ± 0.0080
f_{s4}	0.05	0.0321 ± 0.0056	0.0406 ± 0.0066	0.0387 ± 0.0050
f_{s4}	0.10	0.0547 ± 0.0048	0.0676 ± 0.0061	0.0669 ± 0.0041
f_{s4}	0.20	0.0863 ± 0.0069	0.1105 ± 0.0070	0.1132 ± 0.0084
f_{abs}	0.05	0.0190 ± 0.0020	0.0242 ± 0.0025	0.0243 ± 0.0026
f_{abs}	0.10	0.0349 ± 0.0027	0.0482 ± 0.0034	0.0481 ± 0.0031
f_{abs}	0.20	0.0867 ± 0.0048	0.1191 ± 0.0051	0.1208 ± 0.0052

over 50 runs. The results show that all three XCSF versions are accurate in that the final approximation error is lower than ϵ_0. We can also see that the error of XCSFrls is generally lower than the one of XCSFb and XCSFrb. This comes hand in hand both with a larger evolved population and with a lower average generality of the classifiers, as shown in Tables 3 and 4. These results are not surprising as a more reliable classifier error estimate can be expected to improve the classifier system's generalization capabilities, as confirmed by the outcome of this experiment.

The statistical analysis of the data reported in Tables 2, 3 and 4 reveals that the differences between XCSFrb and XCSFb are always *not significant* with a 99.9% confidence. A further analysis of Table 2 shows that the differences between XCSFrls, XCSFrb, and XCSFb are *significant* with a 99.9% confidence for almost all of the experiments, with the exception of cases where complex function are to be estimated with a low error, which prohibits generalization (for example, f_{s3} and f_{s4} with $\epsilon_0 = 0.05$). With respect to the population size (given in Table 3) the statistical analysis indicates that the differences are not always significant, especially on the most complex functions f_{s3} and f_{s4}. Concerning the fitness-weighted average generality, as reported in Table 4, the post-hoc analysis shows that the differences between XCSFrb and XCSFb are always *not significant* with a 99.9% confidence level, while the differences between XCSFb, XCSFrb, and XCSFrls are always significant.

In summary, the results suggest that improving the accuracy and the reliability of the classifier error estimate with the approaches introduced in Sections 4.2 and 4.3 allows XCSF to evolve more general and slightly more compact solutions (even if the size of the populations evolved by XCSFb and XCSFrb are not always significantly smaller than the ones evolved by XCSFrls).

Table 3. Average number of macroclassifiers evolved by XCSFrls, XCSFb, and XCSFrb applied to f_p, f_{s3}, f_{s4}, and f_{abs}. The action-set subsumption is not used. Statistics are averages over 50 runs.

$f(x)$	$\epsilon 0$	XCSFrls	XCSFrb	XCSFb
f_p	0.05	37.6200 ± 3.4922	33.0400 ± 3.6876	32.8200 ± 2.9509
f_p	0.10	33.6000 ± 3.3226	29.5000 ± 2.9883	29.1400 ± 3.0200
f_p	0.20	29.3600 ± 3.2970	27.0800 ± 3.2609	27.0800 ± 3.2609
f_{s3}	0.05	52.7600 ± 4.3523	52.0400 ± 4.1807	51.3200 ± 4.7516
f_{s3}	0.10	48.9400 ± 4.2020	45.9400 ± 4.1493	47.5800 ± 4.9883
f_{s3}	0.20	45.2800 ± 3.6989	42.5000 ± 3.1953	42.8000 ± 3.0331
f_{s4}	0.05	54.4200 ± 4.6047	52.8800 ± 4.4973	54.0000 ± 5.0794
f_{s4}	0.10	52.5800 ± 4.7248	50.0800 ± 4.0686	50.0800 ± 4.8820
f_{s4}	0.20	50.0400 ± 4.0594	47.1000 ± 3.6455	47.9200 ± 3.8773
f_{abs}	0.05	45.3600 ± 3.9180	44.1000 ± 4.4419	42.9000 ± 4.0062
f_{abs}	0.10	44.6800 ± 2.7162	41.6800 ± 3.8494	41.6800 ± 3.3849
f_{abs}	0.20	40.7800 ± 3.9104	35.6400 ± 3.3031	35.8600 ± 3.5497

Table 4. Average generality of classifiers evolved by XCSFrls, XCSFb, and XCSFrb applied to f_p, f_{s3}, f_{s4}, and f_{abs}. The action-set subsumption is not used. Statistics are averages over 50 runs.

$f(x)$	$\epsilon 0$	XCSFrls	XCSFrb	XCSFb
f_p	0.05	0.2916 ± 0.1462	0.3272 ± 0.1693	0.3265 ± 0.1719
f_p	0.10	0.3389 ± 0.2106	0.3614 ± 0.2418	0.3649 ± 0.2430
f_p	0.20	0.3951 ± 0.2928	0.4361 ± 0.3288	0.4361 ± 0.3288
f_{s3}	0.05	0.0719 ± 0.0354	0.0819 ± 0.0405	0.0824 ± 0.0404
f_{s3}	0.10	0.1032 ± 0.0452	0.1155 ± 0.0476	0.1145 ± 0.0478
f_{s3}	0.20	0.1342 ± 0.0503	0.1454 ± 0.0570	0.1457 ± 0.0576
f_{s4}	0.05	0.0556 ± 0.0294	0.0635 ± 0.0329	0.0634 ± 0.0329
f_{s4}	0.10	0.0797 ± 0.0369	0.0892 ± 0.0387	0.0898 ± 0.0384
f_{s4}	0.20	0.1039 ± 0.0430	0.1172 ± 0.0458	0.1170 ± 0.0449
f_{abs}	0.05	0.1255 ± 0.0303	0.1340 ± 0.0363	0.1345 ± 0.0353
f_{abs}	0.10	0.1498 ± 0.0520	0.1637 ± 0.0619	0.1635 ± 0.0622
f_{abs}	0.20	0.2052 ± 0.1264	0.2480 ± 0.1445	0.2518 ± 0.1437

6.3 Classifier Error and Action-Set Subsumption

The action-set subsumption operator [11] is a powerful mechanism to improve the generalization capabilities of XCS. In practice, action-set subsumption is rarely used in XCSF [12,7]. In fact, the action-set subsumption relies heavily on the correctness of the classifier error estimate in order to identify accurate classifiers, and in XCSF this can easily result in evolving overgeneral solutions. This is mainly due to the noisy classifier error estimate computed by XCSF as shown in Figure 1. Thus,

Table 5. Performance of XCSFrls, XCSFb, and XCSFrb applied to f_p, f_{s3}, f_{s4}, and f_{abs}. The action-set subsumption is used with $\theta_{ASsub} = 25$. Statistics are averages over 50 runs.

$f(x)$	$\epsilon 0$	XCSFrls	XCSFrb	XCSFb
f_p	0.05	0.0655 ± 0.0080	0.0419 ± 0.0051	0.0429 ± 0.0049
f_p	0.10	0.1799 ± 0.0078	0.0834 ± 0.0134	0.0816 ± 0.0129
f_p	0.20	0.1863 ± 0.0001	0.1863 ± 0.0001	0.1862 ± 0.0001
f_{s3}	0.05	0.0550 ± 0.0107	0.0461 ± 0.0170	0.0495 ± 0.0295
f_{s3}	0.10	0.1027 ± 0.0230	0.0771 ± 0.0071	0.0794 ± 0.0092
f_{s3}	0.20	0.2314 ± 0.0297	0.1451 ± 0.0112	0.1439 ± 0.0100
f_{s4}	0.05	0.0609 ± 0.0221	0.0512 ± 0.0217	0.0470 ± 0.0109
f_{s4}	0.10	0.1024 ± 0.0108	0.0801 ± 0.0090	0.0844 ± 0.0335
f_{s4}	0.20	0.2246 ± 0.0277	0.1493 ± 0.0116	0.1482 ± 0.0120
f_{abs}	0.05	0.0527 ± 0.0074	0.0361 ± 0.0060	0.0357 ± 0.0046
f_{abs}	0.10	0.1343 ± 0.0235	0.0859 ± 0.0075	0.0825 ± 0.0076
f_{abs}	0.20	0.2899 ± 0.0002	0.1725 ± 0.0279	0.1661 ± 0.0182

Table 6. Average number of macroclassifiers evolved by XCSFrls, XCSFb, and XCSFrb applied to f_p, f_{s3}, f_{s4}, and f_{abs}. The action-set subsumption is used $\theta_{ASsub} = 25$. Statistics are averages over 50 runs.

$f(x)$	$\epsilon 0$	XCSFrls	XCSFrb	XCSFb
f_p	0.05	10.7800 ± 2.9277	7.8600 ± 2.5535	7.3800 ± 2.6449
f_p	0.10	4.5200 ± 1.8027	7.8400 ± 3.7966	7.9000 ± 2.9682
f_p	0.20	2.6000 ± 1.2329	2.1800 ± 1.0713	2.5200 ± 1.2528
f_{s3}	0.05	27.6400 ± 3.5820	29.8600 ± 3.7041	30.4000 ± 3.8158
f_{s3}	0.10	20.5200 ± 4.2391	20.5400 ± 3.5565	20.7800 ± 3.0678
f_{s3}	0.20	16.2800 ± 3.7151	14.6800 ± 3.5068	13.3600 ± 3.0382
f_{s4}	0.05	33.1000 ± 4.1049	38.1800 ± 3.4333	38.2800 ± 5.1109
f_{s4}	0.10	25.9400 ± 4.0812	27.5200 ± 4.1916	27.1600 ± 3.9767
f_{s4}	0.20	20.4000 ± 4.8497	17.7200 ± 2.9465	19.5200 ± 2.9205
f_{abs}	0.05	16.4800 ± 3.3301	16.3400 ± 3.4328	16.5000 ± 2.7514
f_{abs}	0.10	12.3600 ± 2.8549	15.5200 ± 4.4777	15.5800 ± 3.4761
f_{abs}	0.20	2.6600 ± 1.3800	8.8800 ± 3.3205	10.0400 ± 3.9036

Table 7. Average generality of classifiers evolved by XCSFrls, XCSFb, and XCSFrb applied to f_p, f_{s3}, f_{s4}, and f_{abs}. The action-set subsumption is used with $\theta_{ASsub} = 25$. Statistics are averages over 50 runs.

$f(x)$	$\epsilon 0$	XCSFrls	XCSFrb	XCSFb
f_p	0.05	0.6833 ± 0.0820	0.5109 ± 0.0896	0.5149 ± 0.0846
f_p	0.10	0.9861 ± 0.0739	0.7375 ± 0.0846	0.7301 ± 0.0847
f_p	0.20	0.9960 ± 0.0635	0.9971 ± 0.0542	0.9961 ± 0.0619
f_{s3}	0.05	0.0987 ± 0.0487	0.0842 ± 0.0403	0.0843 ± 0.0406
f_{s3}	0.10	0.1437 ± 0.0619	0.1231 ± 0.0506	0.1216 ± 0.0517
f_{s3}	0.20	0.2202 ± 0.1164	0.1732 ± 0.0650	0.1776 ± 0.0638
f_{s4}	0.05	0.0759 ± 0.0398	0.0637 ± 0.0331	0.0638 ± 0.0335
f_{s4}	0.10	0.1094 ± 0.0500	0.0927 ± 0.0413	0.0914 ± 0.0411
f_{s4}	0.20	0.1631 ± 0.0845	0.1338 ± 0.0510	0.1331 ± 0.0517
f_{abs}	0.05	0.1772 ± 0.0465	0.1506 ± 0.0339	0.1491 ± 0.0348
f_{abs}	0.10	0.3034 ± 0.1619	0.2294 ± 0.1049	0.2258 ± 0.1021
f_{abs}	0.20	0.9959 ± 0.0639	0.3469 ± 0.1839	0.3377 ± 0.1683

in the last set of experiments we test whether the new classifier error updates can improve the performance of XCSF when action-set subsumption is used.

We again apply XCSFrls, XCSFb, and XCSFrb to the four functions in Table 1, using the same parameters setting as in the previous experiment, except for the action-set subsumption that is now active with $\theta_{ASsub} = 25$. The performance of XCSFrls, XCSFb, and XCSFrb is reported in Table 5 computed as the average RMSE of the evolved solutions over 50 runs. The results show that XCSFb and XCSFrb are always able to evolve accurate solutions while the solutions evolved by XCSFrls are never accurate except for the simplest function,

f_p, with the highest error threshold, $\epsilon_0 = 0.2$, that allows the evolution of a completely general solution. As we expected, the results suggest that in XCSFrls the action-set subsumption operator may have disruptive effects on the evolved population by considering overgeneral classifiers to be accurate. On the other hand, the more reliable error estimates used in XCSFrb and in XCSFb avoid such a problem. The statistical analysis of the data reported in Table 5 reveals that the differences between XCSFrb and XCSFb are always *not significant* with a 99.9% confidence. A further analysis of Table 5 shows that the differences between XCSFrls and the variants XCSFrb and XCSFb are *significant* with a 99.9% confidence for all the experiments except when the function is of low complexity and generalization is straightforward (e.g. f_p with $\epsilon_0 = 0.2$).

The analysis of the size and generality makes sense only if the the evolved population is accurate. For this reason we have only analyzed the results of XCSFrb and XCSFb, as XCSFrls is almost never accurate. The statistical analysis of the data reported in Table 6 shows that the differences between XCSFrb and XCSFb are always *not significant* with a 99.9% confidence. On the other hand, the same analysis applied to the data in Table 7 shows that XCSFrb evolves slightly more general populations than XCSFb and this difference is significant for most of the experiments. In addition, a comparison with the data reported in the previous section (Tables 3 and 4), shows that by using the action-set subsumption operator it is possible to evolve a more compact and general population (differences are always *significant* with a 99.9% confidence), confirming the results obtained by applying the action-set subsumption to XCS [11].

In summary, the experimental results confirm our hypotheses: the classifier error updates used in XCSFb and in XCSFrb offer a more reliable estimate and therefore allow the action-set subsumption to perform as intended. In fact, the populations evolved by XCSFb and XCSFrb are always accurate and they are also significantly smaller and more general than the ones evolved without using action-set subsumption.

7 Conclusions

In this paper we have proposed a new classifier error definition that is not only more consistent with the XCSF performance component but can also be estimated more effectively. For this purpose, we have introduced the Bayes Linear Analysis framework to compute both the optimal classifier weight vector and the classifier error. In particular, within this framework, we have provided an insight into the relationship between the expected classifier error and the statistical properties of the problem variables, that is, the target payoff and the input vector. Additionally, we have provided two update rules for updating the classifier error more accurately. We have also discussed the similarities between the proposed approach and the Least Squares one that was successfully applied to extending XCSF in [7]. Finally, the classifier error update rules presented in this paper have been empirically compared on several function approximation tasks. Our results suggest that the new error updates do not only compute a more

reliable and accurate estimate, but are also able to improve the performance and the generalization capabilities of XCSF. In particular, the new error update rules (i) allow XCSF to evolve a more compact and general population and (ii) prevent XCSF from evolving inaccurate overgeneral approximations when the action-set subsumption operator is used. On the other hand, improving the error estimate with the usual Widrow-Hoff rule requires the tuning of the learning rate parameters and may significantly slow down the error estimate convergence. However, it is still not clear whether a slower convergence may affect the performance in more complex problems than the ones considered here.

In conclusion, it is important to say that the update rules introduced in this paper have been derived assuming that all the past experiences collected by the system are equally important for solving the problem. Unfortunately this does not hold in multistep problems, where recent experience is usually more important. Therefore, the approach introduced here needs to be extended for multistep problems, possibly with some mechanism of recency-weighting of the collected experience.

References

1. Drugowitsch, J., Barry, A.: A formal framework and extensions for function approximation in learning classifier systems. Machine Learning 70(1), 45–88 (2008)
2. Glantz, S.A., Slinker, B.K.: Primer of Applied Regression & Analysis of Variance, 2nd edn. McGraw Hill, New York (2001)
3. Goldstein, M.: Bayes linear analysis. In: Kotz, S., Read, C.B., Banks, D.L. (eds.) Encyclopedia of Statistical Sciences, vol. 3, pp. 29–34. Wiley, New York (1999)
4. Lanzi, P.L., Loiacono, D., Wilson, S.W., Goldberg, D.E.: Extending XCSF beyond linear approximation. In: Genetic and Evolutionary Computation – GECCO-2005. ACM Press, Washington (2005)
5. Lanzi, P.L., Loiacono, D., Wilson, S.W., Goldberg, D.E.: XCS with Computed Prediction for the Learning of Boolean Functions. In: Proceedings of the IEEE Congress on Evolutionary Computation – CEC-2005, Edinburgh, UK. IEEE Computer Society Press, Los Alamitos (2005)
6. Lanzi, P.L., Loiacono, D., Wilson, S.W., Goldberg, D.E.: Prediction update algorithms for XCSF: RLS, kalman filter, and gain adaptation. In: GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation, pp. 1505–1512. ACM Press, New York (2006)
7. Lanzi, P.L., Loiacono, D., Wilson, S.W., Goldberg, D.E.: Generalization in the XCSF classifier system: Analysis, improvement, and extension. Evolutionary Computation 15(2), 133–168 (2007)
8. Loiacono, D., Marelli, A., Lanzi, P.L.: Support vector regression for classifier prediction. In: GECCO 2007: Proceedings of the 9th annual conference on Genetic and evolutionary computation, pp. 1806–1813. ACM Press, New York (2007)
9. Weisstein, E.W.: Sherman-morrison formula. From MathWorld–A Wolfram Web Resource, `http://mathworld.wolfram.com/Sherman-MorrisonFormula.html`
10. Widrow, B., Hoff, M.E.: Neurocomputing: Foundation of Research. In: Adaptive Switching Circuits, pp. 126–134. MIT Press, Cambridge (1988)

11. Wilson, S.W.: Classifier Fitness Based on Accuracy. Evolutionary Computation 3(2), 149–175 (1995)
12. Wilson, S.W.: Classifiers that approximate functions. Journal of Natural Computing 1(2-3), 211–234 (2002)

A Linear Bayes Analysis

Linear Bayes Analysis defines the *optimal* classifier weight vector $\mathbf{w}$ as the one that minimises the objective function

$$\begin{aligned} J &= E\left[(P - \mathbf{w}^T\boldsymbol{\phi})(P - \mathbf{w}^T\boldsymbol{\phi})^T\right] \\ &= E\left[P^2\right] - E\left[P\boldsymbol{\phi}^T\right]\mathbf{w} - \mathbf{w}^T E\left[P\boldsymbol{\phi}\right] + \mathbf{w}^T E\left[\boldsymbol{\phi}\boldsymbol{\phi}^T\right]\mathbf{w}. \end{aligned} \tag{25}$$

Solving $\partial J/\partial \mathbf{w} = 0$ for $\mathbf{w}$ results in

$$\mathbf{w} = E\left[\boldsymbol{\phi}\boldsymbol{\phi}^T\right]^{-1} E\left[P\boldsymbol{\phi}\right]. \tag{26}$$

The classifier's squared error estimate can be computed by substituting Eq. 26 into Eq. 25, resulting in

$$\begin{aligned} \varepsilon^2 = E\left[P^2\right] - E\left[P\boldsymbol{\phi}^T\right] E\left[\boldsymbol{\phi}\boldsymbol{\phi}^T\right]^{-1} E\left[P\boldsymbol{\phi}\right] - \left(E\left[\boldsymbol{\phi}\boldsymbol{\phi}^T\right]^{-1} E\left[P\boldsymbol{\phi}\right]\right)^T E\left[P\boldsymbol{\phi}\right] + \\ + \left(E\left[\boldsymbol{\phi}\boldsymbol{\phi}^T\right]^{-1} E\left[P\boldsymbol{\phi}\right]\right)^T E\left[\boldsymbol{\phi}\boldsymbol{\phi}^T\right] E\left[\boldsymbol{\phi}\boldsymbol{\phi}^T\right]^{-1} E\left[P\boldsymbol{\phi}\right] = \\ = E\left[P^2\right] - E\left[P\boldsymbol{\phi}^T\right] E\left[\boldsymbol{\phi}\boldsymbol{\phi}^T\right]^{-1} E\left[P\boldsymbol{\phi}\right] \end{aligned} \tag{27}$$

Given that in XCSF we have $\boldsymbol{\phi}^T = [x_0 \quad \mathbf{s}^T]$, and decomposing the weight vector into $\mathbf{w}^T = [w_0 \quad \mathbf{w}'^T]$, we can rewrite $\partial J/\partial \mathbf{w} = 0$ as the following coupled equations

$$E\left[w_0 x_0^2 + \mathbf{w}'\mathbf{s}^T\right] = E\left[P\right], \tag{28}$$

$$E\left[w_0 x_0 \mathbf{s} + \mathbf{w}'\mathbf{s}\mathbf{s}^T\right] = E\left[P\mathbf{s}\right], \tag{29}$$

that, when solved for w_0 and $\mathbf{w}'$, result in,

$$\mathbf{w} = \begin{bmatrix} w_0 \\ \mathbf{w}' \end{bmatrix} = \begin{bmatrix} x_0^{-1}(E\left[P\right] - \mathrm{cov}(P,\mathbf{s})\mathrm{cov}(\mathbf{s},\mathbf{s})^{-1} E\left[\mathbf{s}^T\right]) \\ \mathrm{cov}(P,\mathbf{s})\mathrm{cov}(\mathbf{s},\mathbf{s})^{-1} \end{bmatrix}, \tag{30}$$

where we have

$$\begin{aligned} \mathrm{cov}(P,\mathbf{s}) &= E\left[(P - E\left[P\right])(\mathbf{s} - E\left[\mathbf{s}\right])\right], \\ \mathrm{cov}(\mathbf{s},\mathbf{s}) &= E\left[(\mathbf{s} - E\left[\mathbf{s}\right])(\mathbf{s} - E\left[\mathbf{s}\right])^T\right]. \end{aligned}$$

By substituting the above classifier weight vector definition into Eq. 25, we can rewrite the classifier squared error as follows:

$$\varepsilon^2 = \mathrm{cov}(P,P) - \mathrm{cov}(P,\mathbf{s})^T \mathrm{cov}(\mathbf{s},\mathbf{s})^{-1} \mathrm{cov}(P,\mathbf{s}), \tag{31}$$

where $\mathrm{cov}(P,P) = E\left[(P - E\left[P\right])^2\right]$.

B Bayes Linear Analysis and Least Squares

Let $\mathbf{\Phi}_t$ and $\mathbf{\Pi}_t$ denote respectively the feature and payoff matrix, after t updates, and given by

$$\mathbf{\Phi}_t = \begin{bmatrix} \boldsymbol{\phi}_1^T \\ \vdots \\ \boldsymbol{\phi}_t^T \end{bmatrix}, \qquad \mathbf{\Pi}_t = \begin{bmatrix} P_1 \\ \vdots \\ P_t \end{bmatrix}. \tag{32}$$

The Least Squares algorithm find the weight vector $\mathbf{w}_t$ that minimises the square error $\sum_{i=i}^{t}(P_i - \mathbf{w}_t^T \boldsymbol{\phi}_i)^2$, by solving the normal equation

$$\mathbf{\Phi}_t^T \mathbf{\Phi}_t \mathbf{w}_t = \mathbf{\Phi}_t^T \mathbf{\Pi}_t. \tag{33}$$

By definition, we have

$$\mathbf{\Phi}_t^T \mathbf{\Phi}_t = \sum_{i=1}^{t} \boldsymbol{\phi}_i \boldsymbol{\phi}^T, \tag{34}$$

$$\mathbf{\Phi}_t^T \mathbf{\Pi} = \sum_{i=1}^{t} P_i \boldsymbol{\phi}_i. \tag{35}$$

Multiplying the left and the right hand of Eq. 33 by $1/t$ gives together with Eqs. 34 and 34,

$$\frac{1}{t}\sum_{i=1}^{t} \boldsymbol{\phi}_i \boldsymbol{\phi}^T \mathbf{w}_t = \frac{1}{t}\sum_{i=1}^{t} P_i \boldsymbol{\phi}_i, \tag{36}$$

which, according to the definition introduced by Eqs. 16, 17, and 18, can be written as

$$\hat{E}_{\boldsymbol{\phi}\boldsymbol{\phi},t} \mathbf{w}_t = \hat{E}_{P\boldsymbol{\phi},t}, \tag{37}$$

which results in the classifier weight vector update,

$$\mathbf{w}_t = \hat{E}^{-1}_{\boldsymbol{\phi}\boldsymbol{\phi},t} \hat{E}_{P\boldsymbol{\phi},t}. \tag{38}$$

The squared prediction error (minimised by the above equation) is defined by

$$\varepsilon_t^2 = \frac{1}{t}\sum_{i=1}^{t} \left(P_i - \mathbf{w_t}^T \boldsymbol{\phi}_i\right)^2, \tag{39}$$

and can be expanded to

$$\varepsilon_t^2 = \frac{1}{t}\sum_{i=1}^{t} P_i^2 + \mathbf{w_t}^T \left(\frac{1}{t}\sum_{i=1}^{t} \boldsymbol{\phi}_i \boldsymbol{\phi}_i^T\right) \mathbf{w_t} - \mathbf{w_t}^T \left(\frac{2}{t}\sum_{i=1}^{t} \boldsymbol{\phi}_i\right), \tag{40}$$

which, together with Eqs. 16, 17,18, and 38, gives

$$\varepsilon_t^2 = \hat{E}_{PP,t} - \hat{E}^T_{P\boldsymbol{\phi},t} \hat{E}^{-1}_{\boldsymbol{\phi}\boldsymbol{\phi},t} \hat{E}^T_{P\boldsymbol{\phi},t}. \tag{41}$$

C Recursive Least Squares

The Recursive Least Squares (RLS) algorithm allows tracking the weight vector $\mathbf{w}_t$ that minimises the convex cost function

$$\sum_{t=1}^{t} \left(\mathbf{w}_t^T \boldsymbol{\phi}_t - P_t\right)^2 + \frac{1}{\delta^{\mathrm{RLS}}} \|\mathbf{w}_t\|^2, \tag{42}$$

and satisfies the equality

$$\left(\boldsymbol{\Phi}_t^T \boldsymbol{\Phi}_t + \frac{1}{\delta^{\mathrm{RLS}}} \mathbf{I}\right) \mathbf{w}_t = \boldsymbol{\Phi}_t^T \mathbf{P}_t, \tag{43}$$

where $\mathbf{I}$ denotes the identity matrix and δ^{RLS} is a large positive constant. Let $\mathbf{V}_t^{-1} = \boldsymbol{\Phi}^T \boldsymbol{\Phi}_t$ denote the feature autocorrelation matrix estimate, that satisfies the relation

$$\mathbf{V}_t^{-1} = \mathbf{V}_{t-1}^{-1} + \boldsymbol{\phi}_t^T \boldsymbol{\phi}_t, \tag{44}$$

with $\mathbf{V}_0 = \delta^{\mathrm{RLS}} \mathbf{I}$. Similarly, we have

$$\boldsymbol{\Phi}_t^T \mathbf{P}_t = \boldsymbol{\Phi}_{t-1}^T \mathbf{P}_{t-1} + \boldsymbol{\phi}_t P_t, \tag{45}$$

which, together with Eqs. 43 and 44 allows us to derive

$$\mathbf{V}_t^{-1} \mathbf{w}_t = \mathbf{V}_t^{-1} \mathbf{w}_{t-1} + \boldsymbol{\phi}_t (P_t - \mathbf{w}_{t-1}^T \boldsymbol{\phi}_t). \tag{46}$$

Pre-multiplying the above by $\mathbf{V}_t$ results in the RLS weight vector update

$$\mathbf{w}_t = \mathbf{w}_{t-1} + \mathbf{V}_t \boldsymbol{\phi}_t (P_t - \mathbf{w}_{t-1}^T \boldsymbol{\phi}_t). \tag{47}$$

To get the update for $\mathbf{V}$, we apply the Sherman-Morrison formula [9] to Eq. 44, resulting in

$$\mathbf{V}_t = \mathbf{V}_{t-1} - \frac{\mathbf{V}_{t-1} \boldsymbol{\phi}_t \boldsymbol{\phi}_t^T \mathbf{V}_{t-1}}{1 + \boldsymbol{\phi}_t^T \mathbf{V}_{t-1} \boldsymbol{\phi}}, \tag{48}$$

which can be written as

$$\beta^{\mathrm{RLS}} = 1 + \boldsymbol{\phi}_t^T \mathbf{V}_{t-1} \boldsymbol{\phi}_t, \tag{49}$$

$$\mathbf{V}_t = \mathbf{V}_{t-1} - \frac{1}{\beta^{\mathrm{RLS}}} \mathbf{V}_{t-1} \boldsymbol{\phi}_t \boldsymbol{\phi}_t^T \mathbf{V}_{t-1}, \tag{50}$$

and thus results in the final RLS update for $\mathbf{V}$. Note that the Sherman-Morrison formula is only applicable if $\mathbf{V}^{-1}$ is invertible, and thus $\mathbf{V}$ needs to be initialised to $\mathbf{V}_0 = \delta^{\mathrm{RLS}} \mathbf{I}$ with $\delta^{\mathrm{RLS}} < \infty$, such that $\mathbf{V}_0^{-1} = (1/\delta^{\mathrm{RLS}}) \mathbf{I} > 0\mathbf{I}$. This introduces a bias that is kept small by setting δ^{RLS} to a large value.

D Tracking Mean Square Error

Let us assume that the weight vector $\mathbf{w}$ is estimated by the RLS algorithm, initialised with a very large $\delta^{\mathrm{RLS}} \to \infty$, and therefore by Eq. 43 at t satisfies the normal equation

$$\left(\mathbf{\Phi}_t^T \mathbf{\Phi}_t\right) \mathbf{w}_t = \mathbf{\Phi}_t^T \mathbf{P}_t, \tag{51}$$

which can also be written as

$$\mathbf{w}_t^T \mathbf{\Phi}_t^T \left(\mathbf{\Phi}_t \mathbf{w}_t - \mathbf{P}_t\right) = 0. \tag{52}$$

Our aim is to find an incremental update equation for the MSE, $f_t(\mathbf{W}_t)$, that, following Eq. 3, is in matrix notation given by

$$t f_t(\mathbf{w}_t) = \|\mathbf{\Phi}_t \mathbf{w}_t - \mathbf{P}_t\|^2, \tag{53}$$

Using $-\mathbf{P}_t = -\mathbf{\Phi}_t \mathbf{w}_t + (\mathbf{\Phi}_t \mathbf{w}_t - \mathbf{P}_t)$ and Eq. 52, we can derive

$$\begin{aligned} \mathbf{P}_t^T \mathbf{P}_t &= \mathbf{w}_t^T \mathbf{\Phi}_t^T \mathbf{\Phi}_t \mathbf{w}_t - 2\mathbf{w}_t^T \mathbf{\Phi}_t^T (\mathbf{\Phi}_t \mathbf{w}_t - \mathbf{P}_t) + (\mathbf{\Phi}_t \mathbf{w}_t - \mathbf{P}_t)^T (\mathbf{\Phi}_t \mathbf{w}_t - \mathbf{P}_t) \\ &= \mathbf{w}_t^T \mathbf{\Phi}_t^T \mathbf{\Phi}_t \mathbf{w}_t + \|\mathbf{\Phi}_t \mathbf{w}_t - \mathbf{P}_t\|^2, \end{aligned} \tag{54}$$

and thus we can express the sum of squared errors by

$$\|\mathbf{\Phi}_t \mathbf{w}_t - \mathbf{P}_t\|^2 = \mathbf{P}_t^T \mathbf{P}_t - \mathbf{w}_t^T \mathbf{\Phi}_t^T \mathbf{\Phi}_t \mathbf{w}_t. \tag{55}$$

To express $\|\mathbf{\Phi}_t \mathbf{w}_t - \mathbf{P}_t\|^2$ in terms of $\|\mathbf{\Phi}_{t-1} \mathbf{w}_{t-1} - \mathbf{P}_{t-1}\|^2$, we combine Eqs. 44, 45 and 55, and use $\mathbf{V}_t^{-1} \mathbf{w}_t = \mathbf{\Phi}_t^T \mathbf{P}_t$ after Eq. 51 to get

$$\begin{aligned} & \|\mathbf{\Phi}_t \mathbf{w}_t - \mathbf{P}_t\|^2 \\ &= \mathbf{P}_t^T \mathbf{P}_t - \mathbf{w}_t^T \mathbf{\Phi}_t^T \mathbf{\Phi}_t \mathbf{w}_t \\ &= \|\mathbf{\Phi}_{t-1} \mathbf{w}_{t-1} - \mathbf{P}_{t-1}\|^2 + P_t^2 + \mathbf{w}_{t-1}^T \mathbf{V}_{t-1}^{-1} \mathbf{w}_{t-1} - \mathbf{w}_t^T \mathbf{V}_t^{-1} \mathbf{w}_t \\ &= \|\mathbf{\Phi}_{t-1} \mathbf{w}_{t-1} - \mathbf{P}_{t-1}\|^2 + P_t^2 \\ &\quad + \mathbf{w}_{t-1}^T \left(\left(\mathbf{V}_{t-1}^{-1} + \boldsymbol{\phi}_t \boldsymbol{\phi}_t^T\right) \mathbf{w}_t - \boldsymbol{\phi}_t P_t\right) - \mathbf{w}_t^T \left(\mathbf{V}_{t-1}^{-1} \mathbf{w}_{t-1} + \boldsymbol{\phi}_t P_t\right) \\ &= \|\mathbf{\Phi}_{t-1} \mathbf{w}_{t-1} - \mathbf{P}_{t-1}\|^2 + P_t^2 + \mathbf{w}_{t-1}^T \boldsymbol{\phi}_t \boldsymbol{\phi}_t^T \mathbf{w}_t - \mathbf{w}_{t-1}^T \boldsymbol{\phi}_t P_t - \mathbf{w}_t^T \boldsymbol{\phi}_t P_t \\ &= \|\mathbf{\Phi}_{t-1} \mathbf{w}_{t-1} - \mathbf{P}_{t-1}\|^2 + (\mathbf{w}_{t-1}^T \boldsymbol{\phi}_t - P_t)(\mathbf{w}_t^T \boldsymbol{\phi}_t - P_t). \end{aligned}$$

Thus, we get

$$t f_t(\mathbf{w}_t) = (t-1) f_{t-1}(\mathbf{w}_{t-1}) + (\mathbf{w}_{t-1}^T \boldsymbol{\phi}_t - P_t)(\mathbf{w}_t^T \boldsymbol{\phi}_t - P_t), \tag{56}$$

which, using $\varepsilon_t^2 \equiv f_t(\mathbf{w}_t)$, can be rewritten to

$$\varepsilon_t^2 = \varepsilon_{t-1}^2 + \frac{1}{t}\left((\mathbf{w}_{t-1}^T \boldsymbol{\phi}_t - P_t)(\mathbf{w}_t^T \boldsymbol{\phi}_t - P_t) - \varepsilon_{t-1}^2\right). \tag{57}$$

A Learning Classifier System with Mutual-Information-Based Fitness

Robert Elliott Smith and Max Kun Jiang

Department of Computer Science
University College London
London, United Kingdom
robert.elliott.smith@gmail.com,
m.jiang@cs.ucl.ac.uk

Abstract. This paper introduces a new variety of learning classifier system (LCS), called MILCS, which utilizes mutual information as fitness feedback. Unlike most LCSs, MILCS is specifically designed for supervised learning. We present preliminary results, and contrast them to results from XCS. We discuss the explanatory power of the resulting rule sets and introduce a new technique for visualizing explanatory power. Final comments include future directions of this research, including investigations in neural networks and other systems.

Keywords: Evolutionary computation, learning classifier systems, machine learning, information theory, mutual information, supervised learning, protein structure prediction, explanatory power.

1 Introduction

This paper presents a new form of learning classifier system (LCS) 68 that uses mutual information 1213 as its primary fitness feedback, in supervised learning settings. This system is called the mutual information learning classifier system (MILCS, pronounced "my LCS"). In addition to drawing on current LCS research and information theoretic concerns, the system draws on an analogy to cascade correlation neural networks (CCNs) 5 in its design.

The following sections describe these inspirations and the general design of MILCS. Afterwards, we discuss preliminary results, with comparison to XCS. In this comparison, we are concerned not only with accuracy, generalization, and required computational time, but also with explanatory power of the resulting rule sets. Since explanatory power (in essence, the human understandability of machine-learned concepts) is an abstract concept, we introduce a new technique for visualizing results.

The preliminary results presented are promising, and in final comments we discuss future directions for this research. This clearly includes further research into MILCS performance, but also includes extension of the ideas involved to neural networks, and possibly other systems.

J. Bacardit et al. (Eds.): IWLCS 2006/2007, LNAI 4998, pp. 136–153, 2008.

2 LCSs and CCNs: An Analogy

To introduce the ideas in MILCS, it is first useful to consider the similarities and differences between LCSs and CCNs. This is done in several stages in the following subsections.

2.1 Parameter Versus Structural Learning

In general, machine learning approaches can be broken into two components: the learning of continuous-valued parameters of a solution representation, and the learning of discrete connections between elements of the solution representation. The terms "parameter" and "structural" learning are borrowed from the Bayesian Network community 7, where the former is the learning of probabilities of events, conditioned on one another (the probabilities that reside within Bayesian Network nodes) and the latter is the learning of which events are conditioned on which (the links in the Bayesian Network). It is generally acknowledged that the latter is more difficult than the former. Also, the latter has a profound effect on the computational complexity of the former: the number of probabilities one must learn goes up exponentially with the number of links. Moreover, structural learning is also associated with generalization, parsimony, and explanatory power: fewer discrete connections make for a more understandable representation of a solution.

2.2 CCN and Structural Learning

It is also interesting to note that neural networks do not, in general, employ sophisticated techniques for structural learning. There are many notable exceptions, amongst them the techniques employed in CCNs.

A rough outline of the CCN procedure is as follows:

1. Begin with a single layer neural network. Repeat until a measure of convergence is achieved:
    a. Train existing output layer connection weights (parameter learning) to reduce error on training data (supervised learning)
    b. Insert a new hidden layer node, with inputs from all existing inputs and hidden layer nodes in the network (cascaded). Note that the output of this node is not yet connected to the output layer nodes of the network.
    c. Train the input weights of this new node to maximize the absolute correlation of the node's output to the error of the existing network's output on the training cases (supervised learning).
    d. Connect the output of the new node to all output layer nodes.

At first, this may not seem to meet the discrete-optimization criteria for structural learning discussed above. However, note that input layer node weights are often adjusted to values near zero in step c above. In effect, this "turns off" the connection between a hidden layer node and a particular input (or another hidden layer node). Moreover, there are variations of CCN that employ a population of randomly-initialized nodes in step b and c, culling this population to one node in step d. If these

nodes are initialized with less-than-full input connectivity, the discrete optimization of structural learning is clear.

CCN is relatively straightforward algorithm, with the exception of the somewhat mysterious element of step c. One must consider why one would *maximize* the absolute correlation between a node's output and existing network error. Upon consideration, it becomes clear that this step will allow one to cancel out that existing error with the new node, through the weight adjustments in step a.

2.3 CCN and XCS

XCS, perhaps the most popular LCS paradigm, has a notable similarity to CCN. One of XCS's innovations is its use of accuracy, a second order statistic, as fitness. Similarly, CCN attempts to maximize absolute correlation, another second order statistic, in its creation of hidden layer nodes. If one imagines an analogy where XCS rules are like hidden layer nodes 14, there is a clear correspondence.

However, it is not an exact correspondence, leading one to ask why a difference exists. Note that in CCN, correlation to existing error is justified by supervised learning training of the output layer weights, to cancel out that error. No such "cancellation" exists in XCS, since a rule's "output" (action) is not tuned via supervised learning although extensions of XCS, named XCSF 11 and UCS 2 do train their outputs to reduce errors.

2.4 Supervised Versus Reinforcement Learning

XCS grows out of the LCS tradition of reinforcement learning 15. Reinforcement learning is defined by the lack of supervisory feedback that indicates the correct action the system should take. Instead, only "reward" or "punishment" type feedback is available. Since XCS grew from reinforcement learning, it generally does not employ supervised update of actions. Instead, actions are searched for via the GA, or various covering operations.

However, XCS has been applied to many supervised learning problems. In supervised learning problems direct feedback is available that indicates the correct actions for the system (often via solved training cases). CCN exploits this supervision in its update of output layer weights, which justifies its use of correlation to existing error in hidden layer weights.

This suggests the analogy upon which MILCS is built. However, rather than employing correlation, we have employed mutual information (yet another second order statistic). We feel this provides an additional theoretical backing for the system, which is discussed below.

3 The Role of Mutual Information in MILCS

Shannon's fundamental work in information theory 1213 addresses the following concern: given an input signal to a communication channel, how does one maximize the rate of communication, while minimizing error in that communication? This is closely related to Shannon's work on data compression, which considers the maximum compression ratio for lossless representation of data.

Shannon showed that the zero-error maximum communication rate for a channel is given by maximizing the mutual information between the channel's input and output. Maximization of mutual information is accomplished by manipulation of the probabilities of various inputs to the channel, or through the manipulation of the coding of input signals. Since coding is similar to compression, the analogy to lossless compression is clear.

Imagine that the existing error in step c of the CCN procedure is an input signal to a communication channel. In this case, the hidden layer node plays the role of an encoder for signal. Therefore, we find a firm theoretical foundation for using the mutual information as the fitness of this node, through Shannon's theorems.

Another useful analogy is to sensor placement. Imagine that one is set the task of placing temperature sensors in a large space, with the goal of delivering maximum information about the temperature distribution in that space. If one can estimate the probability distribution of temperatures over the space, and one knows the response field of the sensors, one can maximize mutual information between these distributions to optimally place the sensors. This is similar to the placement of the conditions of classifiers (or the receptive fields of neurons).

Mutual information between variables *X* and *Y* is given by:

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log \frac{p(x,y)}{p(x)p(y)}$$
$$= \sum_{y \in Y} \sum_{x \in X} p(x \mid y) p(y) \log \frac{p(x \mid y)}{p(x)} \quad (1)$$

It is useful to examine the terms of this expression. Consider *x* to be the event of an error, and *y* to be the event of a particular rule matching. Thus, $p(x)$ is the distribution of existing error of a system, and $p(y)$ is the distribution of responses (matching or not matching) of a rule. In this case, the first term, $p(x \mid y)$, can be seen as the relevance of the rule's output to existing error. The second term, $p(y)$, is the generality of the rule. The third term, $\log\left(p(x \mid y)/p(x)\right)$, is a comparison of the error when the rule matches to the overall error in the space. This is a sort of specificity term. Thus, the mutual information expression offers a balance of accuracy, generality, and specificity, in an optimal fashion dictated by Shannon's Theorems.

4 The MILCS Process

Given the above considerations, MILCS operates as follows. Note that to conform to the CCN analogy, and articulate all the terms in the sums of equation 1.1, we have given each rule two actions: one for when the rule matches, and one for when the rule does not match. Both are updated via simple supervised learning:

Table 1. Properties of MILCS classifier

Properties	Explanation	Initial value
act	Action when matched	Randomly selected
actNotMatched	Action when not matched	Randomly selected
pre	Prediction value when matched	10.0
preNotMatched	Prediction value when it is not matched	10.0
fit	Fitness value	0.01
noOfTrainingCases	Number of training cases exposed to	0
exp	Number of times in the action set	0
gaIterationTime	The iteration time when last participated in the GA	0
num	Number of the same classifier	1
MI[][]	Mutual Information counters for modelling an empirical probability distribution over the matched/not-matched condition of the removed rule, and the error of the remaining rules. Fitness is calculated from this bivariate probability distribution	0 (2x2 array)

Table 2. MILCS parameters

Parameters	Explanation
MAX_POP_SIZE	Maximum pop size
THETA_GA	Threshold for running GA to generate new classifiers. Once the average gaIterationTime of all classifiers in the action set is >= THETA_GA, the GA is triggered
INIT_POP_SIZE	Initial Pop Size
MATURITY_ACT_SELECT	noOfTrainingCases required to participate in action selection
MATURITY_SUB	noOfTrainingCases required to participate in subsumption
THETA_SUB	If a classifier's exp is > THETA_SUB, it can be a subsumer
MATURITY_DEL	noOfTrainingCases required for a classifier to be deleted
DEL_WINDOW_SIZE	Window size for deletion. Number of iterations the system should look back in the history
THETA_PERFORMANCE	If the system performance accuracy reaches this threshold, and the population size is larger than the maximum size, group deletion is triggered next time
DONT_CARE_PROB	The probability for having #s in the newly generated classifiers
WIN_FREQ	A deletion threshold: if a classifier's winningFrequency falls below this value, it can be deleted. If the system performance is above THETA_PERFORMANCE, a group of classifiers which fall below this value are deleted
WIN_FREQ_INC	If group deletion is triggered, WIN_FREQ is incremented by this value
MAX_WIN_FREQ	Maximum value of the WIN_FREQ

Table 3. MILCS entities and main methods

Syntax	Explanation
pop	Population set. Population of classifiers
mset	Match set. A sub-set of pop which contains classifiers that match the given training case
nmset	Not matched set. A sub-set of pop which contains classifiers that do not match the given training case
aset	Action set. A sub-set of mset or nmset which contains classifiers that has the same actions
action act = actionSelection (classifierSet set)	Select action based on pre and preNot-Matched of the classifiers of the argument set. Only classifiers with noOfTraining-Cases > MATURITY_ACT_SELECT are taken into account
classifierSet newSet = createSet (classifierSet set, trainingCase t)	Create match set or not matched based on the given trainingCase t
classifierSet newSet = createActionSet (classi-fierSet set, action act)	Create action set based on the selected action
updateMI (classifier i, reward r, trainingCase t)	Update the MI counters of classifier i based on its reward and whether it matches the trainingCase t or not
updateWinningFequency (classifier i)	Update the winningFrequency of classifier i if i.noOfTrainingCases >= MATURITY_ACT_SELECT + MATURITY_ DEL
updateActions (classifier i, reward r)	Adjust the act and actNotMatched of classi-fier i based on its reward r and its previous actions
updateFitness (classifier i)	Update the i.fit based on i.MI
moreGeneral (classifier i, classifier j)	Returns true if each bit of the condition of classifier i is either the same as that bit of the condition of classifier j, or a don't care symbol #
subsumes (classifier i, classifier j)	If classifier i subsumes classifier j then j is removed and i.num is incremented
doGA (classifierSet set1, THETA_GA, classi-fierSet set2)	If the average of classifier i.gaIterationTime of classifier set1 >= THETA_GA, run a non-panmicitic GA on set1 and two children classifiers are added to classifierSet set2
resetMI (classifierSet set)	Reset the MI array to 0s for all classifiers of set

Pseudo code of the MILCS process:

```
initialize(pop, INIT_POP_SIZE)
for each random trainingCase t:
  for each classifier i from pop
    removeClassifier(i,pop)
    action act = actionSelection(pop)
    reward r = getReward(act,t)
    updateMI(i, r, t)
        addClassifier(i,pop)
  classifierSet mset = createSet(pop,t)
  classifierSet nmset = createSet(pop,t)
  act = actionSelection(pop)
  r = getReward(act, t)
  for each classifier i from pop
    updateWinningFrequency(i)
  if act is from mset
    classifierSet aset = createActionSet(mset, act)
  else
    classifierSet aset = createActionSet(nmset, act)
  for each classifier i from aset
    updateFitness(i)
    i.exp ++
    for each rule j from the aset
            if i.noOfTrainingCases > MATURITY_SUB &&  j.noOfTrainingCases >
            MATURITY_SUB && i.act == j.actNotMatched && i.exp > THETA_SUB
            && moreGeneral(i,j) && i.pre => j.pre && i.pre > j.preNotMatched &&
            i.preNotMatched => j.preNotMatched && .PreNotMatched > j.pre
       subsumes(i,j)
  doGA(aset, THETA_GA, pop)
  if pop.size > MAX_POP_SIZE
    if pop.previousPerformance > THETA_PERFORMANCE
      for each rule i from pop
              if i.noOfTrainingCases >= MATURITY_ACT_SELECT +
              THETA_DEL && i.winningFrequency < WIN_FREQ
        removeClassifier(i, pop)
      if WIN_FREQ < MAX_WIN_FREQ
        WIN_FREQ += WIN_FREQ_INC
    else
      for each rule i from pop
        if isMin(i.winningFrequency, pop)
          removeClassifier(i,pop)
  resetMI(aset)
  for each classifier i from pop
    updateWinningFrequency(i)
    updateActions(i, r)
```

Note that the removing and adding procedures (second for loop) are there to conform of the CCN analogy that the new hidden layer node is not yet fully connected to the network thus the new rule's future parent should not have any effect on the system.

5 Results

We have tested MILCS on the multiplexer problem and on the coordination number (CN) protein structure prediction problem.

5.1 Multiplexer Problems

We have evaluated results on the 6, 11, and 20 multiplexer problems. In order to show a thorough comparison, we present these results, along with results obtained from XCS (using 4). Three lines appear on each plot: the percentage correct over the past 50 training cases (solid line), the difference between reward and predicted reward over the past 50 training cases (dashed line), and the number of macro-classifiers (unique classifiers) in the population (dash-dotted line) divided by 1000. Graphs reflect the average of 10 runs.

For XCS, we employ the parameter settings reported in 17. MILCS parameter settings are shown in Table 4.

Table 4. MILCS parameters for the multiplexer problems

Multiplexer problem size	6	11	20
MAX_POP_SIZE	600	1500	2000
THETA_GA	25	25	25
DONT_CARE_PROB	0.33	0.33	0.33
INIT_POP_SIZE	500	500	500
MATURITY_ACT_SELECT	32	128	1024
MATURITY_SUB	64	256	2048
THETA_SUB	70	70	70
MATURITY_DEL	32	128	1024
DEL_WINDOW_SIZE	128	2048	5012
WIN_FREQ	0.0156	0.0005	0.00039
WIN_FREQ_INC	0.00001	0.00001	0.00003
MAX_WIN_FREQ	0.02	0.02	0.02

Figure 1 and Figure 2 show results from XCS and MILCS applied to the 6 multiplexer. Note that MILCS converges more rapidly, and to a smaller final population of unique classifiers.

Figure 3 and Figure 4 show results from XCS and MILCS applied to the 11 multiplexer. While convergence times are similar, MILCS still converges to a smaller final population of unique classifiers.

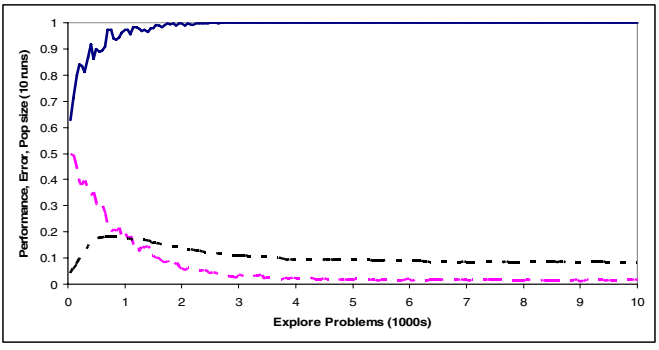


Fig. 1. Results from XCS applied to the 6 multiplexer

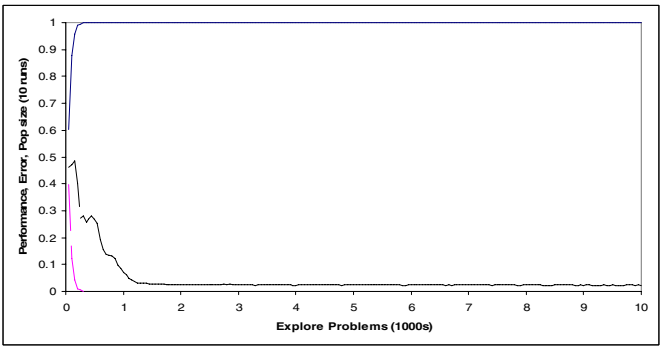


Fig. 2. Results from MILCS on the 6 multiplexer

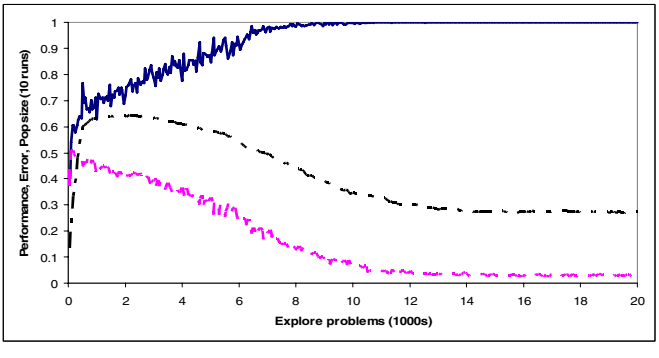


Fig. 3. Results from XCS on the 11 multiplexer

Figure 5 and Figure 6 show results from XCS and MILCS applied to the 20 multiplexer. In this case, both convergence times and final population of unique classifiers are similar.

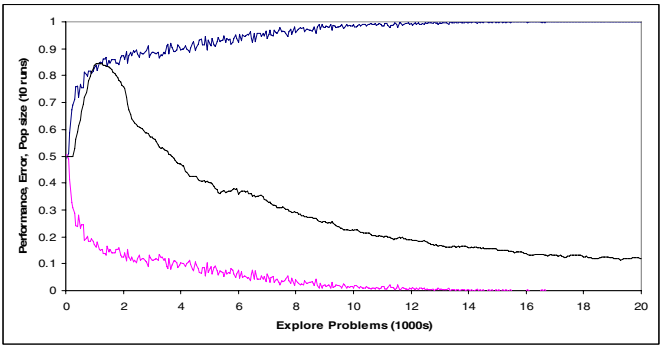


Fig. 4. Results from MILCS on the 11 multiplexer

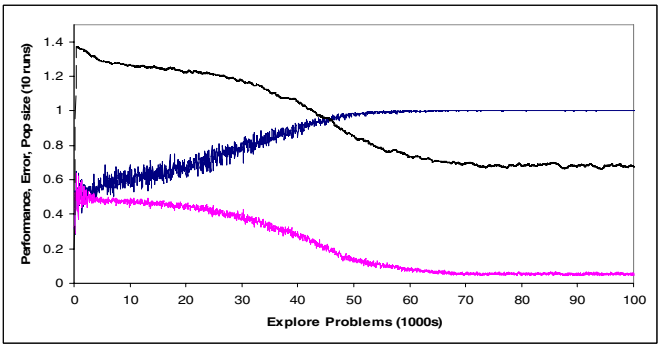


Fig. 5. Results from XCS on the 20 multiplexer

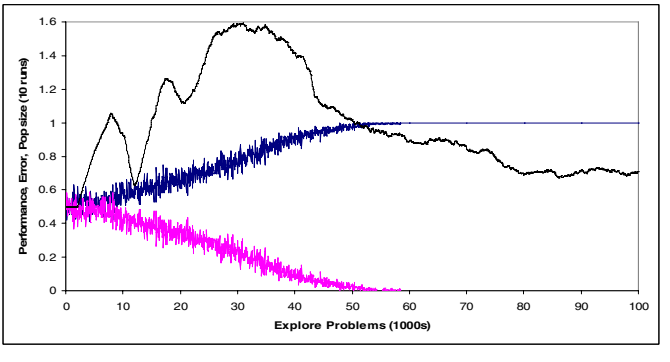


Fig. 6. Results from MILCS on the 20 multiplexer

However, we note that with XCS and MILCS, we ran complete tests on all possible inputs for each of the multiplexer problems. Each system passed this "full test" in each situation, with the exception of XCS applied to the 20 multiplexer, which failed on a small number of cases at the end of some runs portrayed in the average of 10 shown in Figure 5. While we did not overcome this difficulty with the code provided

in 4, we were able to reproduce perfect behavior in approximately 75,000 explore problems using 10. This is consistent with the results on XCS scaling for the multiplexer problems provided in 19.

6 Scalability

Our results reveal MILCS scaling up slightly worse than XCS (see Figure 7)

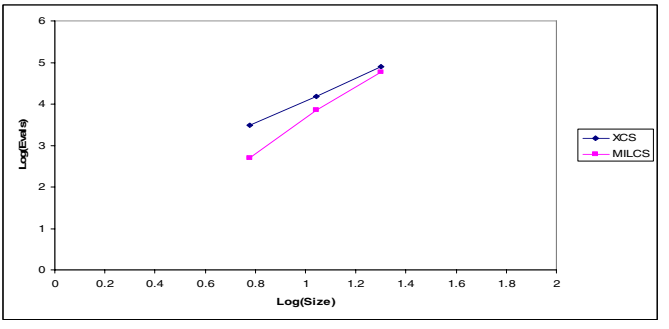


Fig. 7. Log-log plot of polynomial scaling of MILCS and XCS on the multiplexer problems

However, both are scaling as low-order polynomials. While there is an apparent difference in the scalability between the algorithms, it is not of significant order. Moreover, the previous results indicate that MILCS is ending with a substantially smaller set of unique classifiers. Hand examination of rule sets has revealed that almost all of the mature rules at the end of our MILCS runs are best-possible-generalized rules for the multiplexers. However, this examination relies on our knowledge of the underlying problem. In the following section, we introduce a method of visualizing the comparative explanatory power of the final rule sets.

7 Explanatory Power

While accuracy and compute time are important metrics for machine learning systems, it is also important to consider their *explanatory power.* While this term has no formal definition, it is considered to be the subjective human understandability of the representation of the knowledge learned by the system. As a subjective quality, it is somewhat difficult to present, particularly when the knowledge representation exists in a highly multi-dimensional space. It is also important that we avoid using out pre-existing knowledge of a problem's structure in evaluating the explanatory power of resulting knowledge representations.

7.1 Visualization of Explanatory Power

In an attempt to visualize the relative explanatory power of knowledge representations, we have developed the following procedure. Consider a structural element of

the knowledge representation (in particular, a rule). We will represent this rule as a circle, where the diameter reflects the element's generality. We will characterize the overlap of the receptive fields (conditions) of these elements by the overlap of the circles. The color of the circles will represent their output (actions).

Each circle will act as a simulated mass, connected to the other circles via springs, whose spring force is zero when the desired overlap is obtained. If rules do not overlap, we consider the difference between them to determine the relaxed spring distance between corresponding circles. Dampers between the circles are added (to insure convergence). Given this, we can use a simple dynamic systems simulation that iterates to an equilibrium, which will represent the nature of the knowledge in the system in a simple, two-dimensional space.

To make a valid comparison that is not built on pre-existing knowledge of the problem at hand, the visualization will include all elements (rules) that play a role in the output determination of the final system. In XCS exploitation mode, the following factor is computed:

$$\sum(\text{prediction} \times \text{fitness}) / \sum(\text{fitness}) \tag{2}$$

over all matching rules, for all actions, and the action with the highest factor is selected. Therefore, since all the rules participate in action selection, we include all these rules in our visualization.

However, in MILCS on an "exploitation" trial, only the rules with action-selection maturity above a threshold are employed. The rule, either matched or not matched, with the maximum predicted reward is always selected as the rule that acts. Therefore, only these rules are used in our visualization.

Figure 8 and Figure 9 show visualizations of the final rule sets from XCS and MILCS (respectively) applied to the 6 multiplexer. The smaller, final MILCS rule set, and its inclusion of only perfect generalizations is clear. We believe that this visualization shows the superior explanatory power of the resulting rule set in a way that does not depend on human understanding of the rule-form of a correct final solution for this particular problem.

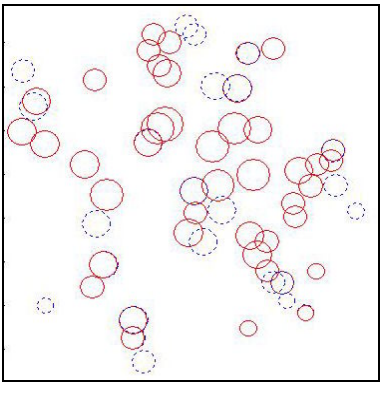


Fig. 8. Visualization of the final rule set developed by XCS on the 6 multiplexer

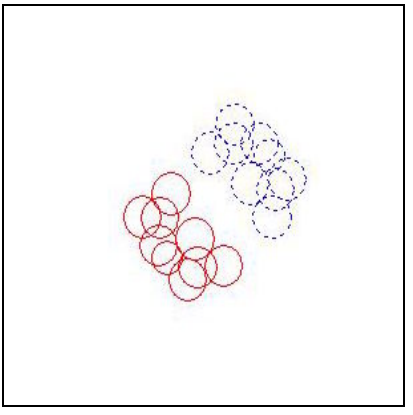


Fig. 9. Visualization of the final rule set developed by MILCS on the 6 multiplexer

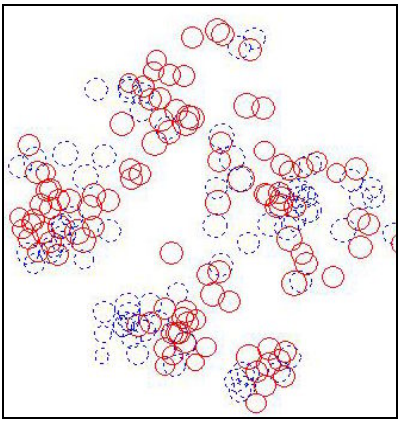


Fig. 10. Visualization of the final rule set developed by XCS on the 11 multiplexer

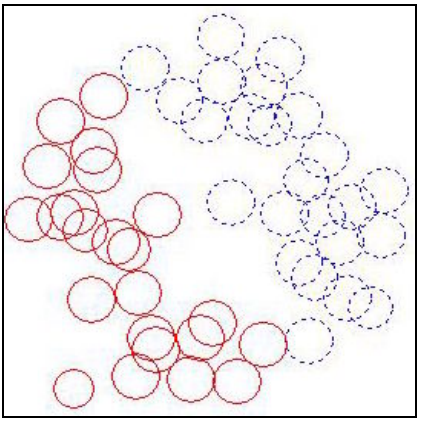


Fig. 11. Visualization of the final rule set developed by MILCS on the 11 multiplexer

Figure 10 and Figure 11 show visualizations of the final rule sets from XCS and MILCS (respectively) applied to the 11 multiplexer. Once again, the superior explanatory power of the MILCS rule set is apparent. As in the 6 multiplexer, a decision surface between the two actions is apparent, even after the projection of the rule set to a two dimensional space.

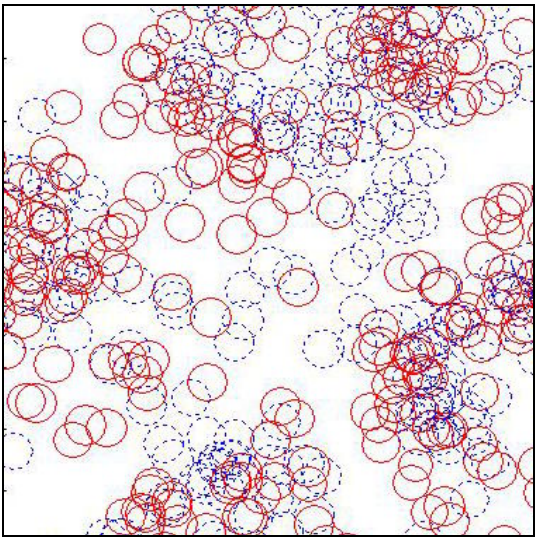


Fig. 12. Visualization of the final rule set developed by XCS on the 20 multiplexer

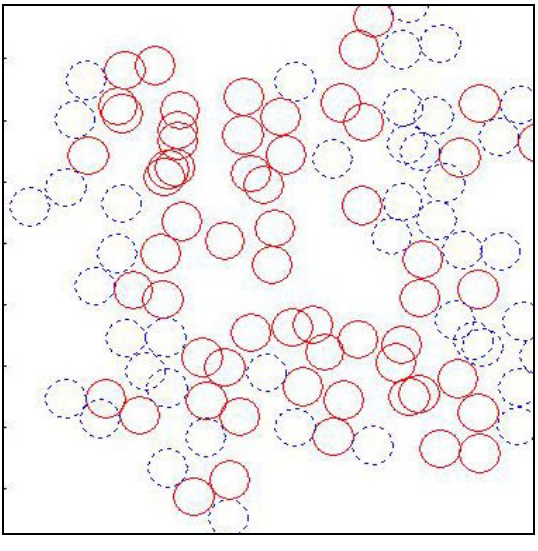


Fig. 13. Visualization of the final rule set developed by MICS on the 20 multiplexer

Figure 12 and Figure 13 show visualizations of the final rule sets from XCS and MILCS (respectively) applied to the 20 multiplexer. While a linear decision surface is no longer apparent in the MILCS result, the less complex structure of the MILCS rule set when compared to the XCS rule set is apparent. Note that while only some small portion of the final XCS rule set are perfect generalizations, and a human could detect these with knowledge of the multiplexer. That would not be the case in a problem of unknown structure.

7.2 Coordination Number Problem

Coordination Number Prediction is one of the popular approaches to prediction the 3D structure of a protein. It is defined as the prediction, for a given residue, of the number of residues from the same protein that are in contact with it. Two residues are said to be in contact when the distance between the two is below a certain threshold. A simplified protein model, the HP model, has been used to understand protein structure prediction. This model represents the sequence of a protein using two residue types: hydrophobic can polar. We have produced preliminary results on this simplified, two-state CN prediction of real proteins with source data provided by Stout et al. in 16. In order to show a thorough comparison, we present these results, along with results obtained from XCS (using 4) and GAssist 1. GAssist is a Pittsburgh-style learning classifier system which has been recently used for many protein structure prediction problems and achieved significant results 16. Accuracy results presented here use the same training and testing procedures as those in 16.

We tested all three classifier systems on 3 different window sizes for 2 state predictions.

For XCS, we employ the same parameter settings for the 20 multiplexer problem whereas MILCS parameter settings are shown in Table 5.

Table 5. MILCS parameters for the CN problem

Window size	1	2	3
MAX_POP_SIZE	100	200	300
THETA_GA	25	25	25
DONT_CARE_PROB	0.66	0.66	0.66
INIT_POP_SIZE	50	50	50
MATURITY_ACT_SELECT	450	450	450
MATURITY_SUB	400	400	400
THETA_SUB	20	20	20
MATURITY_DEL	450	450	450
DEL_WINDOW_SIZE	3200	3200	3200

GAssist results were generously provided by the authors of 16. Results are shown in Table 6.. In XCS and MILCS, the representation was one bit each for the “H”, “P”, and “not present” conditions of the neighboring residues, and one bit each for the “H” and “P” conditions of the target residue. For instance, with window size 1 this yields a 7-bit condition, given that there are two neighboring residues (3 bits each), and the

target residue (2 bits). This sparse encoding was used for consistency with the GAssist representation.

Note that for each algorithm, "Max Evals" is the number of evaluations in a run, and this most likely represents a high upper bound on the number of evaluations required to get results of the indicated quality. However, these numbers do indicate the relative order of magnitudes of convergence times for results shown. Rule set size reflects the number of rules participating in action selection at the end of each run. MILCS performance is statistically similar to GAssist, which provides the best results. MILCS results in larger rule sets than GAssist, but converges an order of magnitude faster. XCS does not perform as well as MILCS or GAssist overall. While these results are preliminary, and we feel we can improve MILCS performance, they are promising.

Table 6. Results of XCS, MILCS, and GAssist on two-state (HP) CN problems of various window sizes

Window size	Method	Accuracy	Final rule set size	Max evals
1	XCS	60.3% ±4.7%	53.1	100000
1	MILCS	63.6% ±0.5%	8.2	100000
1	GAssist	63.6% ±0.6%	4	8000000
2	XCS	61.1% ±3.6%	197.8	100000
2	MILCS	63.9% ±0.6%	15.0	100000
2	GAssist	63.9% ±0.6%	4.4	8000000
3	XCS	61.6% ±3.3%	371.2	250000
3	MILCS	63.3% ±0.8%	40.2	100000
3	GAssist	64.4% ±0.5%	4.8	8000000

8 Final Comments and Future Directions

Preliminary results with MILCS are promising, with respect to accuracy, speed, and explanatory power. While MILCS seems to scale slightly worse than XCS, this may not be an entirely fair comparison, since our preliminary results show that MILCS finds a smaller, more explanatory rule set. We believe this superior effect is to be expected, given the firm information theoretic basis of the mutual information fitness function.

Evaluating the system on more problems is the clearest direction for further investigation.

However, the concepts in MILCS are not specific to the particulars of a rule learning system. Exploring a neural network system that employs a similar structural learning paradigm is also a promising direction for future investigation. The use of mutual information in this fashion may also have application in supervised learning of other knowledge representations.

While the focus of this work has been on supervised learning, it is possible that the system may be adapted to reinforcement learning. Note that to some extent, XCS already adapts reinforcement learning to supervised learning, in its tendency to learn a complete "model" of the long term payoff function across the state/action space. The

mapping from state/action to payoff is a supervised learning problem, drawing on Bellman optimality and Q-learning the appropriate target values and error functions.

It will also be interesting to further investigate the visualization technique employed in this paper to compare explanatory power in a larger variety of knowledge representations.

Acknowledgements

The authors greatly acknowledge support provided by the UK Engineering and Physical Sciences Research Council (EPSRC) under grant GR/T07534/01.

References

1. Bacardit, J.: Pittsburgh Genetics-Based Machine Learning in the Data Mining era: representations, generalization, and run-time. PhD thesis, Ramon Llull University, Barcelona, Catalonia, Spain (2004)
2. Bernadó-Mansilla, E.: Accuracy-Based Learning Classifier Systems: Models. Analysis and Applications to Classification Tasks. Evolutionary Computation 11(3) (2003)
3. Butz, M. V.: Documentation of XCS+TS C-Code 1.2. IlliGAL report 2003023, University of Illinois at Urbana-Champaign(Source code) (2003), `ftp://gal2.ge.uiuc.edu/pub/src/XCS/XCS1.2.tar.Z`
4. (+ tournament selection) classifier system implementation in C, version 1.2 (for IlliGAL Report 2003023, University of Illinois Genetic Algorithms Laboratory) (2003), `ftp://gal2.ge.uiuc.edu/pub/src/XCS/XCS1.2.tar.ZXCS`, `ftp://gal2.ge.uiuc.edu/pub/src/XCS/XCS1.2.tar.Z`
5. Fahlman, S.E., Lebiere, C.: The Cascade-Correlation learning algorithm. In: Advances in Neural Information Processing Systems 2. Morgan Kaufmann, San Francisco (1990)
6. Goldberg, D.E.: Genetic algorithms in search, optimization and machine learning. Addison-Wesley, Reading (1989)
7. Heckerman, H.: A Tutorial on Learning with Bayesian Networks, Technical Report, MSR-TR-95-06 (1996)
8. Holland, J.H.: Adaptation in Natural and Artificial Systems, 2nd edn. MIT Press, Cambridge (1992)
9. Holland, J., Holyoak, K.J., Nisbett, R.E., Thagard, P.: Induction: Processes of Inference Learning and Discovery. MIT Press, Cambridge (1986)
10. Lanzi, P.L.: xcslib: source code, `http://xcslib.sourceforge.net/`
11. Lanzi, P.L., Loiacono, D., Wilson, S.W., Goldberg, D.E.: XCS with Computed Prediction for the Learning of Boolean Functions. Evolutionary Computation 1, 588–595 (2005)
12. Shannon, C.E.: A Mathematical Theory of Communication. Bell System Technical Journal 27, 379–423,623–656 (1948)
13. Shannon, C.E.: Communication in the presence of noise. Proc. Institute of Radio Engineers 37(1), 10–21 (1949)
14. Smith, R.E., Cribbs, H.B.: Is a classifier system a type of neural network? Evolutionary Computation 2(1), 19–36 (1994)
15. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)

16. Stout, M., Bacardit, J., Hirst, J., Krasogor, N., Blazewicz, J.: From HP lattice models to real proteins: coordination number prediction using learning classifier systems. In: 4th European Workshop on Evolutionary Computation and Machine Learning in Bioinformatics (2006)
17. Wilson, S.W.: Classifier Fitness based on Accuracy. Evolutionary Computation 3(2), 149–175 (1994)
18. Wilson, S.W.: ZCS: A Zeroth-Level Classifier System. Evolutionary Computation 2(1), 1–18 (1994)
19. Wilson, S.W.: Generalization in the XCS classifier system. In: Koza, J.R., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D.B., Garzon, M.H., Goldberg, D.E., Iba, H., Riolo, R. (eds.) Genetic Programming 1998: Proceedings of the Third Annual Conference, pp. 665–674. Morgan Kaufmann, San Francisco (1998)

On Lookahead and Latent Learning in Simple LCS

Larry Bull

School of Computer Science
University of the West of England
Bristol BS16 1QY, U.K.
Larry.Bull@uwe.ac.uk

Abstract. Learning Classifier Systems use evolutionary algorithms to facilitate rule- discovery, where rule fitness is traditionally payoff based and assigned under a sharing scheme. Most current research has shifted to the use of an accuracy-based scheme where fitness is based on a rule's ability to predict the expected payoff from its use. Learning Classifier Systems that build anticipations of the expected states following their actions are also a focus of current research. This paper presents a simple but effective learning classifier system of this last type, using payoff-based fitness, with the aim of enabling the exploration of their basic principles, i.e., in isolation from the many other mechanisms they usually contain. The system is described and modelled, before being implemented. Comparisons to an equivalent accuracy-based system show similar performance. The use of self-adaptive mutation in such systems in general is then considered.

1 Introduction

Holland's Learning Classifier System (LCS) [1976] represents a form of machine learning which exploits evolutionary computing to produce inductive structures within an artificial entity. Typically, such systems use stimulus-response rules to form chains of reasoning. However, Holland's architecture has been extended to include mechanisms by which higher levels of cognitive capabilities, along the lines of those envisaged in [Holland et al., 1986], can emerge; the use of predictive modelling within LCS has been considered through alteration to the rule structure [e.g., Riolo, 1991]. Using maze tasks loosely based on those of early animal behaviour experiments, it has been found that LCS can learn effectively when reward is dependent upon the ability to accurately predict the next environment state/sensory input. LCS with such 'lookahead' typically work under latent learning, i.e., they build a full predictive map of the environment without external reinforcement. LCS of this general type have gained renewed interest after Stolzmann presented the heuristics-based ACS [Stolzmann, 1998]. ACS was found to produce over-specific solutions through the workings of its heuristics and was later extended to include a Genetic Algorithm (GA)[Holland, 1975] - ACS2 [Butz & Stolzmann, 2002]. Bull [2002] presented an extension to Wilson's simple payoff-based LCS - ZCS [Wilson, 1994] - which is also able to form anticipations under latent learning. Significantly, this was the first anticipatory system to build such models through the GA alone; Riolo [1991]

J. Bacardit et al. (Eds.): IWLCS 2006/2007, LNAI 4998, pp. 154–168, 2008.

did not include a GA. Most current work in LCS has shifted to using accuracy as rule fitness, after Wilson presented XCS [Wilson, 1995]. Bull [2004] presented a simple accuracy-based LCS which can create such anticipations using only the GA – YCSL. In this paper, a simple payoff-based LCS which can create anticipations using only the GA is presented and explored, based on the ZCS-derived system MCS [Bull, 2005].

2 MCSL: A Simple Anticipatory Classifier System

In this paper, as in ACS and its related systems such as YACS [Gerard & Sigaud, 2002], and in [Bull, 2002; 2004], an explicit representation of the expected next environmental state is used to create a simple payoff-based anticipatory LCS which uses lookahead under latent learning - MCSL. That is, rules are of the general form:

<condition> : <action> : <anticipation>

Generalizations (#'s) are allowed in the condition and anticipation strings. Where #'s occur at the same loci in both, the corresponding environmental input symbol 'passes through' such that it occurs in the anticipated description for that input. Similarly, defined loci in the condition appear when a # occurs in the corresponding locus of the anticipation. MCSL is a Learning Classifier System without internal memory, where the rulebase consists of a number (N) of rules with the above form. Associated with each rule is a fitness and where the initial random population have this initialized to 10.

On receipt of an input message, the rulebase is scanned, and any rule whose condition matches the message at each position is tagged as a member of the current match set [M]. An action is then chosen from those proposed by the members of the match set at random and all rules proposing the selected action form an action set [A].

Although the use of fitness sharing for externally received payoff (P) had been suggested before [e.g., Holland, 1985], it was not until Wilson introduced the action set-based scheme in ZCS that simple but effective fitness sharing in LCS became possible [Bull & Hurst, 2002]. MCSL, like MCS, uses the fitness sharing mechanism of ZCS, i.e., within action sets. Reinforcement consists of updating the fitness f of each member of the current [A] using the Widrow-Hoff delta rule with learning rate β. Payoff is divided equally amongst all members of [A]:

$$f_j \leftarrow f_j + \beta \left((P / |[A]|) - f_j \right) \quad (1)$$

MCSL employs two discovery mechanisms, a GA and a covering operator. On each time-step there is a probability g of GA invocation. When called, the GA uses roulette wheel selection to determine two parent rules from the population based on their fitness. All parts of the rules are considered to be concatenated together for the GA. Offspring are produced via mutation (probability μ, turned into a wildcard at rate $p_{\#}$) and crossover (single point with probability χ), inheriting the parents' fitness values or their average if crossover is invoked. Replacement of existing members of the rulebase is inversely proportional to fitness, i.e., $1/(f_j + 1)$, using roulette wheel selection. If no rules match on a given time step, then a covering operator is used which creates a rule with the message as its condition (augmented with wildcards at

the rate $p_{\#}$) and a random action and anticipation, which then replaces an existing member of the rulebase in the usual way. It is assigned the default fitness f_0.

Hence MCSL represents a simple anticipatory LCS which relies solely upon the GA to search the space of possible generalizations; other heuristics need not be considered as pre-requisites for the effective use of a payoff-based fitness scheme. Here the term effective is taken to mean able to solve problems of low complexity whilst remaining open to close modelling; the canonical GA may be defined in much the same way. The mechanisms of MCSL are now modelled, in keeping with its philosophy, in a simple way.

3 A Simple Model of MCSL

The evolutionary algorithm in MCSL is a steady-state GA. A simple steady-state GA without genetic operators can be expressed in the form:

$$n(k, t+1) = n(k, t) + n(k, t)\, R(k, t) - n(k, t)\, D(k, t) \quad (2)$$

where $n(k, t)$ refers to the number of individuals of type k in the population at time t, $R(k, t)$ refers to their probability of reproductive selection and $R(k, t)$ to their probability of deletion. Roulette-wheel selection is used, i.e., $R(k, t) = f(k, t)/f(K, t)$, where $f(k, t)$ is the fitness of individuals of type k (Equation 1) and $f(K, t)$ is the total population (K) fitness. Replacement is inversely proportional to fitness as described above.

Table 1 shows the error 'rewards' for each of the rules considered. Those rules which experience two rewards have the average shown. Figure 1 shows the maze environment from which the errors are drawn. The maze contains two locations, one providing the LCS with input '0' and the other with input'1'. In both locations an action '0' means no move and action '1' means a move to the other location.

Table 1. Rewards for the modelled maze task

C:A:Ant	Reward	C:A:Ant	Reward	C:A:Ant	Reward
0:0:0	1000	1:0:0	0	#:0:0	500
0:0:1	0	1:0:1	1000	#:0:1	500
0:0:#	1000	1:0:#	1000	#:0:#	1000
0:1:0	0	1:1:0	1000	#:1:0	500
0:1:1	1000	1:1:1	0	#:1:1	500
0:1:#	0	1:1:#	0	#:1:#	0

The rulebase is of size N=400 and the initial proportions of each rule in the population are equal (N/18), and β=0.2. It is assumed that both inputs are presented with equal frequency, that both actions are chosen with equal frequency and that the GA fires once every four cycles (i.e., g=0.25). The rules' parameters are updated according to Equation 1 on each cycle.

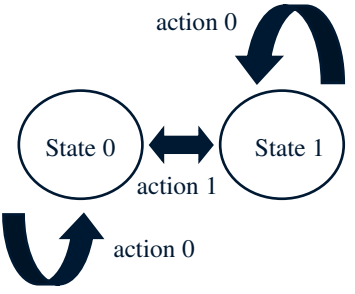
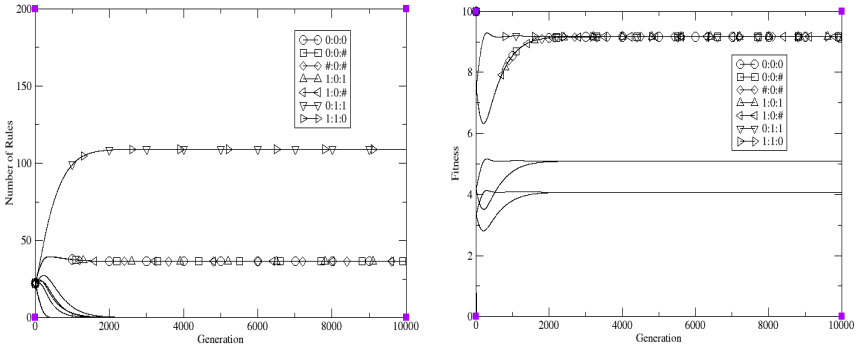


Fig. 1. Simple two location maze considered

Figure 2 shows the behaviour of the modelled MCSL on the simple maze task. Figure 2(a) shows how only the rules which accurately anticipate the next state (i.e., following their action being taken in the locations they match) exist in the final population. The rulebase is roughly divided between rules with action '0' and those with action '1' but there is no explicit pressure for a maximally general solution. Figure 2(b) shows the corresponding trajectories of the rules' fitnesses with all accurate anticipators having the same – highest - fitness. Therefore the simple payoff-based fitness scheme of MCSL results in a rulebase which completely maps the maze environment under a latent learning scenario.



(a) (b)

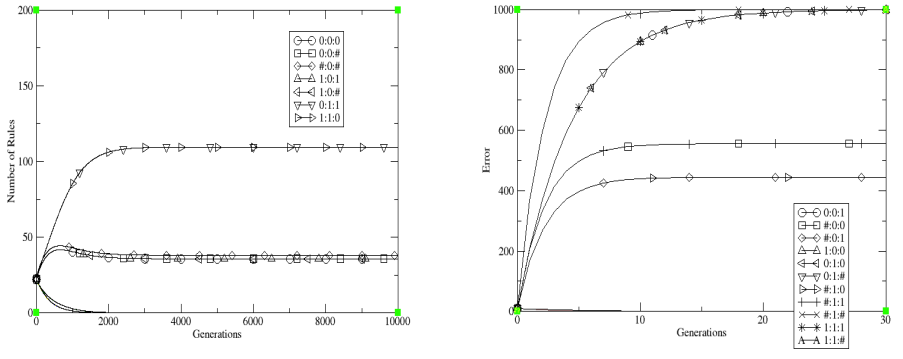
Fig. 2. Behaviour of model MCSL on the maze task, showing numerosity (a) and fitness (b)

As noted above, Bull [2004] has previously presented a simple accuracy-based anticipatory system, YCSL. Instead of maintaining a single fitness parameter, each rule maintains two parameters, an error (ε) in the rule's prediction abilities and an estimate of the average size of the niches in which that rule participates (σ) updated as follows:

$$\varepsilon_j \leftarrow \varepsilon_j + \beta(E - \varepsilon_j) \quad (3)$$

$$\sigma_j \leftarrow \sigma_j + \beta(|[A]| - \sigma_j) \quad (4)$$

where E is zero if the anticipation of the given rule correctly describes the following state, otherwise 1000. Fitness is inversely proportional to the error, i.e., $1/(\varepsilon^{\nu}+1)$, and replacement uses σ. Again, the system is simple enough to be amenable to direct modelling using equation 2, as shown in Figure 3 (from [Bull, 2004]).



(a) (b)

Fig. 3. Behaviour of model YCSL on the maze task, showing numerosity (a) and error (b)

Comparison between the two systems indicates that both are equally able to build a correct model of the simple maze. The population in MCSL is faster to reach its steady state (by approximately 1000 generations) despite the errors in YCSL reaching their steady state very much more quickly than the fitnesses in MCSL.

4 MCSL in T-Mazes

MCSL has been implemented and investigated using a version of the T-maze presented in [Riolo, 1991]. As noted above, motivation for exploring the use of learning without external reinforcement comes from early experiments in animal behaviour. Typically, rats were allowed to run around a T-maze, as shown in Figure 4, where the food cell (state 7) would be empty but a different colour to the rest of the maze. The rats would then be fed in the marked location. Finally, the rats were placed at the start location (state 0) and their ability to take the shortest path (go left at the T-junction in Figure 4) to the food recorded. It was found that rats could do this with around 90% efficiency. Those which were not given the prior experience without food were only 50% efficient, as expected [e.g., Seward, 1949].

To examine the performance of the simple anticipatory LCS presented here the following scenario is used. The LCS is placed randomly in the maze and a matchset is formed. Sensory input in each location of the maze is the binary encoded number for that state (3 bits) and there are four possible actions - Left, Right, Forwards and

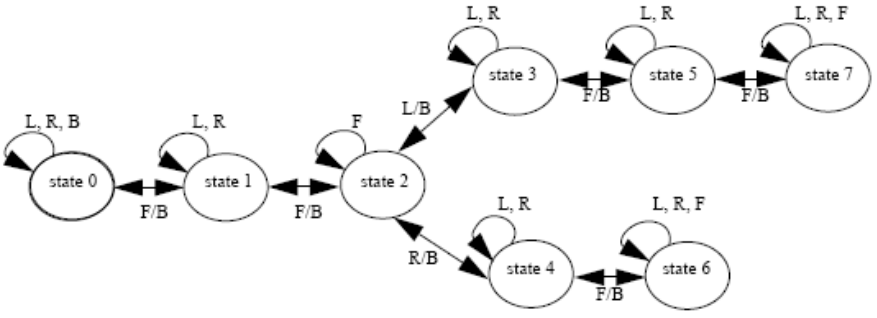


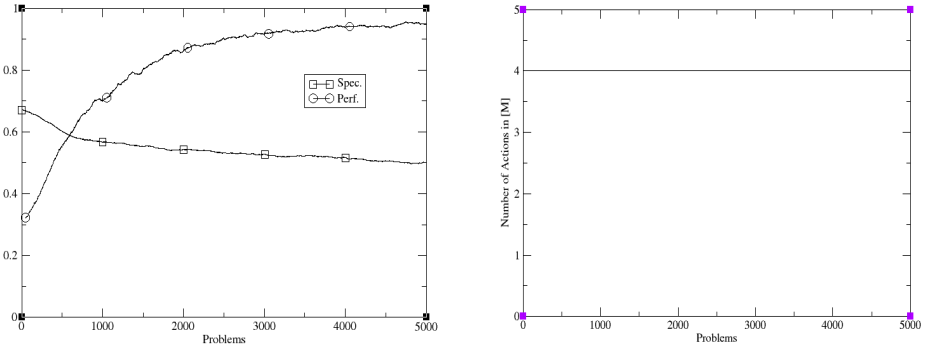
Fig. 4. T-maze considered. A backwards move always causes a move to the location on the left (except state 0)

Backwards (2 bits). An action is chosen at random from the matchset. All rules which propose the chosen action are updated as described in Section 2 and the GA fires probabilistically. That is, each rule in [A] has the anticipation it created on forming [M] compared to the state into which the LCS has since moved. If it matches, a reward of 1000 is given, otherwise 0. The LCS is then randomly placed in another location and the process repeated (in keeping with the previous model). The average specificity (fraction of non-# symbols) of the condition and anticipation is recorded, as is the number of actions present in each [M], with results shown the average of ten runs.

Figure 5 shows the performance of MCSL on the simple T-maze using the same parameters as in Section 3, with $p_{\#}$=0.33, χ=0.5 and μ=0.01 under the same training scheme. Figure 5(a) shows how the fraction of individuals in a given [A] which accurately predict the next state rises to around 1 (50-point moving average, after [Wilson, 1994]) and the average specificity drops to around 50% indicating the exploitation of generalizations. Figure 5(b) shows how all four actions are present and maintained in each [M] throughout learning. Hence MCSL is able to build a complete and accurate anticipatory map of the simple T-maze (assuming the most numerous anticipation within a given [A] is used in internal processing/planning).

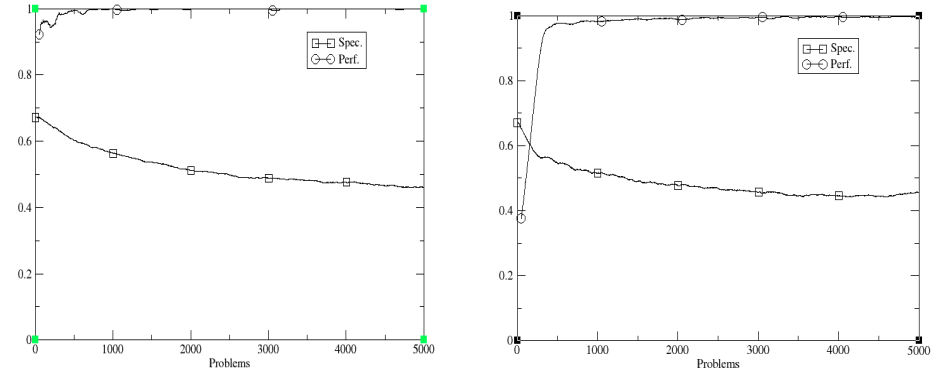
YCSL was applied to the same maze with the same parameters in [Bull, 2004], as shown in Figure 6(a). Figure 6(a) shows how the fraction of trials upon which the rule with the highest fitness has a correct anticipation very quickly rises to 1 (50-point moving average used) and the average specificity drops to around 45% indicating a slightly better level of generalization. The number of actions per [M] was maintained at four throughout (not shown). Hence YCSL is also able to build a complete and accurate anticipatory map of the simple T-maze (assuming the anticipation with the highest fitness within a given [A] is used in internal processing/planning).

Under the standard reinforcement learning scenario, the fitness sharing mechanism in MCS has been found to benefit from a maximally high learning rate and high GA activity [Bull, 2005], i.e., β=1.0 and g=1.0. Payoff-based LCS using fitness sharing hold their estimation of utility in *rule numerosity* and therefore the instantaneous fitness update means a rule's fitness can immediately consider the current numerosity, something which is constantly changed by the actions of the GA; it appears that a high learning and GA rate allows the LCS to approximate rule utility more efficiently. Figure 6(b) shows how the change means MCSL learns faster than before and produces a solution as general as YCSL.



(a) (b)

Fig. 5. Fraction of [A] with correct anticipation (a) and number of actions per [M] (b)



(a) (b)

Fig. 6. YCSL performance on T-maze (a) and MCSL with β=1.0, *g*=1.0 (b)

The maze in Figure 4 is somewhat simple and hence MCSL has also been tested using a more complex version along the lines of other maze tasks typically used in the LCS literature (after [Wilson, 1994]) - Woods 10 [Bull, 2002] (Figure 7). The LCS is placed randomly in the maze and a matchset is formed as before. Sensory input in each location of the maze is encoded as a 16-bit binary string with two bits representing each cardinal direction. A blank cell is represented by 00, the food location (F) by 11 and trees (T) by 10 (01 has no meaning). The message is ordered with the cell directly above the LCS represented by the first bit-pair and then proceeds clockwise around it. Again, an action is chosen at random from the matchset where there are now eight possible actions (cardinal moves) and the LCS can move into any one of the surrounding eight cells on each discrete time step, unless occupied by a tree or it is the food location (this avoids creating a sensory ambiguity). All rules which propose the chosen action are updated and all other details are as before. One further mechanism is incorporated for this harder task (after [Bull, 2002]): the first *N* random

rules of the rulebase have their anticipation created using cover (with #'s included as usual) in the first [A] of which they become a member. This goes some way to make " ... good use of the large flow of (non-performance) information supplied by the environment." [Holland, 1990]. Rules created under the cover operator also receive this treatment. In this way the GA explores the generalization space of the anticipations created by the simple heuristic.

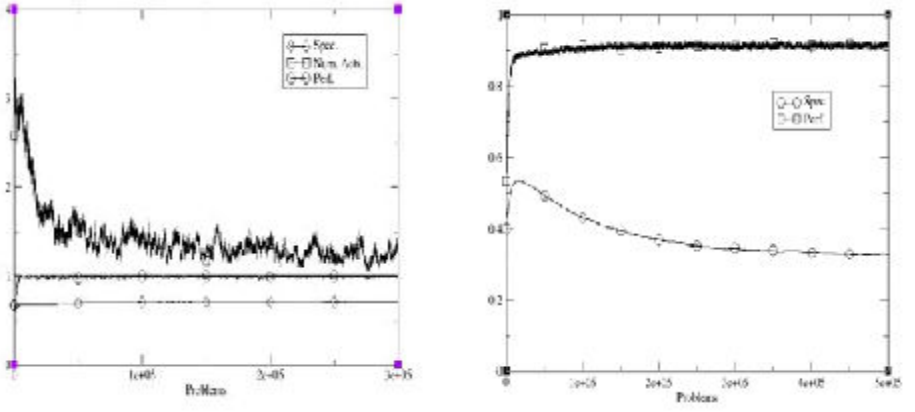
Figure 8(a) shows how the system as used in Figure 6(b), but with N=5000, $p_{\#}$=0.6 and μ=0.04 (after [Bull, 2004]), is unable to produce a full model of Woods 10. Indeed, the system appears to predict a low number of actions, with increasing specificity, and the rule with the highest numerosity in those few [A] rarely anticipates the next state correctly.

Under the operations of the GA within ZCS there is a reproduction cost such that parents give half of their fitness to their offspring. No explanation for this mechanism is given in [Wilson, 1994] but it has been suggested that it reduces "the initial 'runaway' success of those rules in high payoff niches" [Bull & Studley, 2002]. That is, once a rule has reproduced, it and its offspring are much less likely to be picked again under the global GA until their niche occurs, at which point they are assigned a new fitness appropriate for the current numerosity. This last point was shown to be significant above and is fundamental to the way in which fitness sharing avoids overgeneral rules since it removes any advantage in difference between niche payoff levels [Bull & Hurst, 2002]; the payoff available to individual rules becomes the same in all niches once numerosities have been adjusted appropriately by the GA.

Bull [2005] presented a tuneable mechanism with which to achieve the same general dynamic within the fitness sharing process - default fitness allocation (DFA) – where offspring and parents have their fitnesses set to f_0. Using well-known Boolean problems it was suggested that the fitness halving scheme does not scale as well as DFA. Figures 8(b) and 9(b) show how both mechanisms appear to work equally well within MCSL. Eight actions were maintained throughout in both cases (not shown). This result corresponds with those reported in [Bull, 2002] where ZCS was successfully extended to build anticipations in Woods 10 and other mazes. Figure 9(a) shows how performance is equivalent to that of YCSL with XCS's triggered niche GA (θ_{GA}=100) and increased fitness scaling (ν=10), as discussed in [Bull, 2004].

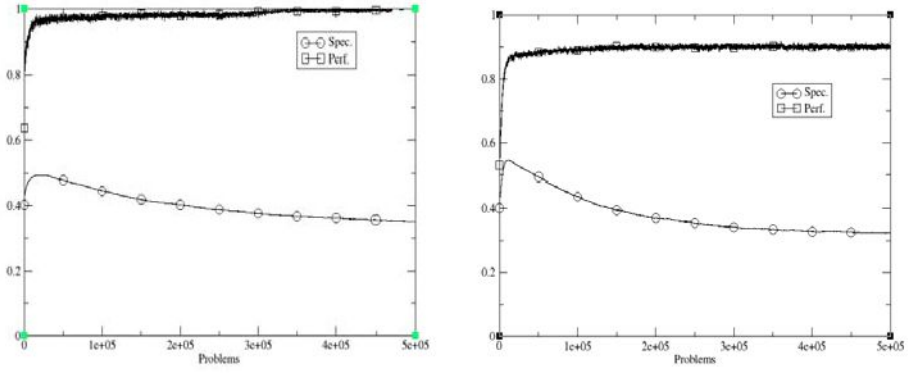
T	T	T	T	T	T	T
T	F	T	T	T		T
T		T	T	T		T
T						T
T	T	T		T	T	T
T	T	T		T	T	T
T	T	T	T	T	T	T

Fig. 7. The Woods 10 maze



(a) (b)

Fig. 8. Fraction of [A] with correct anticipation and number of actions per [M] (a) and with DFA heuristic (b)



(a) (b)

Fig. 9. YCSL performance on same T-maze (a) and MCSL with fitness halving (b)

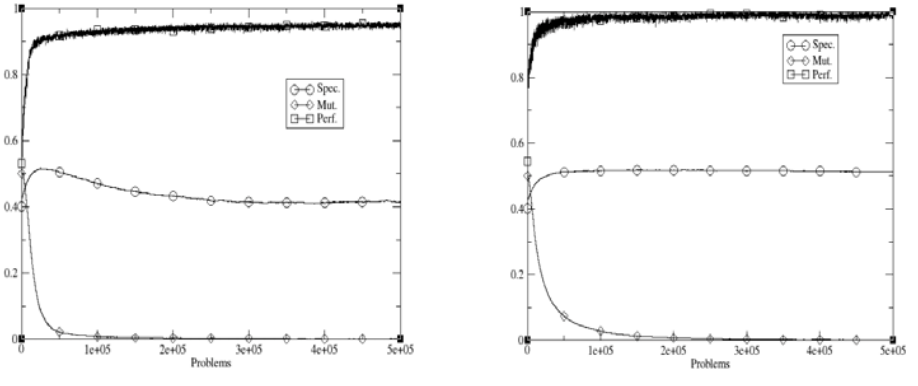
5 Self-adaptive Mutation

Typically in LCS, as within GAs, the parameters controlling the algorithm are global and remain constant over time. However this is not always the case in evolutionary computing; in Evolution Strategies [Rechenberg, 1973], forms of Evolutionary Programming (Meta-EP) [Fogel, 1992] and in some GAs (e.g., [Bäck, 1992]), the mutation rate is a locally evolving entity in itself, i.e., it adapts during the search process. This self-adaptive form of mutation not only reduces the number of hand-tunable parameters of the evolutionary algorithm, it has also been shown to improve performance (e.g., [Bäck, 1992]). The approach has been used to add adaptive mutation to both ZCS and XCS [Bull et al., 2000], and to control other system parameters, such as the learning rate, in ZCS [Hurst & Bull, 2001] and XCS [Hurst &

Bull, 2002] (see also [Bull et al., 2005][Wyatt & Bull, 2004] for other uses of parameter self-adaptation in LCS). The results presented demonstrate that adaptive parameters can provide improved performance, particularly in dynamic environments. It can be noted that the general idea of automatically setting parameters in LCS was first suggested in [Dorigo & Colombetti, 1997].

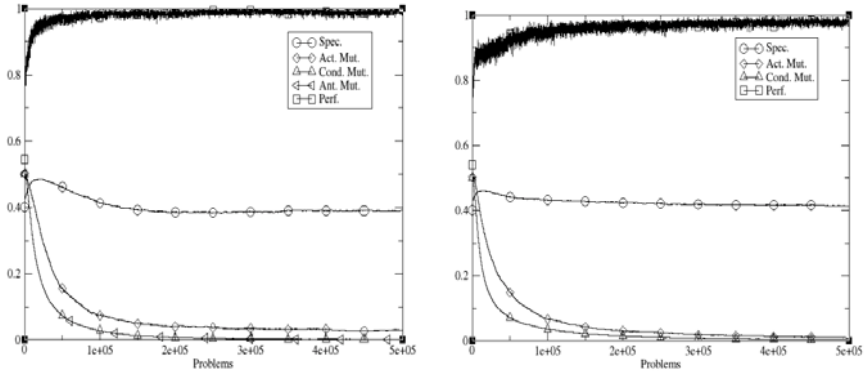
In the previous work, a single self-adapting mutation rate parameter was added to each rule. That is, each rule has its own mutation rate μ, stored as a real number and initially seeded uniform randomly in the range [0.0,1.0]. This parameter is passed to its offspring either under recombination or directly. The offspring then applies its mutation rate to itself using a Gaussian distribution, i.e., $\mu_i' = \mu_i e^{N(0,1)}$, before mutating the rest of the rule at the resulting rate. Figure 10(a) shows how the approach can be successfully used within MCSL in Woods 10 using all other parameters as before. There is a significant rise (T-test, $P<0.05$) in the specificity of the solution produced however compared with the fixed single mutation rate of $\mu=0.04$ used in Figure 8(b). This despite the mean mutation rate dropping considerably lower than the fixed rate. The equivalent YCSL (Figure 10(b)) no longer produces a solution such that the fittest rule in a niche always accurately anticipates the next state and it suffers an even larger rise in average specificity. Bull et al. [2000] noted a greater sensitivity to the self-adapting mutation rate in XCS compared to ZCS in a number of problems. Bull [2001] confirmed the accuracy-based fitness scheme's greater sensitivity to the (fixed) mutation rate using simple Markov models of both schemes. Butz et al. [2003] have also examined this formally for XCS, concluding "mutation can have a negative effect when set too high possibly disrupting subsolutions".

The creation of an anticipatory system through the GA alone means the evolutionary process is designing rule structures of increased complexity in comparison to the traditional stimulus-response rules. It may therefore be beneficial to increase the freedom of the mutation operator to search the sub-spaces of the different parts of the rule encoding at different rates; improvements in performance may be possible with separate self-adapting mutation rates for the condition, action and anticipation. This has been explored with each mutation rate adapting as before.



(a) (b)

Fig. 10. MCSL with single self-adaptive mutation rate in Woods 10 (a) and equivalent YCSL (b)



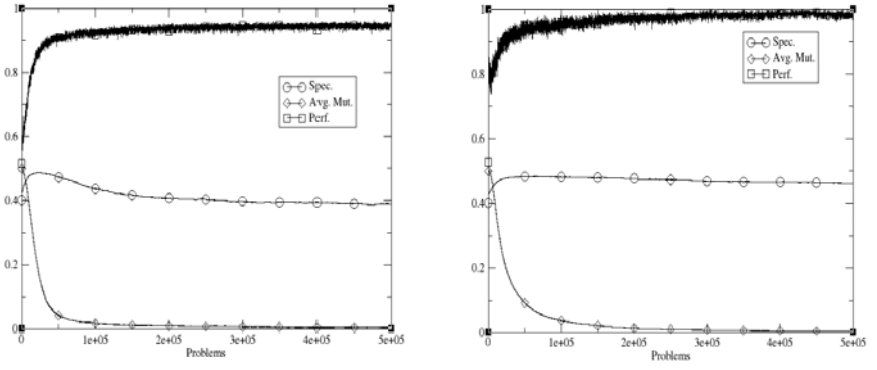
(a) (b)

Fig. 11. MCSL with a self-adaptive mutation rate for each of the condition, action, and anticipation (a) and in the equivalent YCSL (b)

Figure 11(a) shows how the use of a separate self-adapting mutation rate for each component of a rule can be used successfully in MCSL, with the rate for the action component adapting differently to those of the condition and anticipation. Specificity is still higher than with the fixed rate as a consequence however but less than that seen with the single adapting parameter. No significant increase in learning speed is seen, again solutions appear to settle after around 200,000 problems, which is faster than with the fixed rate which typically settle to a solution after around 300,000 problems; a more specific solution is learnt more quickly. Figure 11(b) shows how with YCSL the same difference in mutation rate for the action and the condition and anticipation is again seen. A decrease in the average specificity is also obtained with the three mutation rates but it is again still more specific than with the fixed rate. The fittest rule in each action set is again not always an accurate predictor of the next environmental state, as it was in Figure 9(a) with the fixed mutation rate.

Typically under Evolution Strategies every dimension of a given solution is treated separately by mutation such that each individual consists of n variables and n mutation rates. The updating of the mutation rate is then $\mu_i' = \mu_i \, e^{N(0,1) + N_i(0,1)}$. That is, a common offset is first determined for all variables and then each variable determines a further local offset, both from a Gaussian distribution. The same approach has been tried in MCSL and YCSL. Figure 12 shows that this scheme has little effect over the previous two: solutions are more specific than with the fixed rate and YCSL is unable to produce an accurate map using the fittest niche rule approach.

The increase in average specificity of rules in YCSL results in a decrease in performance since accurate anticipators are not discovered in each of the 80 niches (10 locations, 8 possible actions in each) in the time allowed. Therefore niche populations of approximately 5000/80 = 40 rules appear insufficient when the mutation rate is inappropriately high during the early part of the search. Experiments seeding the initial mutation rate spreads in the range [0,0.5] gave no improvement (not shown). Figure 13 shows the effects of increasing the population size, N=15,000. That is, it appears YCSL is failing to satisfy something akin to the "cover challenge",



(a) (b)

Fig. 12. MCSL with a self-adaptive mutation rate for each gene (a) and in the equivalent YCSL (b)

as identified by Butz et al. [2003], wherein the population is not large enough to sustain all accurate rules discovered at a given time and produce new ones to cover other inputs. Therefore the cover and GA replacement operators cause the constant deletion of required rules. As can be seen, with a larger population the system is again optimal in its map building assuming the fittest rule in each [A] is chosen for planning.

6 Conclusions

Learning Classifier Systems that build a full predictive map of the environment without external reinforcement have recently gained renewed interest. This paper has presented a simple payoff-based system, termed MCSL, which is capable of this task using only the genetic algorithm. Due to its simplicity, an executable model of MCSL has been presented and its finding that optimal performance is possible with such a system was confirmed experimentally. However, it was found that a generalization pressure mechanism was required for a harder maze task. The performance and general characteristics of MCSL were compared to a simple accuracy-based anticipatory LCS and little difference seen.

The use of self-adaptive mutation within such systems was also explored. Results suggest that only in the payoff-based scheme is there some benefit in the maze used from more than a single mutation rate per rule, as was previously implemented [Bull et al., 2000]. Future work should consider non-stationary mazes, as in [Hurst & Bull, 2003] for example. Results also confirm that accuracy-based fitness is more sensitive to the mutation rate than payoff-based fitness.

In contrast to using a purely evolutionary approach, O'Hara and Bull [2005] have highlighted the supervised nature of creating anticipations (see [Lanzi, 2003] for a related study). They present an approach wherein each rule also carries an artificial neural network. The network takes the current stimulus and action as an input and produces the expected next sensory state as its output. These mappings are learned

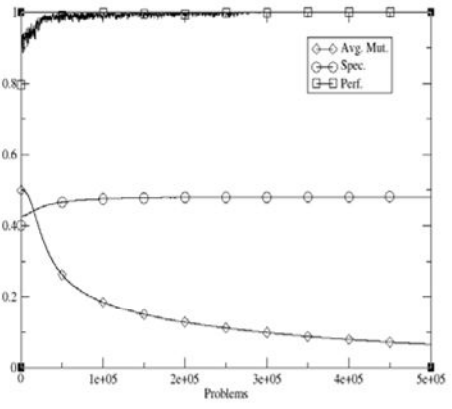


Fig. 13. YCSL with a self-adaptive mutation rate for each gene and a larger population

under a standard incremental supervised learning protocol; each received input after an action is used to further train the network. Future work should consider hybrid systems, as represented by ACS2, perhaps based on the neural rule representation, particularly for continuous-valued spaces. The GA could be exploited to design appropriate prediction network structure, for example (see [Bull & Hurst, 2003] for a purely GA-based approach in this direction).

References

Back, T.: Self-Adaptation in Genetic Algorithms. In: Varela, F.J., Bourgine, P. (eds.) Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life, pp. 263–271. MIT Press, Cambridge (1992)

Booker, L.B.: Triggered Rule Discovery in Classifier Systems. In: Schaffer, J.D. (ed.) Proceedings of the Third International Conference on Genetic Algorithms, pp. 265–274. Morgan Kaufmann, San Francisco (1989)

Bull, L.: Simple Markov Models of the Genetic Algorithm in Classifier Systems. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 2000. LNCS (LNAI), vol. 1996, pp. 29–36. Springer, Heidelberg (2001)

Bull, L.: Lookahead and Latent Learning in ZCS. In: Langdon, W.B., Cantu-Paz, E., Mathias, K., Roy, R., Davis, D., Poli, R., Balakrishnan, K., Honavar, V., Rudolph, G., Wegener, J., Bull, L., Potter, M.A., Schultz, A.C., Miller, J.F., Burke, E., Jonoska, N. (eds.) GECCO-2002: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 897–904. Morgan Kaufmann, San Francisco (2002)

Bull, L.: Lookahead and Latent Learning in a Simple Accuracy-based Learning Classifier System. In: Yao, X., et al. (eds.) Parallel Problem Solving from Nature - PPSN VIII, pp. 1042–1050. Springer, Heidelberg (2004)

Bull, L.: Two Simple Learning Classifier Systems. In: Bull, L., Kovacs, T. (eds.) Foundations of Learning Classifier Systems, pp. 63–90. Springer, Heidelberg (2005)

Bull, L., Hurst, J.: ZCS Redux. Evolutionary Computation 10(2), 185–205 (2002)

Bull, L., Hurst, J.: Lookahead and Latent Learning in a Neural Learning Classifier System with Self-Adaptive Constructivism. Technical Report UWELCSG03-012 (2003), http://www.cems.uwe.ac.uk/lcsg

Bull, L., Studley, M.: Consideration of Multiple Objectives in Neural Learning Classifier Systems. In: Merelo, J., Adamidis, P., Beyer, H.-G., FernandezVillicanas, J.-L., Schwefel, H.-P. (eds.) PPSN 2002. LNCS, vol. 2439, pp. 558–567. Springer, Heidelberg (2002)

Bull, L., Hurst, J., Tomlinson, A.: Self-Adaptive Mutation in Classifier System Controllers. In: Meyer, J.-A., Berthoz, A., Floreano, D., Roitblatt, H., Wilson, S.W. (eds.) From Animals to Animats 6 - The Sixth International Conference on the Simulation of Adaptive Behaviour, pp. 460–470. MIT Press, Cambridge (2000)

Bull, L., Lawson, I., Adamatzky, A., De Lacy Costello, B.: Towards Predicting Spatial Complexity: A Learning Classifier System Approach to Cellular Automata Identification. In: Proceedings of the IEEE Congress on Evolutionary Computation, pp. 136–141. IEEE, Los Alamitos (2005)

Butz, M.V., Wilson, S.W.: An Algorithmic Description of XCS. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 2000. LNCS (LNAI), vol. 1996, pp. 253–272. Springer, Heidelberg (2001)

Butz, M.V., Stolzmann, W.: An Algorithmic Description of ACS2. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 2001. LNCS (LNAI), vol. 2321, pp. 211–230. Springer, Heidelberg (2002)

Butz, M.V., Goldberg, D.E., Tharakunnel, K.: Analysis and Improvement of Fitness Exploitation in XCS: Bounding Models, Tournament Selection, and Bilateral Accuracy. Evolutionary Computation 11(3), 239–278 (2003)

Dorigo, M., Colombetti, M.: Robot Shaping. MIT Press, Cambridge (1997)

Gerard, P., Sigaud, O.: YACS: Combining Dynamic Programming with Generalization in Classifier Systems. In: Lanzi, P.-L., Stolzmann, W., Wilson, S.W. (eds.) Advances in Learning Classifier Systems: Proceedings of the Third International Workshop, pp. 52–69. Springer, Heidelberg (2001)

Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press (1975)

Holland, J.H.: Adaptation. In: Rosen, R., Snell, F.M. (eds.) Progress in Theoretical Biology, vol. 4, pp. 263–293. Plenum (1976)

Holland, J.H.: Properties of the Bucket Brigade. In: Grefenstette, J.J. (ed.) Proceedings of the First International Conference on Genetic Algorithms and their Applications, pp. 1–7. Lawrence Erlbaum Associates, Mahwah (1985)

Holland, J.H.: Escaping Brittleness. In: Michalski, R.S., Carbonell, J.G., Mitchell, T.M. (eds.) Machine Learning: An Artificial Intelligence Approach, vol. 2, pp. 48–78. Morgan Kauffman, San Francisco (1986)

Holland, J.H.: Concerning the Emergence of Tag-Mediated Lookahead in Classifier Systems. Physica D 42, 188–201 (1990)

Holland, J.H., Holyoak, K.J., Nisbett, R.E., Thagard, P.R.: Induction: Processes of Inference, Learning and Discovery. MIT Press, Cambridge (1986)

Hurst, J., Bull, L.: A Self-Adaptive Classifier System. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 2000. LNCS (LNAI), vol. 1996, pp. 70–79. Springer, Heidelberg (2001)

Hurst, J., Bull, L.: A Self-Adaptive XCS. In: Lanzi, P.-L., Stolzmann, W., Wilson, S.W. (eds.) Advances in Learning Classifier Systems: Proceedings of the Fourth International Workshop on Learning Classifier Systems, pp. 57–73. Springer, Heidelberg (2002)

O'Hara, T., Bull, L.: Building Anticipations in an Accuracy-based Learning Classifier System by use of an Artificial Neural Network. In: Proceedings of the IEEE Congress on Evolutionary Computation, pp. 2046–2052. IEEE Press, Los Alamitos (2005)

Rechenberg, I.: Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog (1973)

Riolo, R.: Lookahead Planning and Latent Learning in a Classifier System. In: Meyer, J.-A., Wilson, S.W. (eds.) From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behaviour, pp. 316–326. MIT Press, Cambridge (1991)

Seward, J.P.: An Experimental Analysis of Latent Learning. Journal of Experimental Psychology 39, 177–186 (1949)

Stolzmann, W.: Anticipatory Classifier Systems. In: Koza, J.R. (ed.) Genetic Programming 1998: Proceedings of the Third Annual Conference, pp. 658–664. Morgan Kaufmann, San Francisco (1998)

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

Wilson, S.W.: Classifier Fitness Based on Accuracy. Evolutionary Computation 3(2), 149–177 (1995)

Wyatt, D., Bull, L.: A Memetic Learning Classifier System for Describing Continuous-Valued Problem Spaces. In: Krasnagor, N., Hart, W., Smith, J. (eds.) Recent Advances in Memetic Algorithms, pp. 355–396. Springer, Heidelberg (2004)

A Learning Classifier System Approach to Relational Reinforcement Learning

Drew Mellor

School of Electrical Engineering and Computer Science,
The University of Newcastle, Callaghan, 2308, Australia
Tel.: (+612) 4921 6071; Facsimile: (+612) 4921 6929
drew.mellor@newcastle.edu.au

Abstract. This article describes a learning classifier system (LCS) approach to relational reinforcement learning (RRL). The system, FOXCS-2, is a derivative of XCS that learns rules expressed as definite clauses over first-order logic. By adopting the LCS approach, FOXCS-2, unlike many RRL systems, is a general, model-free and "tabula rasa" system. The change in representation from bit-strings in XCS to first-order logic in FOXCS-2 necessitates modifications, described within, to support matching, covering, mutation and several other functions. Evaluation on inductive logic programming (ILP) and RRL tasks shows that the performance of FOXCS-2 is comparable to other systems. Further evaluation on RRL tasks highlights a significant advantage of FOXCS-2's rule language: in some environments it is able to represent policies that are genuinely scalable; that is, policies that are independent of the size of the environment.

1 Introduction

Methods for reinforcement learning and supervised learning typically adopt a representational framework known as the attribute-value framework. However, despite the success of the attribute-value framework on a large variety of problems, researchers have recognised limitations to its expressive power; a more expressive framework is provided by logic languages, such as languages over first-order logic [33, Sect. 10.4]. The field of inductive logic programming (ILP) [34] aims to utilise the expressive power of first-order logic by combining logical representations with inductive methods from the supervised learning paradigm. The successful development of ILP has recently motivated a corresponding trend in reinforcement learning: relational reinforcement learning (RRL) [48,45]. As a young field, many approaches to RRL are yet to be explored; this article presents FOXCS-2,[1] an approach based on the learning classifier system framework (LCS).

Amongst others, three desirable characteristics of reinforcement learning systems are that they: *i*) apply generally to the underlying problem framework, Markov decision processes; *ii*) are model-free; that is, they can learn satisfactory behaviour without a model, partial or complete, of the environment's dynamics;

[1] An earlier, proof-of-concept version of the system appears in [32].

J. Bacardit et al. (Eds.): IWLCS 2006/2007, LNAI 4998, pp. 169–188, 2008.

and *iii*) can learn "tabula rasa" without the need for initialisation with approximate or candidate policies. FOXCS (the version number is dropped hereon in) possesses all three qualities due to the LCS framework upon which it builds. In contrast, most existing RRL systems do not; exceptions are [20,19,23,18,14] of which [19,23,18] generalise using kernels and distance measures instead of using the abstractive properties of variables, thus limiting the expressive power of the logical representation. A demonstration of the utility of variables is provided in Sect. 3.3, where the policies learnt scale up to larger environments without the need for retraining. An additional advantage of the LCS framework is that due to its rule-based approach it produces relatively comprehensible output; this is particularly true of FOXCS because its rule language, first-order logic, can make use of mnemonics.

FOXCS is derived from the accuracy based XCS [49,50], which is perhaps the most significant LCS implementation—it realises the majority of features from Holland's original framework [25] and also overcomes the problem of strong overgeneral rules that had hindered earlier strength-based systems [26]. Recent analysis supports the view that under favourable conditions XCS has an inductive bias towards the discovery of accurate, maximally general rules [9]. Empirical results showing that the system is competitive with other machine learning approaches have also been reported [1,2].

Originally, the XCS system was specified to use bit-string rules but much subsequent research focused on extending the representational capability of the system. For instance, rule languages over continuous spaces [51,44,8,31], fuzzy logic [13], S-expressions [30,29], and even multi-layer neural networks [6] have been incorporated into XCS. This diversity of representations suggests that the framework underlying XCS is largely representationally neutral and not dependent upon one rule language or another. Further evidence for this view can be found in the pressures that drive the inductive process within XCS [10], most of which operate independently of the rule language. For these reasons—performance and language neutrality—XCS is the most suitable LCS upon which to build FOXCS.

The remainder of this article is set out as follows. First, the design of FOXCS is described (Sect. 2); next, FOXCS is empirically evaluated on both ILP and RRL tasks (Sect. 3); finally, some concluding remarks are made (Sect. 4). For conciseness, the article assumes some familiarity with XCS and the syntax of first-order logic. Readers unfamiliar with XCS might like to consult [49] for an introduction or [12] for an algorithmic specification; for first-order logic, [24,36] provide accounts in the context of artificial intelligence and ILP respectively.

2 System Design

FOXCS is a learning classifier system that evolves rules which are definite clauses over first-order logic. The approach taken to the design of FOXCS was to extend XCS by "upgrading" its representation to first-order logic; a conventional approach within ILP (see [38,4,47] amongst many others). The intention behind

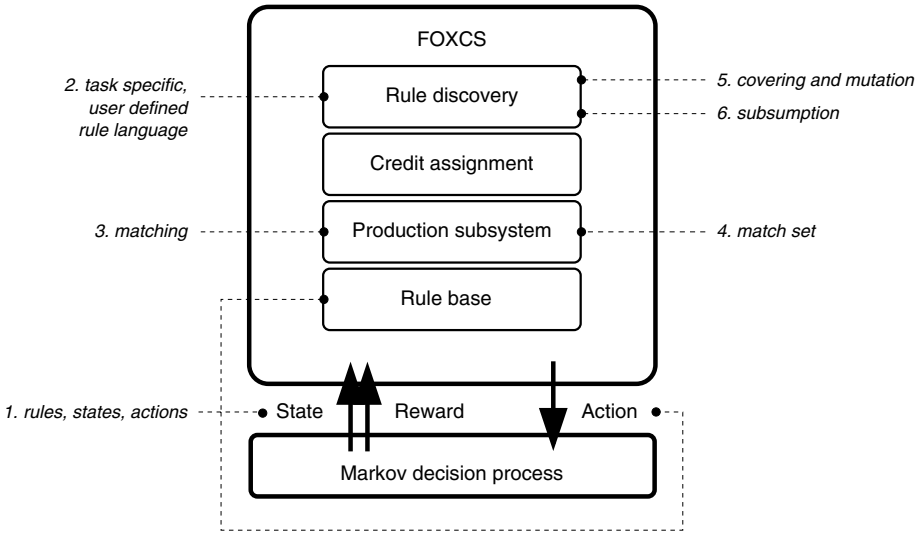


Fig. 1. The architecture of FOXCS with modified components indicated

this approach was that the biases present in XCS, which are the result of substantial innovation and subsequent fine-tuning over many years, would remain largely unaffected and could be directly leveraged by FOXCS. The version of XCS upon which FOXCS is based closely resembles the specification given in [12] with some refinements: the use of tournament selection [11], action set covering [27] and annealing of the learning rate.

Architecturally, the high-level design of FOXCS is identical to that of XCS: a four-tiered system consisting of a rule base, production subsystem, credit assignment subsystem and rule discovery subsystem. However, components within the subsystems have been modified to support representation with first-order logic. Figure 1 shows the high-level architecture of FOXCS and locates the modifications, which are summarised below:

1. Rules, states, and actions are represented by expressions in first-order logic.
2. A task specific rule language is defined by the user of the system using special language declaration commands.
3. The matching operation is redefined for first-order logic.
4. During the operational cycle, the match set is partitioned into non-disjoint subsets, one for each possible action. On any particular cycle an individual rule may belong to multiple subsets.
5. The covering and mutation operations are tailored for producing and adapting rules in first-order logic.
6. The test for subsumption deletion is redefined.

The following subsections describe these modifications in greater detail.

2.1 Representation

The FOXCS system, like its parent XCS, accepts inputs and produces rules; however, unlike XCS these inputs and rules are expressed in a language over first-order logic rather than as bit-strings.

A language in first-order logic, $\mathcal{L}$, is a tuple $\langle \mathcal{C}, \mathcal{F}, \mathcal{P}, d \rangle$, consisting of a set of constant symbols $\mathcal{C}$; a set of function symbols $\mathcal{F}$ (however, functions are not supported by FOXCS, i.e. $\mathcal{F} = \emptyset$); a set predicate symbols $\mathcal{P}$; and an arity function $d : \mathcal{F} \cup \mathcal{P} \rightarrow \mathbb{Z}$, which specifies the number of arguments associated with each element of $\mathcal{F} \cup \mathcal{P}$. For the purpose of describing a task environment, it will be convenient to partition $\mathcal{P}$ into three disjoint subsets, $\mathcal{P}_\mathcal{S}$, $\mathcal{P}_\mathcal{A}$ and $\mathcal{P}_\mathcal{B}$, which contain the predicate symbols for states, actions and background knowledge respectively.

Example 1. The blocks world environment, denoted BW^n, contains a finite collection of n blocks and a floor. Each block is associated with a label—for example, a lower case letter of the alphabet—that uniquely identifies the block. Each block either rests on top of exactly one other block or on the floor. If a block has nothing on top of it then it is said to be "clear", and it can be moved, either to the floor or onto another clear block.

A language for describing the states and actions of BW^n is $\mathcal{L}_\text{BW}$, where $\mathcal{L}_\text{BW} = \langle \mathcal{C}, \emptyset, \mathcal{P}_\mathcal{S} \cup \mathcal{P}_\mathcal{A}, d \rangle$, $\mathcal{P}_\mathcal{S} = \{on, on\_fl, cl\}$, $\mathcal{P}_\mathcal{A} = \{mv, mv\_fl\}$ and $\mathcal{C} = \{a, b, c, \ldots\}$ such that $|\mathcal{C}| = n$. The arity of the predicate symbols are: $d(on) = 2$, $d(on\_fl) = 1$, $d(cl) = 1$, $d(mv) = 2$ and $d(mv\_fl) = 1$.

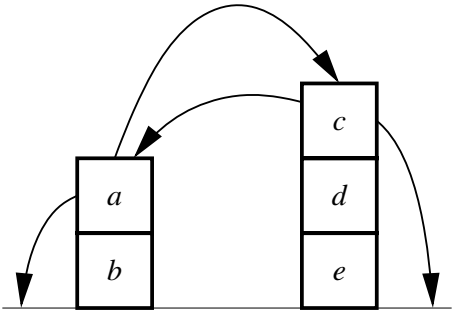
Background Knowledge. As mentioned above, the language $\mathcal{L}$ contains predicates for background knowledge, $\mathcal{P}_\mathcal{B}$, in addition to predicates for describing states, $\mathcal{P}_\mathcal{S}$, and actions, $\mathcal{P}_\mathcal{A}$. A background theory, let us call it $\mathcal{B}$, containing definitions for these predicates is provided by the user of the system. The use of background knowledge is a feature of many ILP and RRL systems, and providing a "good" theory is often the key to solving a task effectively.

Example 2. A useful relation in blocks world is $above(X, Y)$, which is true whenever there is a sequence of blocks leading from X down to Y. A recursive definition for $above(X, Y)$ is:

$$above(X, Y) \leftarrow on(X, Y)$$
$$above(X, Y) \leftarrow on(X, Z), above(Z, Y)$$

Let us extend the language for blocks world so that $\mathcal{L}_\text{BW} = \langle \mathcal{C}, \emptyset, \mathcal{P}_\mathcal{S} \cup \mathcal{P}_\mathcal{A} \cup \mathcal{P}_\mathcal{B}, d \rangle$ where $\mathcal{P}_\mathcal{B} = \{above\}$ and $d(above) = 2$.

The Representation of Inputs. An input to the system describes the current state of the environment and the potential for action within it. More specifically, it is a pair $(s, \mathcal{A}(s))$ where $s \in \mathcal{S}$ is the current state and $\mathcal{A}(s)$ is the set of admissible actions for s. The state, s, and the set of admissible actions, $\mathcal{A}(s)$, are each represented by an Herbrand interpretation—a set of ground atoms (i.e. atoms that do not any contain variables) which are true—over the language $\mathcal{L}$.



$$s = \{cl(a), on(a, b), on\_fl(b), cl(c), on(c, d), on(d, e), on\_fl(e)\}$$
$$\mathcal{A}(s) = \{mv\_fl(a), mv(a, c), mv\_fl(c), mv(c, a)\}$$

Fig. 2. An hypothetical system input

Example 3. Figure 2 shows a diagram and representation of an hypothetical input $(s, \mathcal{A}(s))$ from BW^5 under the language $\mathcal{L}_{\text{BW}}$.

The Representation of Rules. Rules in FOXCS contain a logical part Φ, which replaces the bit-string action and condition of their counterparts in XCS. Apart from this change, all other parameters and estimates associated with rules in XCS are retained by FOXCS and function as they do in XCS.

The logical part of the rule, Φ, is a definite clause over the language $\mathcal{L}$. The definite clauses have the form: $\varphi_0 \leftarrow \varphi_1, \ldots, \varphi_n$ where each φ_i is an atom. Each atom, φ_i, has the form $\rho(\tau_1, \ldots, \tau_{d(\rho)})$, where $\rho \in \mathcal{P}$, each τ_i is either a constant from $\mathcal{C}$ or a variable, and $d(\rho)$ is the arity of ρ. If $d(\rho) = 0$ then ρ is a proposition and the bracketed list of arguments is omitted. The head of the rule, φ_0, can be thought of as the rule's action and only contains predicates from $\mathcal{P}_{\mathcal{A}}$. The rule body, $\varphi_1, \ldots, \varphi_n$, can be thought of as the rule's condition and only contains predicates from $\mathcal{P}_{\mathcal{S}} \cup \mathcal{P}_{\mathcal{B}}$.

Example 4. The logical form of three rules under the language $\mathcal{L}_{\text{BW}}$ are:

$$\Phi_{rl_1}: \quad mv(A, B) \leftarrow cl(A), cl(B), on(B, C), on\_fl(C)$$
$$\Phi_{rl_2}: \quad mv(A, B) \leftarrow cl(A), cl(B)$$
$$\Phi_{rl_3}: \quad mv\_fl(A) \leftarrow cl(A), above(A, B)$$

These three rules are illustrated in Fig. 3.

Note that rules in FOXCS can generalise over actions as well as states, which necessitates a couple of minor changes to the XCS framework. First, as described in Sect. 2.2, it is convenient to match rules against (s, a) pairs in order to determine which actions the rule advocates. Second, the system's match set is partitioned into non-disjoint subsets, one for each $a \in \mathcal{A}(s)$ where s is the current state. The subsets are non-disjoint because a single rule may advocate multiple actions; note that any such rule will contribute to the prediction calculation of each action that it advocates.

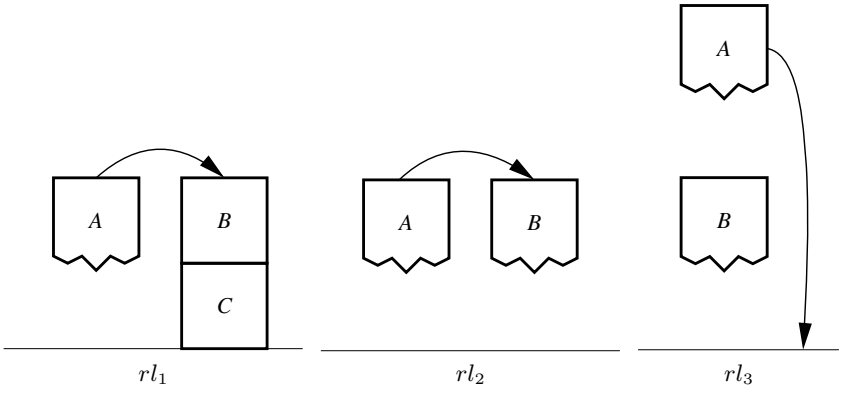


Fig. 3. An illustration of three rules, rl_1, rl_2 and rl_3. A serrated edge signifies that the block rests either on an arbitrarily high stack of blocks or, in some cases, on the floor.

Declaring the Rule Language. Each environment requires its own specific language $\mathcal{L}$; thus FOXCS, like many ILP systems, supports a language declaration command that enables the user to define $\mathcal{L}$. The user defines the predicate symbols belonging to $\mathcal{P}$, and for each $r \in P$ specifies:

- Whether r belongs to $\mathcal{P}_\mathcal{A}$, $\mathcal{P}_\mathcal{S}$ or $\mathcal{P}_\mathcal{B}$
- The minimum and maximum number of atoms allowed in a single rule that are based on r
- Whether an atom based on r may be negated or not
- The type of each argument of r
- Modes for each argument of r; such as whether the argument may be a constant or a free, existing or anonymous variable.

When the system creates and modifies rules through covering and mutation, it consults these declarations in order to determine how atoms may be added, modified, or deleted. The use of type information and modes effectively reduces the size of search space.

2.2 Matching

The matching operation in FOXCS works as follows: a rule rl with logical part Φ successfully matches a ground state-action pair (s, a) if and only if Φ, s, and the background theory $\mathcal{B}$, together entail a. This leads to the following definition of matching:

Definition 1. *A given rule rl with logical part Φ* matches *a ground state-action pair (s, a) under background theory $\mathcal{B}$, if and only if:*

$$\Phi \wedge s \wedge \mathcal{B} \models a \tag{1}$$

Example 5. Let us consider matching rl_3 from Fig. 3 to the input $(s, \mathcal{A}(s))$ illustrated in Fig. 2. The values of Φ, s and $\mathcal{A}(s)$ are reproduced here for convenience:

$$\Phi = mv\_fl(A) \leftarrow cl(A), above(A, B)$$
$$s = \{cl(a), on(a, b), on\_fl(b), cl(c), on(c, d), on(d, e), on\_fl(e)\}$$
$$\mathcal{A}(s) = \{mv\_fl(a), mv(a, c), mv\_fl(c), mv(c, a)\}$$

A separate matching operation is run for each $(s, a) \in \{s\} \times \mathcal{A}(s)$ because matching is defined for (s, a) pairs rather than $(s, \mathcal{A}(s))$ pairs. In this example, rl_3 will be matched to four (s, a) pairs because $|\mathcal{A}(s)| = 4$.

Before matching, the Prolog knowledge base is initialised with $\mathcal{B}$. When matching begins, the first step is to assert Φ and s to the knowledge base. Next, a query is made for each of the actions, $a_1, \ldots, a_4$, in turn. The queries corresponding to a_1 (`?- mv_fl(a)`) and a_3 (`?- mv_fl(c)`) succeed, while the queries for a_2 (`?- mv(a,c)`) and a_4 (`?- mv(c,a)`) fail; hence, rl_3 matches (s, a_1) and (s, a_3), but not (s, a_2) and (s, a_4). Note that the rule in this example has generalised over ground actions.

Note that matching a logical formula is a more expensive operation than the simple linear-time procedure used for matching a bit-string; this represents the cost of increasing the expressive power of the rule language. Experiments have shown that the overall system efficiency on some tasks, particularly ILP tasks, can be significantly improved by associating a cache with each rule that records the results of the rule's previous matching operations (the cache size was 1000 for the experiments in this article).

2.3 Rule Discovery

Rule discovery in FOXCS consists of covering plus various mutation operations; these operations have the same purpose as their counterparts in XCS but have been completely redefined for representation in first-order logic. Crossover was not implemented because a recombination of the syntactic elements of a first-order definite clause generally doesn't produce a recombination effect at the semantic level. New operations, described below, were developed for this version of FOXCS; unlike the previous operations from [32], they are domain independent.

The Covering Operation. The covering operation creates a rule that matches a given state-action pair, (s, a). The covering algorithm firstly creates a new rule, rl, and initialises its parameters (except for Φ) in the same way as XCS. Next, the logical part, Φ, is set to a clause derived from (s, a): $\varphi_0 \leftarrow \varphi_1, \ldots, \varphi_n$ where $\varphi_0 = a$ and $\varphi_1, \ldots, \varphi_n$ is the clause containing all the facts in s. At this stage the rule does not generalise because (s, a) is ground, thus the next step is to generalise the rule by creating and applying an inverse substitution that replaces some or all of the constants with variables. Lastly, rl is inserted in the population where deletion may occur if the number of rules exceeds the user set parameter, N.

The Mutation Operations. Exploration of the rule space in FOXCS is principally achieved by applying mutation operators, which perform a role analogous to refinement in ILP systems. The mutations operations can be characterised along two dimensions. First, they are either generalising or specialising operations. Second, they operate at different levels: either the level of the atoms in the rule or the level of the arguments to the atoms. In addition to the mutation operations, FOXCS also employs a reproduction operation that copies the rule but does not mutate it.

The operations are described below:

Delete atom (DEL, *generalises.*)**.** An atom from the body of the rule is randomly selected and deleted.

Constant to variable (C2V, *generalises.*)**.** Each occurrence of a constant c is replaced with a variable v. The operation first randomly selects the constant c from the set of constants occurring in the rule. It then selects the variable v from the union of the set of variables occurring in the rule and a new variable not already occurring in the rule. An inverse substitution is then performed which replaces c with v.

Variable to anonymous variable (V2A, *generalises.*)**.** This operation randomly selects and replaces a variable in the rule with the anonymous variable.

Add atom (ADD, *specialises.*)**.** An atom is generated and added to the body of the rule. A predicate is selected from $\mathcal{P}_{\mathcal{S}} \cup \mathcal{P}_{\mathcal{B}}$ and its arguments filled in. The arguments may be variables already occurring in the rule, new variables or constants. If a constant is to be assigned to the argument of the predicate, then it is generated from the current state s such that the new rule matches (s, a) for some action $a \in \mathcal{A}(s)$.

Variable to constant (V2C, *specialises.*)**.** Each occurrence of a variable v is replaced with a constant c. The operation first randomly selects the variable v from the set of variables occurring in the rule. It then finds a set C containing constants which are candidates for replacement. Each constant $c' \in C$ is such that the rule j matches (s, a) for the current state s and for some action $a \in \mathcal{A}(s)$, where j is the new rule is obtained by replacing each occurrence of v with c'. The constant c is then selected at random from C.

Anonymous variable to variable (A2V, *specialises.*)**.** An anonymous variable is replaced with a variable, v. The variable v is randomly selected from a set of candidates that include all the variables already occurring in the rule and a new free variable.

Reproduction (REP.)**.** Increments the numerosity of the parent rule. The purpose of this operation to encourage the most highly fit rules to dominate the population.

These descriptions have been simplified by ignoring several factors: the minimum and maximum number of atoms, types, modes and potential failure of the operation (for e.g., the C2V operation is invoked on a rule containing no constants).

When mutation is triggered the system randomly selects one of the above seven operations to apply. Each operation, $i \in$ {DEL,C2V,V2A,ADD,V2C,A2V,REP},

is associated with a weight μ_i, and its selection probability is proportional to its relative weight, $\frac{\mu_i}{\sum_j \mu_j}$. If the operation fails to produce offspring then another randomly selected operation is applied, and so on, until one succeeds. The weighting scheme allows the search to be biased in favour of some operations over others. For example, the search can be biased in favour of the general-to-specific direction by setting $\mu_{\text{ADD}} > \mu_{\text{DEL}}$.

Subsumption Deletion. The subsumption deletion technique encourages an XCS rule-base to converge to a population of maximally general rules; if a rule is sufficiently accurate and experienced, then it may subsume other rules which are specialisations of it. Subsumption deletion may occur in two places: in *GA subsumption* it happens following a call to the GA and before insertion of the new rule into the population, and in *action set subsumption* it is applied directly after the action set is updated. Subsumption deletion requires two rules to be compared for generality: for GA subsumption the relative generality of the parent and child rules can be determined in constant time from knowing which mutation operation was invoked; while for action set subsumption the relative generality of the two rules is tested using θ-subsumption [37]. Ideally, there is an efficiency gain associated with the use subsumption deletion; however, in FOXCS this is potentially offset by the time cost of running θ-subsumption. Experiments have shown that in practice there is a significant improvement in the efficiency of FOXCS when using subsumption deletion.

3 Evaluation

In this section, FOXCS is empirically evaluated in a number of experiments. The first experiment (Sect. 3.1) seeks to verify that FOXCS's novel inductive mechanism, which synthesises evolutionary computation and refinement operators in a unique way, performs effectively. This was achieved by benchmarking FOXCS against several algorithms specialised for induction over first-order logic, ILP algorithms. The second experiment (Sect. 3.2) assesses the system on two RRL benchmarking tasks from the BW^n environment. The final experiment (Sect. 3.3) aims at demonstrating the power FOXCS's rule language for expressing generalisations: its ability to represent policies in BW^n environments that are independent of n, the number of blocks in the environment.

3.1 Comparison to ILP Algorithms

In this section, FOXCS is compared to several well known ILP algorithms. ILP tasks are essentially classification tasks and it has been shown that XCS is competitive with other machine learning algorithms at classification. For instance, [1] compared XCS to six algorithms (each belonging to different a methodological paradigm) on 15 classification tasks and found that it was not significantly outperformed by any of the algorithms in relation to its predicative accuracy; and [7] has also obtained similar results. We are therefore interested to determine whether FOXCS inherits XCS's ability at classification and performs at a level comparable with existing ILP algorithms.

Setup. The experiment compared FOXCS to several ILP algorithms and systems. Four well-known ILP algorithms were selected, FOIL [38], PROGOL [35], ICL [15] and TILDE [4]. The first three systems are rule-based, while TILDE uses a tree-based representation. A fifth system containing an evolutionary component, ECL [17], was also selected; it employs a memetic algorithm that hybridises evolutionary and ILP search heuristics. These algorithms are all supervised learning algorithms since—as far as the author is aware—FOXCS is the first reinforcement learning system to be applied to ILP tasks.

The following three benchmarking data sets were used:

Mutagenesis [43]. The aim is to predict the mutagenic activity of nitroaromatic compounds. In these experiments the 188 molecule "regression friendly" subset was used. There are two different levels of description for the molecules. The first level, NS+S1, primarily contains low level structural descriptions of the molecules in terms of their constituent atoms and bonds. The second level, NS+S2, contains additional information about higher level sub-molecular structures. The second level is a superset of the first and generally allows an increase in predictive accuracy to be attained.

Biodegradability [5]. This data set contains molecules that must be classified as either resistant or degradable from their structural descriptions (which are very similar to those in the Mutagenesis data set) and molecular weights. Several levels of description for the molecules are given in [5]; the Global+R level is used here.

Traffic [22]. The aim is to predict critical road sections responsible for accidents and congestion given traffic sensor readings and road geometry.

All results for the FOXCS system were obtained under the following settings: $N = 1000$, $\epsilon = 10\%$, $\alpha = 0.1$, $\beta = 0.1$, $\epsilon_0 = 0.01$, $\nu = 5$, $\theta_{\text{GA}} = 50$, $\theta_{sub} = 20$, $\theta_{del} = 20$ and $\delta = 0.1$ (these parameters are described in [12]). The mutation parameters were: $\mu_{\text{REP}} = 20$, $\mu_{\text{ADD}} = 60$, $\mu_{\text{DEL}} = 20$, and $\mu_i = 0$ for all $i \in \{\text{C2V}, \text{V2A}, \text{V2C}, \text{A2V}\}$. Tournament selection was used with $\tau = 0.4$. The learning rate was annealed and both GA and action set subsumption were used. The system was trained for 100,000 steps, but rule discovery was switched off after 90,000 steps in order to reduce the disruptive effect of rules generated late in training that have not had sufficient experience to be adequately evaluated (see Fig. 4). Finally, the reward was 10 for a correct classification and -10 for an incorrect classification.

FOXCS's performance was a measure of its predictive accuracy (the percentage of correct classifications made by the system on data withheld from it during training). A typical procedure for measuring predictive accuracy is 10-fold cross-validation; however, care must be taken when using this method with FOXCS because the system is not deterministic and produces different rules when an experiment is re-run, which potentially reduces the reproducibility of a result even when the same data folds are used. All predictive accuracies reported for FOXCS are the mean over ten repetitions of the 10-fold cross validation procedure. Such a measure will minimise the effects of non-determinism and ensure that results are reproducible.

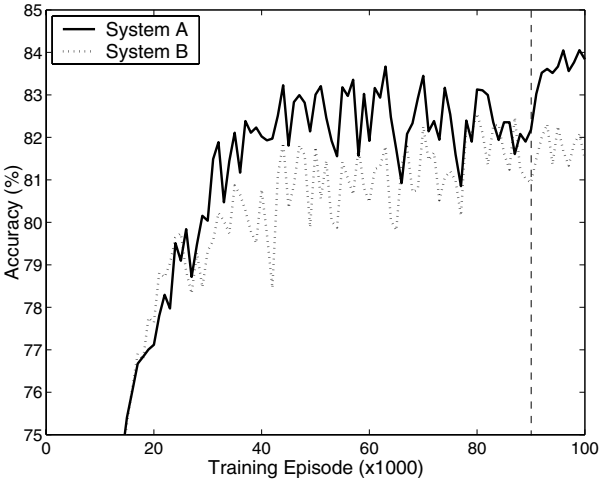


Fig. 4. This graph of FOXCS's performance on the Mutagenesis NS+S1 data set is indicative of the performance improvement gained by switching off rule discovery prior to the completion of training. After 90,000 training episodes rule discovery is switched off for system A but left on for system B. Comparing the performance of the two systems after this point illustrates the negative effect which is exerted by freshly evolved rules.

Results. Table 1 compares the predictive accuracy of FOXCS to that of the ILP systems on the Mutagenesis, Biodegradability, and Traffic data sets. The results were taken from the following sources. For the Mutagenesis data set: [43] for PROGOL, [4] for TILDE and FOIL,[2] [46] for ICL and [16] for ECL; for Biodegradability: [5] for TILDE and ICL, and [16] for ECL; and for Traffic: [22] for all systems except ECL, which is [16]. All predictive accuracies have been measured using 10-fold cross validation (note that, as mentioned above, the predictive accuracy for FOXCS is the mean over ten repetitions of 10-fold cross validation). The folds are provided with the Mutagenesis and Biodegradability data sets but are generated independently for the Traffic data. For the Biodegradability data, five different 10-fold partitions are provided and the final result is the mean performance over the five 10-fold partitions. For consistency, all results have been rounded to the lowest precision occurring in the sources (i.e. whole numbers). Where a source provides multiple results due to different settings of the algorithm, the best results that were obtained are given.

From the table it can be seen that FOXCS generally performs at a level comparable to the other systems; in only one case is FOXCS significantly outperformed (by ECL on Mutagenesis (NS+S2)). This result is compelling because FOXCS is at a relative disadvantage: feedback under reinforcement learning is less informative than under supervised learning. This finding confirms that FOXCS retains

[2] Note that [4] is a secondary source for FOIL on the Mutagenesis data set; this is because the primary source [41] has been withdrawn and its replacement [42] reassesses PROGOL but unfortunately does not replicate the experiments for FOIL.

Table 1. Comparison between the predictive accuracy of FOXCS and selected algorithms on the ILP data sets. The standard deviations, where available, are given in parentheses. An * (**) indicates that the value is significantly different from the corresponding value obtained by FOXCS according to an unpaired t-test (assuming unequal variance) with confidence level 95% (99%) (note that a significance test could not be run for the cases where no standard deviation was reported).

Algorithm	*Predictive Accuracy (%)*			
	Mute (NS+S1)	*Mute (NS+S2)*	*Biodegradability*	*Traffic*
FOXCS	84 (3)	87 (2)	74 (2)	93 (1)
ECL	–	90 (1)**	74 (4)	93 (2)
FOIL	83	82	–	–
ICL	87 (10)	88 (8)	75 (1)	93 (4)
PROGOL	82 (3)	88 (2)	–	94 (3)
TILDE	85	86	74 (1)	94 (4)

XCS's ability at classification; thus, we conclude that FOXCS's novel inductive mechanism performs effectively.

3.2 Relational Reinforcement Learning

The aim of these experiments is to demonstrate that FOXCS can achieve optimal or near-optimal behaviour on RRL tasks. We also wish to compare the performance of FOXCS to other RRL systems. Note that the number of comparison systems is more limited here than in the previous section; this is a consequence of the fact that RRL is a more recent development than ILP.

Setup. For these experiments, FOXCS was benchmarked on two BW^n tasks, *stack* and *onab* [21]. The goal of the *stack* task is arrange all the blocks into a single stack; the order of the blocks in the stack is unimportant. An optimal policy for this task is to move any clear block onto the highest block. In the *onab* task, two blocks are designated A and B respectively; the goal is to place A directly on B. An optimal policy is to first move the blocks above A and B to the floor and then to move A onto B.

Foxcs was run 10 separate times on both the *stack* and *onab* tasks. Each run interleaved training and evaluation: after every 50 training episodes, the system was evaluated on 100 test episodes during which all learning behaviour was switched off. Training episodes began with the blocks in randomly generated positions (positions were generated according to a uniform distribution [39]; but positions corresponding to goal states were discarded) and continued until either a goal state was reached or an upper limit on the number of time steps (set to 100) was exceeded. For test episodes, 100 start starts were randomly generated and used for all test episodes in a single run. The performance measure was the percentage of the 100 test episodes that were completed in the optimal number of steps and all results were averaged over the 10 separate runs.

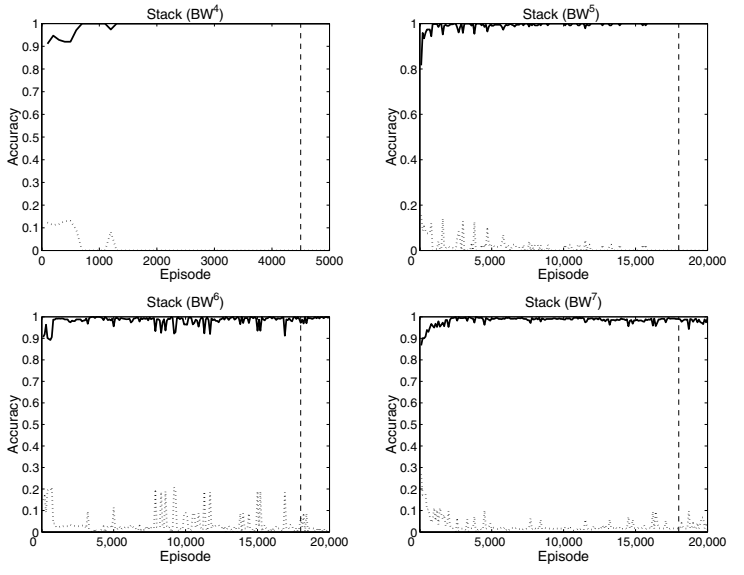


Fig. 5. The performance of FOXCS on the *stack* task in BW$^n|n = 4, 5, 6, 7$ blocks. The graphs show the mean accuracy (solid line) and standard deviation (dotted line) over ten runs. Accuracy is the percentage of episodes completed in the optimal number of steps. The point at which evolution was switched off is indicated by the vertical dashed line.

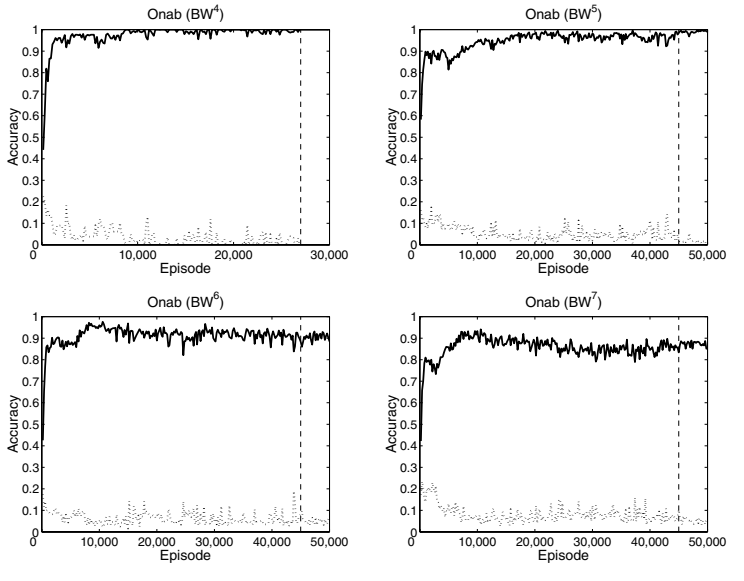


Fig. 6. The performance of FOXCS on the *onab* task in BW$^n|n = 4, 5, 6, 7$ blocks

Results for the FOXCS system were obtained under the following settings. The system parameters were: $N = 1000$, $\epsilon = 10\%$, $\alpha = 0.1$, $\beta = 0.1$, $\epsilon_0 = 0.001$, $\nu = 5$, $\theta_{\text{GA}} = 50$, $\theta_{sub} = 100$, $\theta_{del} = 20$, and $\delta = 0.1$. The mutation parameters were: $\mu_{\text{REP}} = 25$, $\mu_{\text{ADD}} = 50$, $\mu_{\text{DEL}} = 25$, and $\mu_i = 0$ for all $i \in \{\text{C2V}, \text{V2A}, \text{V2C}, \text{A2V}\}$. Proportional selection was used, the learning rate was annealed and GA subsumption was used but not action set subsumption. A reward of -1 was given on each time step. The rule language was based on $\mathcal{L}_{\text{BW}}$.

Covering was not used to generate the initial population. The tasks were "too easy" under covering because the operation alone produces rules that are sufficient to form an optimal policy without any need for mutation. Thus, the rule base was given an initial population consisting of: $mv\_fl(A) \leftarrow cl(A), on(A, B)$ and $mv(A, B) \leftarrow cl(A), cl(B)$. These two rules were selected because between them they cover the entire state-action space of blocks world.

Results. Figures 5 and 6 show the performance of FOXCS on the *stack* and *onab* tasks respectively in $\text{BW}^n | n = 4, 5, 6, 7$. After evolution was switched off, the performance of FOXCS on the *stack* task was optimal in BW^4 and BW^5 and near-optimal in BW^6 and BW^7. Here, near-optimal means that the final accuracy is $\geq 98\%$. For *onab* it was optimal in BW^4 and near-optimal in BW^5 but not for BW^6 and BW^7.

These experiments are most directly comparable to those reported by [21, section 6.3.1] for Q-RRL and by [20, section 4.1] for RRL-TG. In those experiments they addressed both the *stack* and *onab* tasks in $\text{BW}^n | n = 3, 4, 5$. In terms of the accuracy level attained, FOXCS performed at least as well as Q-RRL and RRL-TG, except for the *onab* task in BW^5, where RRL-TG was able to achieve optimal performance. However, Q-RRL and RRL-TG did require significantly fewer training episodes to converge; perhaps because LCS systems train more slowly than other reinforcement learning systems due to the stochastic nature of the evolutionary component.

3.3 Learning Scalable Policies

Note that the performance of FOXCS in BW^n in the previous section was affected by n. Each additional block tends to decrease *i*) the maximum predictive accuracy obtained, and to increase *ii*) the number of training episodes required to

Table 2. Mean execution time (and standard deviation) over 10 runs of 40,000 episodes on the *stack* and *onab* tasks in $\text{BW}^n | n = 4, 5, 6, 7$. For each run, training was interleaved with evaluation: 100 evaluation episodes were run after every 100 training episodes (all start states were selected randomly as described in Sect. 3.2).

Task	*Execution Time (sec)*			
	BW^4	BW^5	BW^6	BW^7
stack	1,578 (383)	7,136 (4,036)	17,336 (1,529)	35,655 (7,946)
onab	7,165 (708)	19,852 (1,987)	42,684 (5,940)	79,450 (7,859)

reach a certain level of performance, *iii*) the size of the population, and *iv*) the average matching time per rule. Table 2 illustrates the net effect on execution time: as n increases from 5 to 7, FOXCS's execution time for BW^n is approximately double that for BW^{n-1}. These effects are not surprising since the size of the state space of BW^n is greater than $O(n!)$ ([39] gives a precise expression).

However, there is an approach, P-Learning [21], that avoids any extra time cost as n increases. Under this approach, training is only performed under small values of n, which minimises training times; importantly however, the resulting policies scale up to arbitrary values of n without requiring additional retraining. The aim of this section is to demonstrate that FOXCS, when combined with P-Learning, can learn optimal policies that scale to arbitrary sized environments.

Setup. For these experiments, P-FOXCS, a version of FOXCS that incorporates P-Learning (see Appendix A), was benchmarked on *stack* and *onab*. The system used the parameters values given in the previous section; however, a number of changes were made to the training and evaluation procedure as set out below.

First, the behavioural policy of the system depended on whether the system was performing a training or evaluation episode. For training episodes, the policy was determined by the primary population as usual. However, during evaluation the behavioural policy was determined by the second population, which contains the rules learnt through P-Learning. As usual, all learning was suspended when the system was performing an evaluation episode.

Second, the number of blocks, n, was varied from episode to episode throughout training in order to encourage the P-Learning component to learn policies that were independent of n. Thus, the initial state of each training episode was randomly selected from BW$^n|n = 3, 4, 5$. (In preliminary experiments, n was held constant throughout training; unfortunately, this resulted in P-Learning policies that were optimal with respect to BW^n only.) It is desirable, though, for the rules in the primary population to actually be *dependent on* n. This was accomplished by placing an additional atom, *num_blocks*(n), into the state signal and adding *num_blocks* to $\mathcal{P}_{\mathcal{S}}$.

Finally, n was also varied from episode to episode throughout evaluation. Before each run, 100 states were randomly selected from BW$^n|n = 3, 4, \ldots, 16$ to be the start states for the test episodes; thus, P-FOXCS was tested on much larger blocks world environments than it was trained on. Note that the P-Learning component was not activated or evaluated until after a delay period (of 5,000 and 20,000 training episodes for *stack* and *onab* respectively) had elapsed; this delay allowed time for the primary population to evolve so that system predictions would be relatively stable by the time P-Learning commenced. After this point, both populations were updated and evolved in parallel.

Results. Figure 7 shows the results of the experiment; by the end of training, the system was able to act optimally on *stack* and near-optimally on *onab* (for *onab* the mean accuracy after evolution was switched off was $\geq 98.6\%$). Recall that most evaluation episodes occurred in worlds containing many more blocks than were present in training episodes; the results thus demonstrate that

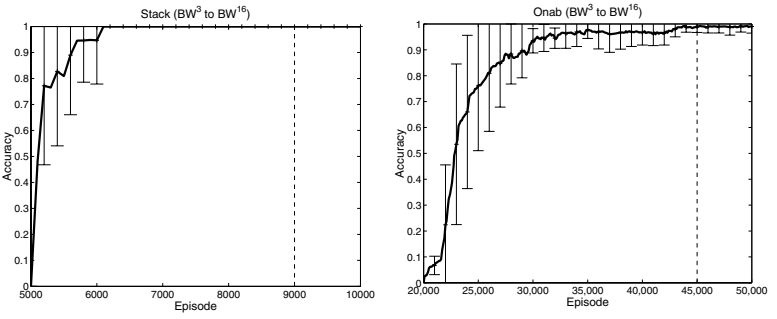


Fig. 7. The performance of P-FOXCS on the *stack* and *onab* tasks. Graphs give performance in terms of accuracy, the percentage of episodes completed in the optimal number of steps; each plot is averaged over ten runs and the dashed line indicates the point at which evolution was switched off.

P-FOXCS had learnt genuinely scalable policies which could handle blocks worlds of effectively arbitrary size.

The problem of learning scalable policies for *stack* and *onab* has been previously addressed in [21,20] using the P-Learning technique. In related experiments, [18] trained several RRL systems on the *stack* and *onab* tasks in $\text{BW}^n|n = 3, 4, 5$ without using the P-Learning extension and then evaluated them in $\text{BW}^n|n = 3, 4, \ldots, 10$. However, the systems were also allowed to learn from samples of optimal trajectories in BW^{10}; so testing did not explicitly show that the policies learnt scaled to environments containing more blocks than previously experienced during training.

Table 3 summarises the results obtained by [21,20,18]. As can be observed in the table, P-FOXCS achieved accuracy results as good as or better than these previous systems. Also shown in the table are the number of episodes used to train the systems. Although all other systems except for RRL-TG† trained on

Table 3. The accuracy of RRL systems on the BW^n tasks *stack* and *onab* where n varied from episode to episode. Note that results were originally presented graphically, therefore all accuracy values (less than 100%) are approximate and should be considered as indicative rather than absolute. Also given are the number of training episodes taken. RRL-TG† used P-Learning; RRL-TG did not. Source for P-RRL is [21], for RRL-TG† [20], and for all other comparison systems [18].

System	*Accuracy* (%)		*Training Episodes* ($\times 10^3$)	
	stack	*onab*	*stack*	*onab*
FOXCS	100%	$\sim$ 98%	20	50
P-RRL	100%	$\sim$ 90%	0.045	0.045
RRL-TG†	100%	$\sim$ 92%	30	30
RRL-TG	$\sim$ 88%	$\sim$ 92%	0.5	12.5
RRL-RIB	$\sim$ 98%	$\sim$ 90%	0.5	2.5
RRL-KBR	100%	$\sim$ 98%	0.5	2.5
TRENDI	100%	$\sim$ 99%	0.5	2.5

significantly less episodes than P-FOXCS, the cost of training is irrelevant when scaling up as an acceptable training time can be produced straightforwardly by selecting a small environment to train in.

It is important to emphasize that the systems evaluated in [18] have not learnt policies that are genuinely independent of the number of blocks—if evaluated in $\text{BW}^n|n > 10$, their performance would most likely degrade. Only P-RRL and RRL-TG are able to express genuinely scalable policies (of course they require P-Learning to actually learn them). On the other hand, RRL-RIB and RRL-KBR do not have sufficient expressive capability to be able to represent the scalable policies because they do not support variables. The TRENDI system, as a hybrid of RRL-RIB and RRL-TG, would also have problems expressing scalable policies. Thus, apart from P-FOXCS, only P-RRL and RRL-TG have the potential to learn policies that are genuinely independent of the number of blocks.

4 Conclusion

This article presents a new RRL system, FOXCS, a derivative of the learning classifier system XCS. The rules learnt by FOXCS are definite clauses over first-order logic, necessitating modifications to support matching, covering, mutation and several other functions. Unlike many RRL systems, the LCS approach adopted by FOXCS is general, model-free and "tabula rasa"; general in the sense that the problem framework, Markov decision processes, is not restricted and "tabula rasa" in the sense that the initial policy can be left unspecified. An additional advantage is that the rules produced are relatively comprehensible.

Experiments verified that the predictive accuracy of FOXCS is comparable to that of several well-known ILP algorithms, thus demonstrating the efficacy of its novel inductive mechanism. Further experiments benchmarked FOXCS on standard RRL tasks; it was able to learn optimal or near-optimal policies. A final experiment highlighted a significant advantage of the use of variables in FOXCS's rule language: in contrast to most of the comparison RRL systems, P-FOXCS—a version of FOXCS extended with P-Learning—learnt genuinely scalable policies (policies which are independent of the size of the environment). In conclusion, these findings establish FOXCS as a promising framework for RRL.

Acknowledgements. The author would like to thank Sašo Džeroski for kindly supplying the Biodegradability data set and Federico Divina and Martin Molina for their assistance in obtaining the Traffic data set.

References

1. Bernadó, E., Llorà, X., Garrel, J.M.: XCS and GALE: A comparative study of two learning classifier systems on data mining. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 2001. LNCS (LNAI), vol. 2321, pp. 115–132. Springer, Heidelberg (2002)
2. Bernadó-Mansilla, E., Garrell-Guiu, J.M.: Accuracy-based learning classifier systems: Models, analysis and applications to classification tasks. Evolutionary Computation 11(3), 209–238 (2003)

3. Beyer, H.-G., O'Reilly, U.-M. (eds.): Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2005. ACM Press, New York (2005)
4. Blockeel, H., Raedt, L.D.: Top-down induction of first-order logical decision trees. Artificial Intelligence 101(1–2), 285–297 (1998)
5. Blockeel, H., Džeroski, S., Kompare, B., Kramer, S., Pfahringer, B., Laer, W.V.: Experiments in predicting biodegradability. Applied Artificial Intelligence 18(2), 157–181 (2004)
6. Bull, L., O'Hara, T.: Accuracy-based neuro and neuro-fuzzy classifier systems. In: Langdon, et al. (eds.) [28], pp. 905–911
7. Butz, M.V.: Rule-based Evolutionary Online Learning Systems: Learning Bounds, Classification, and Prediction. PhD thesis, University of Illinois at Urbana-Champaign, 104 S. Mathews Avenue, Urbana, IL 61801, U.S.A (2004)
8. Martin, V.: Kernel-based, ellipsoidal conditions in the real-valued XCS classifier system. In: Beyer, O'Reilly (eds.) [3], pp. 1835–1842
9. Butz, M.V., Kovacs, T., Lanzi, P.L., Wilson, S.W.: Toward a theory of generalization and learning in XCS. IEEE Transactions on Evolutionary Computation 8(1), 28–46 (2004)
10. Butz, M.V., Pelikan, M.: Analyzing the evolutionary pressures in XCS. In: Spector, et al. (eds.) [40], pp. 935–942
11. Butz, M.V., Sastry, K., Goldberg, D.E.: Strong, stable, and reliable fitness pressure in XCS due to tournament selection. Genetic Programming and Evolvable Machines 6(1), 53–77 (2005)
12. Butz, M.V., Wilson, S.W.: An algorithmic description of XCS. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) Advances in Learning Classifier Systems. Third International Workshop (IWLCS-2000), pp. 253–272. Springer, Heidelberg (2001)
13. Casillas, J., Carse, B., Bull, L.: Fuzzy XCS: an accuracy-based fuzzy classifier system. In: Proceedings of the XII Congreso Espanol sobre Tecnologia y Logica Fuzzy (ESTYLF 2004), pp. 369–376 (2004)
14. Cole, J., Lloyd, J., Ng, K.S.: Symbolic learning for adaptive agents. In: Proceedings of the Annual Partner Conference, Smart Internet Technology Cooperative Research Centre (2003), http://users.rsise.anu.edu.au/~jwl/crc_paper.pdf
15. Raedt, L.D., Laer, W.V.: Inductive constraint logic. In: Jantke, K.P., Shinohara, T., Zeugmann, T. (eds.) ALT 1995. LNCS, vol. 997, pp. 80–94. Springer, Heidelberg (1995)
16. Divina, F.: Hybrid Genetic Relational Search for Inductive Learning. PhD thesis, Department of Computer Science, Vrije Universiteit, Amsterdam, The Netherlands (2004)
17. Divina, F., Marchiori, E.: Evolutionary concept learning. In: Langdon, et al. (eds.) [28], pp. 343–350
18. Driessens, K., Džeroski, S.: Combining model-based and instance-based learning for first order regression. In: Raedt, L.D., Wrobel, S. (eds.) Proceedings of the Twenty-Second International Conference on Machine Learning (ICML 2005). ACM International Conference Proceeding Series, vol. 119, pp. 193–200. ACM Press, New York (2005)
19. Driessens, K., Ramon, J.: Relational instance based regression for relational reinforcement learning. In: Fawcett, T., Mishra, N. (eds.) Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), pp. 123–130. AAAI Press, Menlo Park (2003)
20. Driessens, K., Ramon, J., Blockeel, H.: Speeding up relational reinforcement learning through the use of an incremental first order decision tree learner. In: Raedt, L.D., Flach, P. (eds.) Proceedings of the 12th European Conference on Machine Learning, pp. 97–108. Springer, Heidelberg (2001)

21. Džeroski, S., Raedt, L.D., Driessens, K.: Relational reinforcement learning. Machine Learning 43(1–2), 7–52 (2001)
22. Džeroski, S., Jacobs, N., Molina, M., Moure, C., Muggleton, S., van Laer, W.: Detecting traffic problems with ILP. In: Page, D.L. (ed.) ILP 1998. LNCS, vol. 1446, pp. 281–290. Springer, Heidelberg (1998)
23. Gärtner, T., Driessens, K., Ramon, J.: Graph kernels and Gaussian processes for relational reinforcement learning. In: Horváth, T., Yamamoto, A. (eds.) ILP 2003. LNCS (LNAI), vol. 2835, pp. 146–163. Springer, Heidelberg (2003)
24. Genesereth, M.R., Nilsson, N.J.: Logical Foundations of Artificial Intelligence. Morgan Kaufmann, San Francisco (1987)
25. Holland, J.H.: Adaptation. In: Rosen, R., Snell, F.M. (eds.) Progress in Theoretical Biology, vol. 4. Plenum, NY (1976)
26. Kovacs, T.: Towards a theory of strong overgeneral classifiers. In: Martin, W., Spears, W. (eds.) Foundations of Genetic Algorithms 6, pp. 165–184. Morgan Kaufmann, San Francisco (2001)
27. Kovacs, T.: A Comparison of Strength and Accuracy-Based Fitness in Learning Classifier Systems. PhD thesis, School of Computer Science, University of Birmingham, UK (2002)
28. Langdon, W.B., Cantú-Paz, E., Mathias, K.E., Roy, R., Davis, D., Poli, R., Balakrishnan, K., Honavar, V., Rudolph, G., Wegener, J., Bull, L., Potter, M.A., Schultz, A.C., Miller, J.F., Burke, E.K., Jonoska, N.: GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, 9-13 July 2002. Morgan Kaufmann, San Francisco (2002)
29. Lanzi, P.L.: Extending the representation of classifer conditions, part II: From messy codings to S-expressions. In: Banzhaf, W., Daida, J., Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M., Smith, R.E. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999), pp. 345–352. Morgan Kaufmann, San Francisco (1999)
30. Lanzi, P.L.: Mining interesting knowledge from data with the XCS classifier system. In: Spector, et al. (eds.) [40], pp. 958–965
31. Lanzi, P.L., Loiacono, D., Wilson, S.W., Goldberg, D.E.: XCS with computed prediction in multistep environments. In: Beyer, O'Reilly (eds.) [3], pp. 1859–1866
32. Mellor, D.: A first order logic classifier system. In: Beyer, O'Reilly (eds.) [3], pp. 1819–1826
33. Mitchell, T.M.: Machine Learning. McGraw-Hill, New York (1997)
34. Muggleton, S.: Inductive Logic Programming. In: The MIT Encyclopedia of the Cognitive Sciences (MITECS). Academic Press, London (1992)
35. Muggleton, S.: Inverse entailment and Progol. New Generation Computing, Special issue on Inductive Logic Programming 13(3–4), 245–286 (1995)
36. Nienhuys-Cheng, S.-H., de Wolf, R.: Foundations of Inductive Logic Programming. LNCS, vol. 1228. Springer, Heidelberg (1997)
37. Plotkin, G.D.: Automatic Methods of Inductive Inference. PhD thesis, Edinburgh University (1971)
38. Quinlan, J.R.: Learning logical definition from relations. Machine Learning 5(3), 239–266 (1990)
39. Slaney, J., Thiébaux, S.: Blocks World revisited. Artificial Intelligence 125, 119–153 (2001)
40. Spector, L., Goodman, E.D., Wu, A., Langdon, W.B., Voigt, H.-M., Gen, M., Sen, S., Dorigo, M., Pezeshk, S., Garzon, M.H., Burke, E. (eds.): Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001), July 7-11 2001. Morgan Kaufmann, San Francisco (2001)

41. Srinivasan, A., Muggleton, S., De King, R.: Comparing the use of background knowledge by inductive logic programming systems. In: Raedt, L.D. (ed.) Proceedings of the Fifth International Inductive Logic Programming Workshop, Katholieke Universteit, Leuven (1995); Withdrawn from publication and replaced by [42]
42. Srinivasan, A., King, R.D., Muggleton, S.: The role of background knowledge: using a problem from chemistry to examine the performance of an ILP program. Technical Report PRG-TR-08-99, Oxford University Computing Laboratory, Oxford, UK (1999)
43. Srinivasan, A., Muggleton, S., Sternberg, M.J.E., King, R.D.: Theories for mutagenicity: A study in first-order and feature-based induction. Artificial Intelligence 85(1-2), 277–299 (1996)
44. Stone, C., Bull, L.: For real! XCS with continuous-valued inputs. Evolutionary Computation 11(3), 299–336 (2003)
45. Tadepalli, P., Givan, R., Driessens, K.: Relational reinforcement learning: an overview. In: Tadepalli, P., Givan, R., Driessens, K. (eds.) Proceedings of the ICML2004 Workshop on Relational Reinforcement Learning, pp. 1–9 (2004), http://eecs.oregonstate.edu/research/rrl/index.html
46. Van Laer, W.: From Propositional to First Order Logic in Machine Learning and Data Mining. PhD thesis, Katholieke Universiteit Leuven, Belgium (2002)
47. Van Laer, W., De Raedt, L.: How to upgrade propositional learners to first order logic: A case study. In: Paliouras, G., Karkaletsis, V., Spyropoulos, C.D. (eds.) ACAI 1999. LNCS (LNAI), vol. 2049, pp. 102–126. Springer, Heidelberg (2001)
48. van Otterlo, M.: A survey of reinforcement learning in relational domains. Technical Report TR-CTIT-05-31, University of Twente, The Netherlands (2005)
49. Wilson, S.W.: Classifier fitness based on accuracy. Evolutionary Computation 3(2), 149–175 (1995)
50. Wilson, S.W.: Generalization in the XCS classifier system. In: Koza, J.R., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D.B., Garzon, M.H., Goldberg, D.E., Iba, H., Riolo, R. (eds.) Genetic Programming 1998: Proceedings of the Third Annual Conference, University of Wisconsin, Madison, Wisconsin, USA, pp. 665–674. Morgan Kaufmann, San Francisco (1998)
51. Wilson, S.W.: Get real! XCS with continuous-valued inputs. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 1999. LNCS (LNAI), vol. 1813, pp. 209–222. Springer, Heidelberg (2000)

A P-Learning

This appendix outlines P-Learning [21] and its implementation in P-Foxcs. In brief, a P-Learning system aims to learn a function $P : \mathcal{S} \times \mathcal{A} \rightarrow \{0, 1\}$ such that $(s, a) \mapsto 1$ if a is the optimal action to take in state s, otherwise $(s, a) \mapsto 0$. Because the optimal action is not directly known by the P-Learning agent, it is instead estimated by finding the action with the maximum estimated long term accumulated reward. In P-Foxcs, P-Learning involves maintaining two separate populations of rules. The first population functions as usual; that is, it is used to estimate the long term accumulated reward associated with (s, a) pairs. The second population, on the other hand, is used to estimate P. Rules from the second population are updated and evolved in parallel to and in an analogous fashion to rules from the primary population; the principal difference being that when the rules are updated the training signal originates from the first population instead of the environment.

Linkage Learning, Rule Representation, and the χ-Ary Extended Compact Classifier System

Xavier Llorà[1], Kumara Sastry[2,⋆], Cláudio F. Lima[3], Fernando G. Lobo[3], and David E. Goldberg[2]

[1] National Center for Supercomputer Applications, University of Illinois at Urbana-Champaign, IL 61801, USA
[2] Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, IL 61801, USA
[3] Informatics Laboratory (UALG-ILAB), Dept. of Electronics and Computer Science Engineering, University of Algarve, Campus de Gambelas, 8000-117 Faro, Portugal

Abstract. This paper reviews a *competent* Pittsburgh LCS that automatically *mines* important substructures of the underlying problems and takes problems that were *intractable* with first-generation Pittsburgh LCS and renders them *tractable*. Specifically, we propose a χ-ary extended compact classifier system (χeCCS) which uses (1) a competent genetic algorithm (GA) in the form of χ-ary extended compact genetic algorithm, and (2) a niching method in the form restricted tournament replacement, to evolve a set of maximally accurate and maximally general rules. Besides showing that linkage exist on the multiplexer problem, and that χeCCS scales exponentially with the number of address bits (building block size) and quadratically with the problem size, this paper also explores non-traditional rule encodings. Gene expression encodings, such as the Karva language, can also be used to build χeCCS probabilistic models. However, results show that the traditional ternary encoding `0,1,#` presents a better scalability than the gene expression inspired ones for problems requiring binary conditions.

1 Introduction

One of the daunting challenges in genetics based machine learning (GBML) is the principled integration of *competent* genetic algorithms (GAs) [1]—GAs that solve boundedly difficult problems quickly, reliably, and accurately—for evolving maximally general, maximally accurate rules. Despite their demonstrated scalability—on both problems that are difficult on a single as well as hierarchical level—limited studies have used competent GAs in GBML [2,3]. Butz *et al* [2] studied techniques to identify problem structures in XCS. In a subsequent study, Butz *et al* [3] investigated methods to identify and effectively process *building blocks* in XCS. Specifically, they used a two-level composed problem (n-parity m-multiplexer), and used competent GAs to identify global and local structure. Due to the binary encoding used, the low-level structure (parity) was successfully identified and processed, but left the high-level structure (multiplexer) unidentified.

⋆Currently at Intel Corp.

J. Bacardit et al. (Eds.): IWLCS 2006/2007, LNAI 4998, pp. 189–205, 2008.

Modeling on the evolution of estimation of distribution algorithms (EDAs) [4], we approached the integration of competent GAs into the Pittsburgh LCS in a principled manner by first considering a simplest EDA—compact genetic algorithm [5]—and developed the compact classifier system (CCS) [6]. Our initial analysis with CCS showed that it is able to evolve maximally general and maximally accurate rule sets while guaranteeing a compact representation of the rule set space. However, the scalability analysis of CCS revealed that it requires exponential number of function evaluations to fully solve the multiplexer problem. The exponential scalability is due two factors: (1) *Attribute independence assumption* that leads CCS to require an exponentially large population size to evolve some of the rules belonging to the optimal rule set, which strongly hints to the presence of interactions among the decision variables, and (2) *rule set formation by multiple cGA runs*, that is, CCS requires multiple cGA runs to assemble a rule set that solves the problem. Assuming that the probability of drawing a rule in a given cGA run is $1/|[O]|$ ($|[O]|$ the optimal rule set [7]), the number of runs required to ensemble an rule set increases exponentially with the number of rules in the optimal set $[O]$.

In this paper, we remedy both drawbacks of CCS and propose a method that not only discovers necessary substructures, but also evolves a set of maximally general and maximally accurate rules simultaneously within the framework of Pittsburgh LCS. Specifically, we propose a χ-ary extended compact classifier system (χeCCS) which uses (1) a linkage-learning GA in the form of χ-ary extended compact genetic algorithm (χeCGA) [8,9], and (2) a niching method in the form of restricted tournament replacement [10] to evolve a set of maximally general and maximally accurate rule set. Confirming CCS results, χeCCS results show that linkage does exist in the multiplexer problem, which needs to be discovered in order to evolve a rule set in *tractable* time. We show that in accordance with existing population-sizing models for EDAs, the population size required by χeCCS scales exponentially with the number of address bits (building block size) and linearly with the problem size (number of building blocks). Additionally, the number of function evaluations required by χeCCs to successfully evolve an optimal rule set scales exponentially with the number of address bits (building block size) and quadratically with the problem size, despite the exponential growth in the size of the optimal rule set.

In a quest to push the design envelope, we also explore an alternative rule encoding to the traditional ternary alphabet {`0`,`1`,`#`}. Recently, Wilson [11] has introduced XCSF-GEP. The key modification introduced is the usage of rule conditions *á la* gene expression programming (GEP) [12,13]. Wilson [11] explored GEP encodings to facilitate the expression of arithmetic expressions. His elegant solution allowed XCS to express a rich set of conditions, not possible otherwise. We will follow Wilson's approach, but we will only focus on traditional binary rules—a more simple problem than the originally proposed by Wilson. In this paper we also review the key elements used in XCSF-GEP: the Karva language and its expression mechanism. The Karva encoding provides finite length and finite arity genomes, whereas still can produce variable length rule conditions.

Hence, we can use without modification the proposed χ-eCCS on individuals where its genome follows the Karva encoding scheme. Results show that despite being a feasible approach, the GEP-based encoding places a serious threat to the scalability of χ-eCCS solving the multiplexer problem due to the population sizes required to obtain accurate probabilistic models.

The rest of the paper is structured as follows. We introduce χeCCS in section 2. In section 3, we summarize the initial analysis and results obtained using χeCCS. Section 4 presents a quick overview of gene expression programming and the Karva language, as well as the results and scalability analysis of χeCCS using such encoding. Finally, we summarize the work done and present key conclusions in section 5.

2 The χ-ary Extended Compact Classifier System

The χ-ary extended compact classifier system (χeCCS) relies on a χ-ary extended compact genetic algorithm (χeCGA) [8,9] to identify *building blocks* among the rules. As in CCS, χeCCS uses a default rule for close-world assumption, but represents the rules using a ternary encoding instead of the binary one used in CCS. The use of a χ-ary approach is to focus the linkage learning between the conditions of the rules. Whereas, a binary version would be misled and only group bits of a single condition together (low-level *building blocks*) [3]. Another key element to the evolution of a set of rules is the ability to provide proper niching capabilities—as already pointed out elsewhere by Bernadó-Mansilla et al. [14,15].

The χ-ary extended compact genetic algorithm (χeCGA) [8,9], is an continuations of Harik's binary eCGA [16]. Unlike the original eCGA, χeCGA can handle fixed-length chromosomes composed of genes with arbitrary cardinalities (denoted by χ). As in the original eCGA, χeCGA is based on a key idea that the choice of a good probability distribution is equivalent to linkage learning. The measure of a good distribution is quantified based on minimum description length(MDL) models. The key concept behind MDL models is that given all things are equal, simpler distributions are better than the complex ones. The MDL restriction penalizes both inaccurate and complex models, thereby leading to an optimal probability distribution. The probability distribution used in eCGA is a class of probability models known as marginal product models (MPMs). MPMs are formed as a product of marginal distributions on a partition of the genes. MPMs also facilitate a direct linkage map with each partition separating tightly linked genes.

The χeCCS can be algorithmically outlined as follows:

1. Initialize the population with random individuals.
2. Evaluate the fitness value of the individuals
3. Select good solutions by using s-wise tournament selection without replacement [17].

4. Build the probabilistic model: In χeCGA, both the structure of the model as well as the parameters of the models are searched. A greedy search is used to search for the model of the selected individuals in the population.
5. Create new individuals by sampling the probabilistic model.
6. Evaluate the fitness value of all offspring
7. Replace the parental population (before selection) with the offspring population using restricted tournament replacement (RTR) [10]. We use RTR in order to maintaining multiple maximally general and maximally accurate rules as niches in the population.
8. Repeat steps 3–7 until some convergence criteria are met.

Three things need further explanation: (1) the fitness measure, (2) the identification of MPM using MDL, and (3) the creation of a new population based on MPM.

In order to promote maximally general and maximally accurate rules *á la* XCS [18], χeCCS compute the *accuracy* (α) and the *error* (ε) of an individual [19]. In a Pittsburgh-style classifier, the accuracy may be computed as the proportion of overall examples correctly classified, and the error is the proportion of incorrect classifications issued. Let n_{t+} be the number of positive examples correctly classified, n_{t-} the number of negative examples correctly classified, n_m the number of times that a rule has been matched, and n_t the number of examples available. Using these values, the *accuracy* and *error* of a rule r can be computed as:

$$\alpha(r) = \frac{n_{t+}(r) + n_{t-}(r)}{n_t} \tag{1}$$

$$\varepsilon(r) = \frac{n_{t+}}{n_m} \tag{2}$$

We note that the error (equation 2) only takes into account the number of correct positive examples classified. This is due to the close-world assumption of the knowledge representation which follows from using a default rule. Once the *accuracy* and *error* of a rule are known, the fitness can be computed as follows.

$$f(r) = \alpha(r) \cdot \varepsilon(r) \tag{3}$$

The above fitness measure favors rules with a good classification accuracy and a low error, or maximally general and maximally accurate rules. In another words, rules that cover the maximum number of examples without covering examples of the non-described class.

The identification of MPM in every generation is formulated as a constrained optimization problem,

$$\text{Minimize} \quad C_m + C_p \tag{4}$$

$$\text{Subject to}$$

$$\chi^{k_i} \leq n \quad \forall i \in [1, m] \tag{5}$$

where C_m is the model complexity which represents the cost of a model and is given by

$$C_m = \log_\chi(n+1)\sum_{i=1}^{m}\left(\chi^{k_i}-1\right) \tag{6}$$

and C_p is the compressed population complexity which represents the cost of using a simple model as against a complex one and is evaluated as

$$C_p = \sum_{i=1}^{m}\sum_{j=1}^{\chi^{k_i}} N_{ij}\log_\chi\left(\frac{n}{N_{ij}}\right) \tag{7}$$

where χ is the alphabet cardinality, m in the equations represent the number of BBs, k_i is the length of BB $i \in [1,m]$, n the population size, and N_{ij} is the number of chromosomes in the current population possessing bit-sequence $j \in [1,\chi^{k_i}]$[1] for BB i. The constraint (Equation 5) arises due to finite population size.

The greedy search heuristic used in χ-eCGA starts with a simplest model assuming all the variables to be independent and sequentially merges subsets until the MDL metric no longer improves. Once the model is built and the marginal probabilities are computed, a new population is generated based on the optimal MPM as follows, population of size $n(1-p_c)$ where p_c is the crossover probability, is filled by the best individuals in the current population. The rest $n \cdot p_c$ individuals are generated by randomly choosing subsets from the current individuals according to the probabilities of the subsets as calculated in the model.

One of the critical parameters that determines the success of χeCGA is the population size. Analytical models have been developed for predicting the population-sizing and the scalability of eCGA [20]. The models predict that the population size required to solve a problem with m building blocks of size k with a failure rate of $\alpha = 1/m$ is given by

$$n \propto \chi^k\left(\frac{\sigma_{BB}^2}{d^2}\right)m\log m, \tag{8}$$

where n is the population size, χ is the alphabet cardinality (here, $\chi = 3$), k is the building block size, $\frac{\sigma_{BB}^2}{d^2}$ is the noise-to-signal ratio [21], and m is the number of building blocks. For the experiments presented in this paper we used $k = |a| + 1$ ($|a|$ is the number of address inputs for the multiplexer problem described in the next section), $\frac{\sigma_{BB}^2}{d^2}$=1.5, and $m = \frac{\ell}{|I|}$ (where ℓ is the rule size and $|I|$ the number of inputs of the multiplexer problem).

2.1 Restricted Tournament Replacement

As mentioned earlier, to assemble a rule set that describes the concept we need to maintain multiple maximally accurate and maximally general rules. Since we

[1] Note that a BB of length k has χ^k possible sequences where the first sequence is $00\cdots0$ and the last sequence $(\chi-1)(\chi-1)\cdots(\chi-1)$.

would like to maintain multiple maximally accurate, maximally general rules, we need an efficient niching method, that does not adversely affect the quality of the probabilistic models. Therefore, following previous studies in EDAs [22], we use restricted tournament replacement (RTR) [10]. We note that a sub-structural niching method might be better than RTR in stably maintaining multiple niches [23], and it can be readily incorporated into the proposed algorithm. In RTR, each new offspring solution $\mathbf{x}$ is incorporated into the original population using the following three steps: (1) select a random subset $\mathbf{W}$ of size w (called window size) from the original population (before selection), (2) find the solution $\mathbf{y}$ in $\mathbf{W}$ that is most similar to $\mathbf{x}$ (in terms of Euclidean distance), and (3) make a tournament between $\mathbf{x}$ and $\mathbf{y}$ where $\mathbf{x}$ replaces $\mathbf{y}$ if it is better than $\mathbf{y}$. The parameter w is called window size, and a good rule of thumb for setting this parameter is $w \propto \ell$, where ℓ is the problem size [22]. We note that the window size w affects the number of niches that can be maintained by RTR. That is, increasing the window size can potentially increase the number of niches that can be maintained in the population and also increases the probability of maintaining the niches [10,22].

We note that the population size n, affects the success probability of maintaining *all* maximally general, maximally accurate rules, γ. In essence, RTR requires larger population sizes to maintain the global optima for longer time. This is a well understood phenomena of niching methods and has been analyzed by Mahfoud for fitness sharing [24] and is applicable to RTR as well [23]. The minimum population size required by RTR for maintaining at least one copy of all but one maximally general maximally accurate rules in the population is given by [24,23]

$$n \propto \frac{\log\left[\left(1-\gamma^{1/t}\right)/n_{opt}\right]}{\log\left[\left(n_{opt}-1\right)/n_{opt}\right]} \tag{9}$$

where t is the number of generations we need to maintain all the niches, n_{opt} is the total number of maximally general maximally accurate rules.

3 Results

We conducted a set of initial tests of the χeCCS to evaluated whether it is capable of: (1) identifying and exploit problem structure, and (2) co-evolving a set of rules in a single run. Specifically, we used the multiplexer problem [18] with 3, 6, 11, 20, 37, and 70 inputs.

3.1 Substructure in the Multiplexer

We begin by investigating whether χeCCS is able to *mine* the substructures of the multiplexer problem. The results show that χeCGA does indeed discover important substructures, and the models for each of the multiplexer problems are shown in Table 1. From the models shown in Table 1, we can clearly see that the *building-block* size grows linearly with the number of address bits ($|a|$). Since the maximally accurate and maximally general rules specify $|a|+1$ positions in the rule, we can intuitively expect the building block size to grow with the problem size.

Table 1. Illustrative models evolved by eCCS for different sizes of the multiplexer problem. The number in parenthesis shows the average number of maximally accurate and maximally general rules evolved after 10 independent runs.

3-input multiplexer (3)
`[0 2] [1] (3)`
`01#:1, 1#1:1, #11:1`
6-input multiplexer (7)
`[0 3][1 4][2 5]`
`001###:1, 0#11##:1`
`01#1##:1, 1###11:1`
`10##1#:1, #01#1#:1`
`11###1:1, #1#1#1:1`
11-input multiplexer (14)
`[0 4 5][1 6 10][2 7 8][3 9]`
`0001#######:1, 00#11######:1`
`001#1######:1, 01###11####:1`
`010##1#####:1, 0#1#1#1####:1`
`011###1####:1, 10#####11##:1`
`100####1###:1, 11#######11:1`
`101#####1##:1, 1#1#####1#1:1`
`110######1#:1, #001###1###:1`
`111#######1:1, ...`
20-input multiplexer (31)
`[0 1 3 4][2 10 11 15][5 13][6][7][8]`
`[9 17][12 18][14][16][19]`
`00001################:1, 1000########1#######:1`
`0001#1###############:1, 1001########1######:1`
`0010##1##############:1, 1010#########1#####:1`
`0011###1#############:1, 1011##########1####:1`
`0100####1############:1, 1100###########1###:1`
`0101#####1###########:1, 1101############1##:1`
`0110######1##########:1, 1110#############1#:1`
`0111#######1#########:1, 1111##############1:1`
...
37-input multiplexer (38)
`[0 1 3 9 15][2 7 11 29 33][4 16 34]`
`[5 20 21][6 8 12 14][10 25 26]`
`[13 18 23 30][17 19 24 31][22 32]`
`[27 35][28][36]`
Rules omitted.
70-input multiplexer (—)
`[0 6 43 64][1 8 23 27][2 22][3 40 50 66][4 5 7 61][9 33 53 55]`
`[10 11 12 32 36 62][13 51 57 68][14 15 16 26 42 45]`
`[17 24 34 59][18 19 44 46 52][20 25 30 58][21 47 49 54]`
`[28 29 41 60][31 37 56 69][35 38 39 48 63 65 67]`
Rules omitted.

3.2 Getting a Set of Rules

We now investigate the niching capability of RTR that permits the evolution of an optimal rule set. Before we present the number of rules evolved in χeCCS, we first calculate the total number of maximally accurate and maximally general rules that exist for a given multiplexer, given a default rule. That is, we can compute the number of rules in [O] [7] and the number of overlapping rules given the number of address inputs $|a|$ as follows:

$$size(|a|) = |[O]| + |[OV]| = 2^{|a|} + |a|\, 2^{|a|-1} = (2 + |a|)2^{|a|-1} \tag{10}$$

For the the 3-input, 6-input, 11-input, 20-input, 37-input, and 70-input multiplexer, the total number of maximally accurate and maximally general rules is 3, 8, 20 ,48, 118, 256 respectively.

However, not all these rules are needed to assemble a ruleset that describes the target concept. For instance, a minimal ensemble is the one provided by $[O]$. The number of maximally accurate and maximally general rules evolved on an average in the ten independent runs are shown in Table 1. The results clearly show that RTR does indeed facilitate the simultaenous evolution of the optimal rule set. We also investigated the scalability of the number of function evalutions required by χeCCS to evolve at least $|[O]|$ rules during a single run. We note that the size of $[O]$ grows exponentially with respect of the number of address inputs. The population size used was the one introduced in the previous section. Figure 1 shows the number of iterations required by χeCCS only grows linearly. Therefore, the number of function evaluations scale exponentially with the number of address bits (building-block size) and quadratically with the problem size

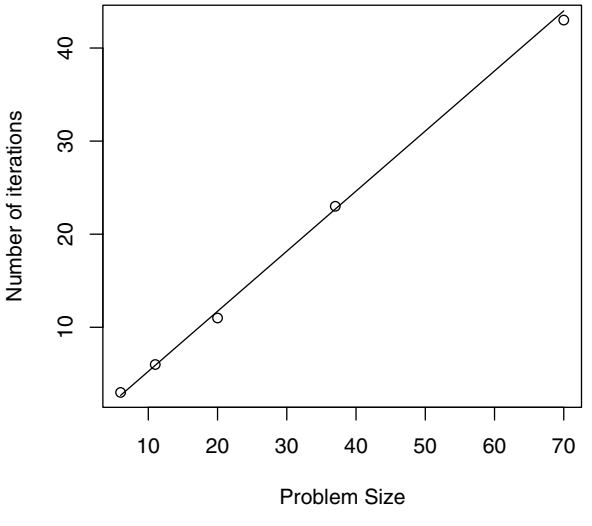


Fig. 1. Number of iterations required to obtain at least $|[O]| = 2^{|a|}$ maximally accurate and maximally general rules. Results are the average of ten independent runs.

(number of building blocks). This is despite the exponential growth in the size of the optimal rule set. Moreover, the scalability results are as predicted by the facetwise models developed for competent GAs.

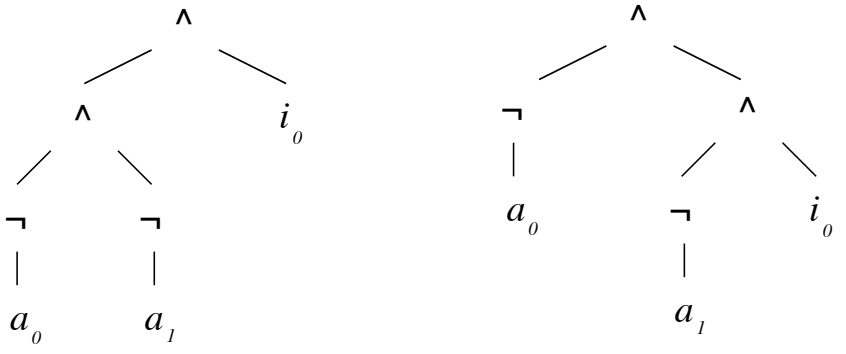
4 Probabilistic Models and Knowledge Representations

Building probabilistic models using rules is quite simple when a finite length and cardinality exists. For instance, the χ-eCCS models a populations of rules encoded using the well-known ternary alphabet {0,1,#} scheme. χ-eCCS shows that when using the proper niching technique (RTR), it can evolve a population of rules, diverse enough, to create an ensemble that solves the problem accurately. Moreover, such probabilistic model building has also been shown as the basis of the estimation of rule fitness—as shown elsewhere [25]. However, the ternary alphabet {0,1,#} can only express rather simple conditions. Other encoding schemes may provide improved expressiveness of condition clauses [26,27], but at the expense of either violating the finite length or cardinality assumptions that make the probabilistic model building possible.

Recently, Wilson [11] introduced XCSF-GEP. The key modification introduced is the usage of rule conditions *á la* gene expression programming (GEP) [12,13]. A complete description of GEP is beyond the scope of this paper. However, we will review the key elements used in XCSF-GEP: the Karva language and its expression mechanism. GEP relies on gene expression mechanisms. Each individual contains one (unigenic individuals) or more genes (multigenic individuals). Wilson [11] explored GEP encodings to facilitate the expression of arithmetic expressions. His elegant solution allowed XCS to express a rich set of conditions, not possible otherwise. We will follow Wilson's approach, but we will only focus on traditional binary rules—a more well defined area than the original Wilson's work. The gene contains a sequence of alleles, which in XCSF-GEP is the basis of the condition obtained by the expression mechanism. When using multigenic individuals, the linking function is the one responsible of combining the expressed genes. The Karva language specifies what symbols can be used in a genome sequence as well as the expression mechanism that translate the genome sequence into a phenotype. A complete description of GEP and the Karva language can be found elsewhere [12,13].

4.1 Gene Expression Programming and the Karva Language

To illustrate how conditions are expressed using the Karva language, we will introduce a simple example to illustrate its main properties. GEP, similarly as genetic programming [28] does, has two distinct elements in its genome: *functionals* and *terminals*. For instance, for the 6-input multiplexer the set of functionals might be $\{\neg, \wedge\}$, whereas the set of terminals could be $\{a_0, a_1, i_0, i_1, i_2, i_3\}$—$a_i$ standing for the i_{th} address line and i_j standing for the j_{th} input line. Using this basic set of functionals and terminals, we can express the same kind of conditions as the ones expressed using the traditional ternary {0,1,#} alphabet. We



(a) First expression tree (b) Second expression tree

Fig. 2. Two possible expression trees for rule `001###:0`. Several other expressions are possible by swapping a_0 and a_1 in each of the trees presented, or by traversing the tree using different orders.

illustrate this point with the following simple example. Given the rule `001###:0`, expressed using the ternary alphabet, Figure 2 presents the 2 possible expression trees for this rule. Figure 2 also shows how the same rule may have multiple expression trees.

However, Figure 2 also illustrates a property of the encoding used by GEP via the Karva language, redundancy at the expression level, as well as at the encoding level. These expressed trees in Figure 2 are the results of the following two genomes: $\{\wedge, \wedge, i_0, \neg, \neg, a_0, a_1, \cdots\}$ and $\{\wedge, \neg, \wedge, a_0, \neg, i_0, a_1, \cdots\}$. The expression mechanism explores the genome from left to right building the expression tree using breadth-first traversal [12,13].

This encoding has also another interesting property. The genome can be split in two different sections: the head and the tail. The head may contain functionals and terminals, whereas the tail only contains terminals. Given the example above, the head needs to be up to 5 symbols in length to be able to encode the proper conditions. Once the length of the head is fixed, the length of the tail is defined as:

$$t = h(n_{max} - 1) + 1 \tag{11}$$

where h is the length of the head, t the length of the tail, and n_{max} is the maximum function arity. Hence, if we fix the length of the head we also fix the length of the gene. GEP [12,13] fixes the maximum length of the head and, thus, it deals with finite length genes. Despite of the fixed length, the gene can still express different size trees. Wilson [11] showed how fixed length genes can provide a rich set of expressible conditions which can be efficiently searched using XCSF-GEP. Thus, the Karva language can provide an elegant way to express variable length conditions while still maintaining finite length and arity genomes.

Multigenic individuals could easily represent a rule set. The linking function would combine the expressed rules together. A possible linking function could be

Table 2. Maximally general and maximally accurate rules for the 6-input multiplexer. The table below shows one of the possible encoding of the rule under the Karva encoding language.

Ternary alphabet encoding	Karva language encoding
`001###:1`	$\{\wedge, \wedge, i_0, \neg, \neg, a_0, a_1, \cdots\}$
`01#1##:1`	$\{\wedge, \wedge, i_1, \neg, a_1, a_0, \cdots\}$
`10##1#:1`	$\{\wedge, \wedge, i_2, a_0, \neg, a_1, \cdots\}$
`11###1:1`	$\{\wedge, \wedge, i_3, a_0, a_1, \cdots\}$
`0#11##:1`	$\{\wedge, \wedge, i_1, \neg, i_0, a_0, \cdots\}$
`1###11:1`	$\{\wedge, \wedge, i_3, a_0, i_2, \cdots\}$
`#01#1#:1`	$\{\wedge, \wedge, i_2, \neg, i_0, a_1, \cdots\}$
`#1#1#1:1`	$\{\wedge, \wedge, i_3, a_1, i_1, \cdots\}$

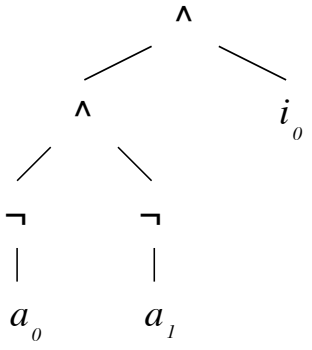
the *or* function, which would lead to individuals to express rule sets normalized in a disjunctive normal form. A more Pittsburgh flavored linking function could also be the decision list, which would be the equivalent of the χ-eCCS final rule set ensembles.

4.2 Building Probabilistic Models for the Karva Language

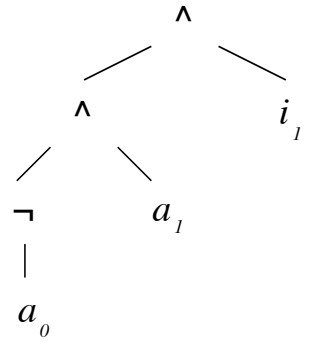
The χ-eCCS relies on the model-building mechanism proposed by eCGA [16]. The Karva encoding presented on the previous section still provides finite length and finite arity genomes. Hence, we can use the proposed χ-eCCS on individuals where its genome follows the Karva encoding scheme. In order to validate such assumption we used unigenic individuals to solve the 6-input multiplexer problem. As introduced above, in order to solve the 6-input multiplexer, the unigenic GEP-like individuals are expressed using the set of functionals $\{\neg, \wedge\}$ and terminals $\{a_0, a_1, i_0, i_1, i_2, i_3\}$—$a_i$ standing for the i_{th} address line and i_j standing for the j_{th} input line. Table 2 shows a possible translation for each of the maximally general and maximally accurate rules for the 6-input multiplexer. Figures 3 and 4 show one of the possible expressed trees (and its genome) for each of the 6-input multiplexer maximally accurate and maximally general rules.

In order to test the viability of such and encoding, we modified the χ-eCCS to evolve unigenic Karva-encoded individuals as described above. We set the length of the head to $h = 5$ (the minimum required to express the more verbose rule `001###`)[2]. Since, the maximum function arity $n_{max} = 2$, the length of the tail is $t = h(n_{max} - 1) + 1 = 5(2-1)+1 = 6$ and, hence, the total length of the unigenic individual is $\ell = h + t = 5 + 6 = 11$. We also set the same population size as the one used for solving the 6-input multiplexer using the {`0,1,#`} ternary alphabet. Using this new encoding χ-eCCS was able to accurately solve the 6-input multiplexer by evolving the eight maximally general and maximally accurate rules required—see Table 2. More interesting were the evolved probabilistic models.

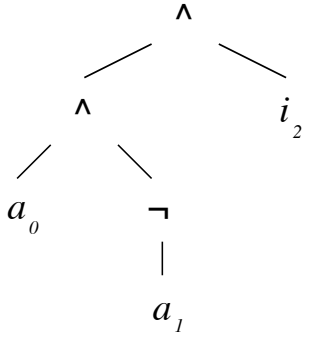
[2] It is important to note here that χ-eCCS relies on a close world assumption, only modeling rules for the positive examples [29,25].



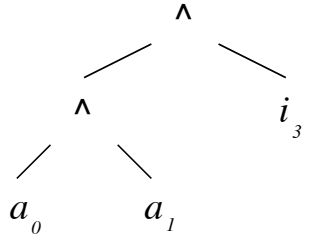
(a) `001###` $\longrightarrow \{\wedge, \wedge, i_0, \neg, \neg, a_0, a_1, \cdots\}$



(b) `01#1##` $\longrightarrow \{\wedge, \wedge, i_1, \neg, a_1, a_0, \cdots\}$



(c) `10##1#` $\longrightarrow \{\wedge, \wedge, i_2, a_0, \neg, a_1, \cdots\}$



(d) `11###1` $\longrightarrow \{\wedge, \wedge, i_3, a_0, a_1, \cdots\}$

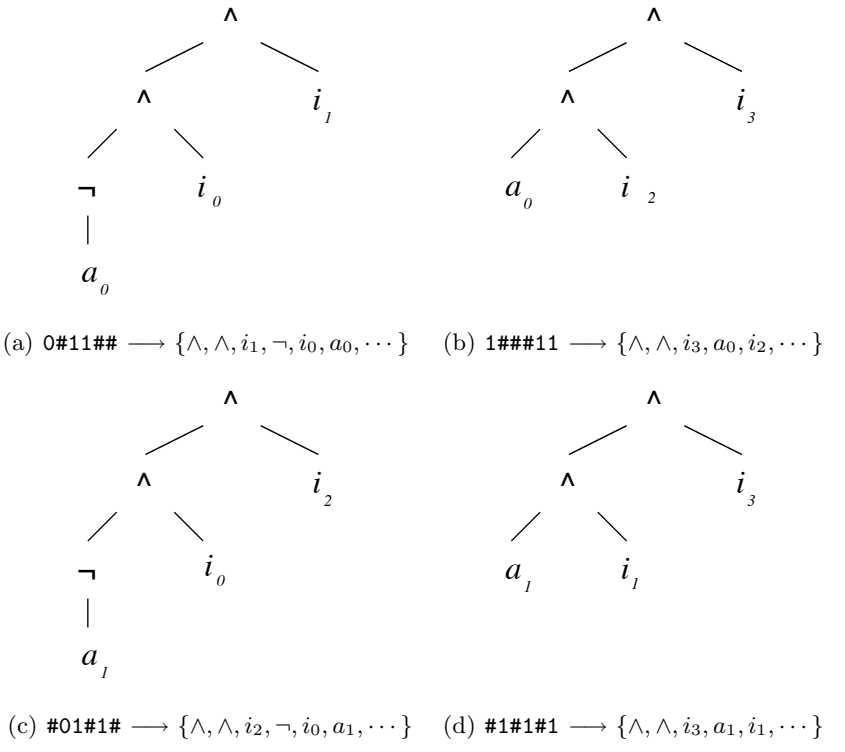
Fig. 3. Figures above show one of the possible expressed trees (and its genome) for each of the 6-input multiplexer non-overlapping maximally accurate and maximally general rules

After checking several runs of χ-eCCS, the dominant model used throughout the evolutionary process was:

```
[0 1][2][3][4][5][6][7][8][9][10]
```

The two first elements of the head were repeatedly grouped together. Reviewing some of the possible Karva-encodings—see Table 2 and Figures 3 and 4—it is clear that for the 6-input multiplexer problem those two head positions define the main context of the rule $(\wedge, \wedge)$[3]. Context identification becomes key to solve the multiplexer problem (express a maximally general and maximally accurate rule). Thus, if the Karva-encoding is also a viable encoding for χ-eCCS allowing it to express arbitrary complex conditions, the open question becomes which one should we choose.

[3] The $\wedge, \wedge$ turned out to be quite a dominant one when compare to the other possible encodings.



(a) `0#11##` $\longrightarrow \{\wedge, \wedge, i_1, \neg, i_0, a_0, \cdots\}$ (b) `1###11` $\longrightarrow \{\wedge, \wedge, i_3, a_0, i_2, \cdots\}$

(c) `#01#1#` $\longrightarrow \{\wedge, \wedge, i_2, \neg, i_0, a_1, \cdots\}$ (d) `#1#1#1` $\longrightarrow \{\wedge, \wedge, i_3, a_1, i_1, \cdots\}$

Fig. 4. Figures above show one of the possible expressed trees (and its genome) for each of the 6-input multiplexer overlapping maximally accurate and maximally general rules

4.3 Rule Representation, Probabilistic Model Building, and Population Sizes

The answer to which of the possible knowledge representations would be more efficient (the ternary {`0,1,#`} or the Karva encoding) can be deduced from the population size required for accurately building probabilistic models. The χ-eCCS model builder relies on marginal probability models (MPM). As shown by Llorà et al. (2006), the population size requirements for an accurate model building follow the same theory as χ-ary eCGA [20,8]—as shown in equation 8. We can rewrite the population sizing model in equation 8 for both rule representations.

Assuming that $\ell_t = k \cdot m$—k being the building block order and m the number of building blocks—and $\ell_t = |a| + 2^{|a|}$—multiplexer problem—for the 6-input multiplexer the populating sizing, shown in equation 9, can be rewritten substituting $k = |a| + 1$ and $m = \ell_t/(|a| + 1)$ as:

$$n_{01\#} \propto 3^{|a|+1} c_t \left(\frac{|a| + 2^{|a|}}{|a| + 1} \right) \log \left(\frac{|a| + 2^{|a|}}{|a| + 1} \right). \tag{12}$$

The same substitutions can be done the Karva encoding. Assuming $k = a$—see the models presented on the previous section for Karva-encoded populations[4]—and $m = \ell_k/a$[5], and the minimum length required of the head can be approximated by $h = a$, hence $\ell_k = |a| + |a|(2-1) + 1 = 2|a| + 1$ the population size, shown in equation 9, rewrites into:

$$n_{Karva} \propto \left(|a| + 2^{|a|} + 2\right)^{|a|} c_k \left(\frac{2|a|+1}{|a|}\right) \log\left(\frac{2|a|+1}{|a|}\right). \tag{13}$$

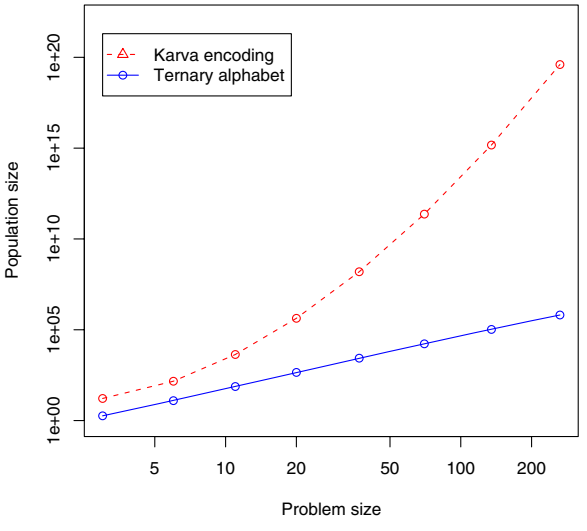


Fig. 5. Asymptotic population-sizing requirements for the ternary alphabet and Karva encodings

Figure 5 presents the asymptotic population-sizing requirements for the ternary alphabet and Karva encodings presented in equations 12 and 13. It is important to note that both axis of the figure are plotted on a logarithmic scale. The population sizing for the Karva encoding grows much faster than the ternary encoding. This is mainly the results of the increased cardinality—from $\chi = 3$ of the ternary encoding to $\chi = |a| + 2^{|a|} + 2$ of the Karva encoding. As the problem grows the Karva-encoding arity grows, whereas the ternary-encoding one remains constant. The growth can be approximated as

$$growth = \frac{n_{Karva}}{n_{01\#}} \propto \frac{1}{3}\left(\frac{\ell+2}{3}\right)^a, \tag{14}$$

[4] The key share element among all the 6-input multiplexer rules is the need for two conjunctions that provide the overall context to bind the two address lines and one input. For this reason, it is reasonable to assume that a lower bound for the building block size is the number of conjunctions required, which in turn is equal to the number of address lines ($|a|$).

[5] Where ℓ_k is determined by the length of the head as shown in equation 11.

being $\ell_t = |a| + 2^{|a|}$. It is clear after equation 14 that the Karva encoding scheme quickly becomes unfeasible due to its extreme growth rate when compared to the traditional ternary {0,1,#} encoding.

5 Conclusions

We have presented how linkage can be successfully identified and exploited to evolve a set of maximally general and maximally accurate rules using Pittsburgh-style learning classifier systems. We introduced the χ-ary extended compact classifier system (χeCCS) which uses (1) a χ-ary extended compact genetic algorithm (χeCGA), and (2) restricted tournament replacement to evolve a set of maximally accurate and maximally general rule set. The results show that linkage exists in the multiplexer problem—confirming CCS results—and also show that in accordance with the facetwise models from GA theory, the number of function evaluations required by χeCCs to successfully evolve an optimal rule set scales exponentially with the number of address bits (building block size) and quadratically with the problem size.

We also explored an alternative rule encoding scheme for multiplexer prolems based on gene expression programming. Unigenic individuals encoded using the Karva language, and its associated expression mechanism, still retain a finite genome length and arity that makes them good candidates for χeCCS probabilistic model building process. However, despite this good preamble, the Karva encoding introduced an explosion on the arity of the individuals. Instead of the quite contained arity $\chi = 3$ of the traditional ternary {0,1,#} encoding, the Karva-encoded individuals present and arity that increased linearly with the problem size, $\chi = 2 + \ell$. As was shown in equation 14 and figure 5, the Karva encoding scheme quickly becomes unfeasible due to its extreme population size growth rate when compared to the traditional ternary {0,1,#} encoding. Thus, although the Karva-encoded individuals still provide a feasible solution for small problems, the traditional ternary {0,1,#} alphabet provides a better scalability when facing larger problems that requires binary rules. However, these results are only applicable to multiplexer problems.The Karva encoding can solve elegantly and efficiently other classes of problems, such at the creation of arithmetic conditions as Wilson showed in XCSF-GEP [11].

Acknowledgments

We would like to thank Stewart W. Wilson for his valuable comments and input during the preparation of this manuscript that helped to greatly improve its quality.

This work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant FA9550-06-1-0370, the National Science Foundation under ITR grant DMR-03-25939 at Materials Computation Center and under grant ISS-02-09199 at the National Center for Supercomputing

Applications, UIUC. The U.S. Government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the National Science Foundation, or the U.S. Government.

References

1. Goldberg, D.E.: The Design of Innovation: Lessons from and for Competent Genetic Algorithms. Kluwer Academic Publishers, Norwell (2002)
2. Butz, M.V., Lanzi, P.L., Llorà, X., Goldberg, D.E.: Knowledge extraction and problem structure identification in XCS. Parallel Problem Solving from Nature - PPSN VIII 3242, 1051–1060 (2004)
3. Butz, M.V., Pelikan, M., Llorà, X., Goldberg, D.E.: Extracted global structure makes local building block processing effective in XCS. In: Proceedings of the 2005 Conference on Genetic and Evolutionary Computation, vol. 1, pp. 655–662 (2005)
4. Pelikan, M., Lobo, F., Goldberg, D.E.: A survey of optimization by building and using probabilistic models. Computational Optimization and Applications 21, 5–20 (2002)
5. Harik, G., Lobo, F., Goldberg, D.E.: The compact genetic algorithm. In: Proceedings of the IEEE International Conference on Evolutionary Computation, pp. 523–528 (1998) (Also IlliGAL Report No. 97006)
6. Llorà, X., Sastry, K., Goldberg, D.E.: The Compact Classifier System: Motivation, analysis, and first results. In: Proceedings of the Congress on Evolutionary Computation, vol. 1, pp. 596–603 (2005)
7. Kovacs, T.: Strength or Accuracy: Credit Assignment in Learning Classifier Systems. Springer, Heidelberg (2003)
8. de la Ossa, L., Sastry, K., Lobo, F.G.: Extended compact genetic algorithm in C++: Version 1.1. IlliGAL Report No. 2006013, University of Illinois at Urbana-Champaign, Urbana, IL (March 2006)
9. Sastry, K., Goldberg, D.E.: Probabilistic model building and competent genetic programming. In: Riolo, R.L., Worzel, B. (eds.) Genetic Programming Theory and Practise, pp. 205–220. Kluwer Academic Publishers, Dordrecht (2003)
10. Harik, G.R.: Finding multimodal solutions using restricted tournament selection. In: Proceedings of the Sixth International Conference on Genetic Algorithms, pp. 24–31 (1995) (Also IlliGAL Report No. 94002)
11. Wilson, S.W.: Classier Conditions Using Gene Expression Programming. IlliGAL Technical Report No. 2008001, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign (2008)
12. Ferreira, C.: Gene expression programming: A new algorithm for solving problems. Complex Systems 13(2), 87–129 (2001)
13. Ferreira, C.: Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence. Springer, Heidelberg (2006)
14. Bernadó-Mansilla, E., Garrell-Guiu, J.M.: MOLeCS: A MultiObjective Learning Classifier System. In: Proceedings of the 2000 Conference on Genetic and Evolutionary Computation, vol. 1, p. 390 (2000)

15. Bernadó-Mansilla, E., Llorà, X., Traus, I.: MultiObjective Learning Classifier System. In: MultiObjective Machine Learning, pp. 261–288. Springer, Heidelberg (2005)
16. Harik, G.R., Lobo, F.G., Sastry, K.: Linkage learning via probabilistic modeling in the ECGA. In: Pelikan, M., Sastry, K., Cantú-Paz, E. (eds.) Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications. Springer, Heidelberg (2006) (Also IlliGAL Report No. 99010)
17. Goldberg, D.E., Korb, B., Deb, K.: Messy genetic algorithms: Motivation, analysis, and first results. Complex Systems 3(5), 493–530 (1989)
18. Wilson, S.: Classifier fitness based on accuracy. Evolutionary Computation 3(2), 149–175 (1995)
19. Llorà, X., Sastry, K., Goldberg, D.E., Gupta, A., Lakshmi, L.: Combating user fatigue in iGAs: Partial ordering, support vector machines, and synthetic fitness. In: GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation, 25-29 June 2005, vol. 2, pp. 1363–1370. ACM Press, Washington (2005)
20. Sastry, K., Goldberg, D.E.: Designing competent mutation operators via probabilistic model building of neighborhoods. In: Proceedings of the Genetic and Evolutionary Computation Conference, vol. 2, pp. 114–125 (2004) (Also IlliGAL Report No. 2004006)
21. Goldberg, D.E., Deb, K., Clark, J.H.: Genetic algorithms, noise, and the sizing of populations. Complex Systems 6, 333–362 (1992)
22. Pelikan, M.: Hierarchical Bayesian Optimization Algorithm: Toward a New Generation of Evolutionary Algorithm. Springer, Berlin (2005)
23. Sastry, K., Abbass, H.A., Goldberg, D.E., Johnson, D.D.: Sub-structural niching in estimation of distribution algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 671–678 (2005)(Also IlliGAL Report No. 2005003)
24. Mahfoud, S.W.: Population size and genetic drift in fitness sharing. Foundations of Genetic Algorithms 3, 185–224 (1994)
25. Llorà, X., Sastry, K., Yu, T.L., Goldberg, D.E.: Do not match, inherit: fitness surrogates for genetics-based machine learning techniques. In: GECCO 2007: Proceedings of the 9th annual conference on Genetic and evolutionary computation, pp. 1798–1805. ACM, New York (2007)
26. Lanzi, P.L.: Extending the Representation of Classifier Conditions Part I: From Binary to Messy Coding. In: Proceedings of the Genetic and Evolutinary Computation Conference (GECCO 1999), pp. 337–344. Morgan Kaufmann, San Francisco (1999)
27. Lanzi, P., Perrucci, A.: Extending the Representation of Classifier Conditions Part II: From Messy Coding to S-Expressions. In: Proceedings of the Genetic and Evolutinary Computation Conference (GECCO 1999), pp. 345–352. Morgan Kaufmann, San Francisco (1999)
28. Koza, J.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
29. Llorà, X., Alías, F., Formiga, L., Sastry, K., Goldberg, D.E.: Evaluation consistency in iGAs: User contradictions as cycles in partial-ordering graphs. In: IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2006), vol. 1, pp. 865–868 (2006) (Also as IlliGAL TR No. 2005022)

Classifier Conditions Using Gene Expression Programming

Invited paper

Stewart W. Wilson

Prediction Dynamics, Concord MA 01742 USA
Department of Industrial and Enterprise Systems Engineering
The University of Illinois at Urbana-Champaign IL 61801 USA
wilson@prediction-dynamics.com

Abstract. The classifier system XCSF was modified to use gene expression programming for the evolution and functioning of the classifier conditions. The aim was to fit environmental regularities better than is typically possible with conventional rectilinear conditions. An initial experiment approximating a nonlinear oblique environment showed excellent fit to the regularities.

1 Introduction

A learning classifier system (LCS) [14] is a learning system that seeks to gain reinforcement from its environment via an evolving population of condition-action rules called classifiers. Broadly, each classifier has a condition, an action, and a prediction of the environmental payoff the system will receive if the system takes that action in an environmental state that satisfies its condition. Through a Darwinian process, classifiers that are useful in gaining reinforcement are selected and propagated over those less useful, leading to increasing system performance.

A classifier's condition is important to this improvement in two senses. First, the condition should contribute to the classifiers's *accuracy*: the condition should be satisfied by, or *match*, only states such that the classifier's action indeed results in the predicted payoff. Second, the condition should be *general* in the sense that the classifier should match as many such states as possible, leading to compactness of the population and, for many applications, transparency of the system's knowledge. In effect, the conditions should match the environment's *regularities*—state subsets with similar action payoffs. This depends in part on the course of the evolutionary process. But it also depends on whether the condition syntax actually permits the regularities to be represented.

Classifier system environments were initially [10] defined over binary domains. The corresponding condition syntax consisted of strings from {1,0,#}, with # a "don't care" symbol matching either 1 or 0. This syntax is effective for conjunctive regularities—ANDs of variables and their negations—but cannot express, e.g., x_1 OR x_2. Later, for real-vector environments, conditions were introduced [20] consisting of conjunctions of interval predicates, where each predicate matches if the corresponding input variable is between a pair of values.

J. Bacardit et al. (Eds.): IWLCS 2006/2007, LNAI 4998, pp. 206–217, 2008.

The same logical limitation also applies—only conjuncts of intervals can be represented. But going to real values exposes the deeper limitation that accurately matchable state subsets must be hyperrectangular, whereas many environmental regularities do not have that shape and so will elude representation by single classifiers.

Attempts to match regularities more adroitly include conditions based on hyperellipsoids [2] and on convex hulls [15]. Hyperellipsoids are higher-dimensional ellipse-like structures that will evolve to align with regularity boundaries. Convex hulls—depending on the number of points available to define them—can be made to fit any convex regularity. Research on both techniques has shown positive results, but hyperellipsoids are limited by being a particular, if orientable, shape, and the number of points needed by convex hulls is exponential with dimensionality.

Further general approaches to condition syntax include neural networks (NNs) [1] and compositions of basis functions—trees of functions and terminals—such as LISP S-expressions [13]. In both cases, matching is defined by the output exceeding a threshold (or equal to 1 (`true`) in the case of S-expressions of binary operators). NNs and S-expressions are in principle both able to represent arbitrary regularities but NNs may not do so in a way that makes the regularity clear, as is desirable in some applications. Moreover, unlike its weights, the NN's connectivity is in most cases fixed in advance, so that every classifier must accept inputs from all variables, whereas this might not be necessary for some regularities. In contrast to NNs, functional conditions such as S-expressions offer greater transparency—provided their complication can be controlled—and have the ability to ignore unneeded inputs or add ones that become relevant.

This paper seeks to advance understanding of functional conditions by exploring the use of gene expression programming [7,8] to define LCS conditions. Gene expression programming (GEP) is partially similar to genetic programming (GP) [11] in that their phenotype structures are both trees of functions and terminals. However, in GEP the phenotype results from translation of an underlying genome, a linear chromosome, which is the object of selection and genetic operators; in GP the phenotype itself acts as the genome and there is no translation step. Previous classifier system work with functional conditions has employed GP [13]. For reasons that will be explained in the following, GEP may offer more powerful learning than GP in a classifier system setting, as well as greater transparency. However, the primary aim of the paper is to test GEP in LCS and assess how well it fits environmental regularities, while leaving direct comparisons with GP for future work.

The next section examines the limits of rectilinear conditions in the context of an example landscape that will be used later on. Section 3 presents basics of GEP as they apply to defining LCS conditions and introduces our test system, XCSF-GEP. Section 4 applies XCSF-GEP to the example landscape. The paper concludes with a discussion of the promise of GEP in LCS and the challenges that have been uncovered.

2 Limits of Traditional Conditions

Classifier systems using traditional hyperrectangular conditions have trouble when the regularities of interest have boundaries that are oblique to the coordinate axes. Because classifier fitness (in current LCSs like XCS [19] and its variants) is based on accuracy, the usual consequence is evolution of a patchwork of classifiers with large and small conditions that cover the regularity, including its oblique boundary, without too much error. Although at the same time there is a pressure toward generality, the system cannot successfully cover an oblique regularity with a single large condition because due to overlap onto adjacent regularities such a classifier will not be accurate. Covering with a patchwork of classifiers is, however, undesirable because the resulting population is enlarged and little insight into the regularity or even its existence is gained.

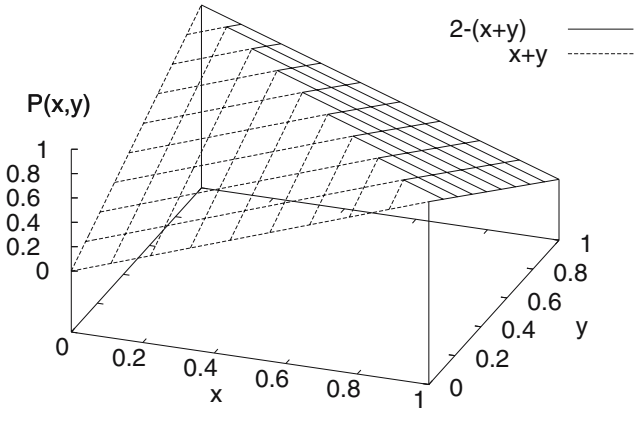


Fig. 1. "Tent" payoff landscape $P(x, y)$

An example will make these ideas clear. Figure 1 shows a tent-like two-dimensional payoff landscape where the projection of each side of the tent onto the x-y plane is a triangle (the landscape is adapted from [25]). The two sides represent different regularities: each is a linear function of x and y but the slopes are different. The equation for the payoff function is

$$P(x,y) = \begin{cases} x+y & : \quad x+y \leq 1 \\ 2-(x+y) & : \quad x+y \geq 1 \end{cases} \tag{1}$$

The landscape of Figure 1 can be learned by XCSF [24] in its function approximation version [21,22]. XCSF approximates non-linear functions by covering the landscape with classifiers that compute local linear approximations to the function's value. Each such classifier will match in a certain subdomain and its prediction will be a linear function—via a weight vector—of the function's input. The classifier's condition evolves and its weight vector is adjusted by feedback until the prediction is within an error criterion of the function's value in

that subdomain. At the same time, the condition will evolve to be as large as possible consistent with the error criterion. In this way XCSF forms a global piecewise-linear approximation to the function (for details of XCSF please see the references).

In the case of Figure 1, XCSF will evolve two sets of classifiers corresponding to the two sides of the tent. All the classifiers on a side will end up with nearly identical weight vectors. But their conditions will form the patchwork described earlier—a few will be larger, located in the interior of the triangle, the rest will be smaller, filling the spaces up to the triangle's diagonal. The problem is: rectangles don't fit triangles, so the system is incapable of covering each side with a single classifier. If the conditions could *be* triangles, the system would need only two classifiers, and they would clearly represent the underlying regularities.

3 Gene Expression Programming in XCSF

3.1 Some Basics of GEP

As noted earlier, in GEP there is both a genome, termed *chromosome*, and a phenotype, termed *expression tree* (ET), with the expression tree derived from the chromosome by a process called *translation*. The ET's performance in the environment determines its fitness and that of the corresponding chromosome, but it is the chromosome which undergoes selection and the actions of genetic operators.

An example chromosome might be `-*ea+b/cdbbaddc`. The arithmetic symbols stand for the four arithmetic *functions*. The alphabetic symbols are called *terminals* and take on numeric values when the expression tree is evaluated. The first seven symbols of this chromosome form the *head*; the rest, all terminals, form the *tail*. Whatever the length of the chromosome, the head, consisting of functions and terminals, and tail, consisting only of terminals, must satisfy

$$t = h(n_{max} - 1) + 1, \tag{2}$$

where h is the length of the head, t the length of the tail, and n_{max} is the maximum function arity (in this case 2). The reason for the constraint implied by this equation will be explained shortly.

Many genetic operators can be employed in GEP. The simplest and, according to Ferreira [8], the most powerful, is *mutation*. It simply changes a symbol to one of the other symbols, with the proviso that, in the head, any symbol is possible, but in the tail, symbols can only be changed to other terminal symbols. Another operator is *inversion*: it picks start and termination symbols of a subsequence within the head, and reverses that subsequence. *Transposition* is a further operator which comes in three forms [7]. The simplest, IS transposition, copies a subsequence of length n from anywhere in the chromosome and inserts it at a random point between two elements of the head; to maintain the same chromosome length, the last n elements of the head are deleted. Several *recombination* operators are used, including one- and two-point, which operate like traditional crossover on a pair of chromosomes.

The translation from chromosome to expression tree is straightforward. Proceeding from left to right in the chromosome, elements are written in a breadth-first manner to form the nodes of the tree. The basic rule is: (1) the first chromosome element forms a single node at the top level (*root*) of the tree; (2) further elements are written left-to-right on each lower level until all the arities on the level above are satisfied; (3) the process stops when a level consists entirely of terminals. Due to Equation 2, the tail is always long enough for translation to terminate before running out of symbols. Figure 2, left side, shows the ET for the example chromosome. Evaluation of the tree occurs from bottom up, in this case implementing the expression $a(b + c/d) - e$.

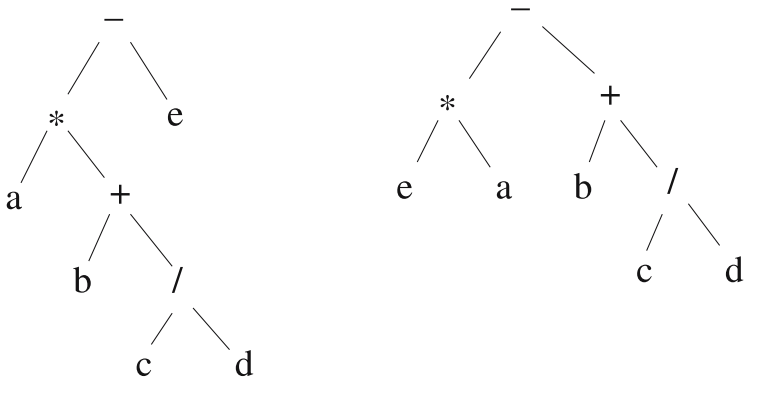


Fig. 2. Translation of chromosome `-*ea+b/cdbbaddc` into expression trees (ETs). Left, standard Karva translation. Right, prefix translation (Sec. 4.1).

The most important property of GEP translation is that every chromosome is *valid*; that is, as long as it obeys Equation 2, every possible chromosome will translate into a legal, i.e., syntactically correct, tree. The reason is that in the translation process all function arities are correctly satisfied. The validity of every chromosome has the significant consequence that the genetic operators cannot produce an illegal result. In fact, as long as it does not leave function symbols in the tail, *any* definable operator will be "safe". This is in some contrast to other functional techniques such as GP, where certain operators cannot be used without producing illegal offspring, or if used, the offspring must be either edited back to legality or discarded. Ferreira ([8], pp. 22-27, 33-34) regards this property of GEP as permitting a more thorough and therefore productive search of the problem space. However, the search issue calls for further investigation since GEP *does* have the restriction that the tail must be free of function symbols and few direct performance comparisons have been made.

Gene expression programming has further important features, but since they were not used in the present work they will only be mentioned here. One is the ability to combine several *genes* into a single chromosome. These are chromosome segments that translate into separate expression trees. They are individually evaluated but their results are *linked* either by a predetermined operator

such as addition or by a linkage structure that is itself evolved in another part of the chromosome. A second feature is that GEP has several methods for the concurrent evolution of real-valued constants that may be needed as coefficients and terms in expressions. Both features (plus others) of GEP are likely to contribute to the representational power of classifier conditions.

Finally, as in other functional approaches, GEP needs a method of dealing with undefined or otherwise undesirable results from arithmetic operators. In contrast to GP, GEP does not replace such operators with protected versions for which the result cannot occur. Instead, normal operators are retained. Then when, say, an operator like "/" receives a zero denominator argument, its tree's evaluation is aborted and the chromosome's fitness is set to zero. The philosophy is to remove (via lack of selection) such flawed chromosomes instead of further propagating their genetic material.

3.2 XCSF-GEP

The application of GEP in classifier conditions is not complicated. Basically, the condition is represented by a chromosome whose expression tree is evaluated by assigning the system's input variables to the tree's terminals, evaluating the tree, and comparing the result with a predetermined threshold. If the result exceeds the threshold, the condition matches and its classifier becomes part of the match set. From that point on, XCSF-GEP differs from XCSF only in that its genetic operators are GEP operators (as in Sec. 3.1). Covering—the case where no classifier matches the input—is handled by repeatedly generating a new classifier with a random condition (but obeying Equation 2) until one matches.

Adding GEP to XCSF would appear to have three main advantages. The first stems from use of functional (instead of rectilinear) conditions, with the consequent ability to represent a much larger class of regularities. The next two stem from GEP in particular: simplicity of genetic operations and conciseness of classifier conditions. Genetic operations are simple in GEP because they operate on the linear chromosome and not on the expression tree itself. They are also simple because offspring never have to be checked for legality, a requirement which in other functional systems can be complex and costly of computation time. However, perhaps the most attractive potential advantage of GEP is that expression tree size is limited by the fixed size of the chromosome.

As noted in the Introduction, a classifier system seeks not only to model an environment accurately, but to do so with transparency, i.e., in a way that offers insight into its characteristics and regularities. It does this by evolving a collection of separate classifiers, each describing a part of the environment that ideally corresponds to one of the regularities, with the classifier's condition describing the part. Thus, it is important that the conditions be concise and quite easily interpretable. For this, GEP would seem to be better than other functional systems such as GP because once a chromosome size is chosen, the expression tree size is limited and very much less subject to "bloat" [17] than GP tree structures are. In GP, crossover between trees can lead to unlimited

tree size unless constrained for example by deductions from fitness due to size. In GEP, the size cannot exceed a linear function ($hn_{max} + 1$) of the head length and no fitness penalty is needed.

4 An Experiment

4.1 Setup

The XCSF-GEP system was tested on the tent landscape of Figure 1. As with XCSF in its function approximation version [21,22], XCSF-GEP was given random x, y pairs from the domain $0.0 \leq x, y \leq 1.0$, together with payoff values equal to $P(x, y)$. XCSF-GEP formed a match set [M] of classifiers matching the input, calculated its system prediction, $\hat{P}$, and the *system error* $|\hat{P} - P(x, y)|$ was recorded. Then, as in XCSF, the predictions of the classifiers in [M] were adjusted using $P(x, y)$, other classifier parameters were adjusted, and a genetic algorithm was run in [M] if called for. In a typical run of the experiment this cycle was repeated 10,000 times, for a total of 20 runs, after which the average system error was plotted and the final populations examined. Runs were started with empty populations.

For classifier conditions, XCSF-GEP used the *function set* {`+ - * / >`} and the *terminal set* {`a b`}. If the divide function "`/`" encountered a zero denominator input the result was set to 1.0 and the fitness of the associated chromosome was set to a very small value. The function "`>`" is the usual "greater than" except the output values are respectively 1 and 0 instead of `true` and `false`. To be added to [M], the evaluation of a classifier's expression tree was required to exceed a *match threshold* of 0.0. In covering and in mutation, the first (root) element of the chromosome was not allowed to be a terminal.

Partially following [21] and using the notation of Butz and Wilson [5], parameter settings for the experiment were: population size $N = 100$, learning rate $\beta = 0.4$, error threshold $\epsilon_0 = 0.01$, fitness power $\nu = 5$, GA threshold $\theta_{GA} = 12$, crossover probability (one point) $\chi = 0.3$, deletion threshold $\theta_{del} = 50$, fitness fraction for accelerated deletion $\delta = 0.1$, delta rule correction rate $\eta = 1.0$, constant x_0 augmenting the input vector $= 0.5$. Prior to the present experiment, an attempt was made to find the best settings (in terms of speed of reduction of error) for β, θ_{GA}, and χ. The settings used in the experiment were the best combination found, with changes to θ_{GA} (basically, the GA frequency) having the greatest effect.

Parameters specific to XCSF-GEP included a mutation rate $\mu = 2.0$. Following Ferreira ([8], p. 77), μ sets the average mutation rate (number of mutations) per chromosome which, divided by the chromosome length, gives the rate per element or allele. A rate of $\mu = 2.0$ has been found by Ferreira to be near-optimal. The head length was set to 6, giving a chromosome length of 13. Inversion and transposition were not used, nor was subsumption since there is no straightforward way to determine whether one chromosome subsumes another.

Besides testing XCSF-GEP as described in this paper, the experiment also tested the same system, but with a different technique for translating the

chromosome. Ferreira calls the breadth-first technique "Karva" whereas it is also possible to translate in a depth-first fashion called "prefix" (see, e.g., [16]). Like Karva, prefix has the property that every chromosome translates to a valid expression tree. Figure 2, right side, shows the translation of the chromosome of Sect. 3.1 using prefix. Looking at examples of chromosomes and their trees, it is possible to see a tendency under prefix more than under Karva for subsequences of the chromosome to translate into compact functional subtrees. This may mean (as Li et al [16] argue) that prefix preserves and propagates functional building blocks better than Karva. We therefore also implemented prefix translation.

4.2 Results

Figure 3 shows the results of an experiment using the parameter settings detailed above, in which XCSF-GEP learned to approximate the tent landscape of Figure 1. For both translation techniques, the error fell to nearly zero, with prefix falling roughly twice as fast as Karva. Still, initial learning was slower than with ordinary XCSF (also shown; relevant parameters the same as for XCSF-GEP). However, XCSF's error performance was markedly worse.

Evolved populations contained about 70 macroclassifiers [5] so that they had definitely not reduced to the ideal of just two, one for each of the tent sides. However, roughly half of the classifiers in a population were accurate and the conditions of roughly half of those precisely covered a tent-side domain. Figure 4 shows the conditions of eight high-numerosity classifiers from four runs of the experiment. The first and second pairs are from two runs in which Karva

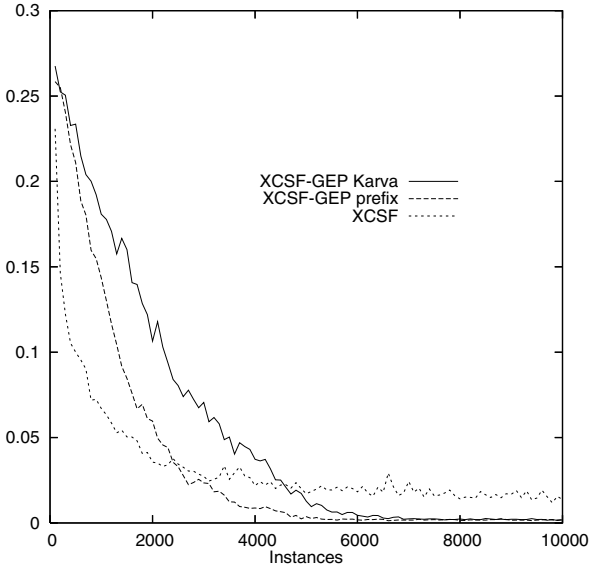


Fig. 3. XCSF-GEP system error vs. learning instances for tent landscape using Karva and prefix translation. Also, system error for XCSF. (Averages of 20 runs).

Chromosome	Algebraic equivalent	Domain
1. (- - * b > + b b b b a b b)	$b-(b+a)b$	$a+b<1$
2. (* - * + / b b a b b b a b)	$((a+b)-1)b^2$	$a+b>1$
3. (> - - / b + a a a a a b a)	$(1-b)>a$	$a+b<1$
4. (> b - > a / b b a b a b a)	$b>((b/a>b)-a)$	$a+b>1$
5. (> - * a / / b b a b a b a)	$(a(1/a)-b)>a$	$a+b<1$
6. (- * * a / + b a b b a b a)	$a((b+a)/b)b-a$	$a+b>1$
7. (- * a + * + b a b a a b a)	$((b+a)b+a)a-a$	$a+b>1$
8. (- - > / / + a b b b b b a)	$((a+b)/b^2>b)-b-a$	$a+b<1$

Fig. 4. Algebraic equivalents and domains of match for high-numerosity chromosomes evolved in the experiment of Section 4. In 1-4, expression trees were formed by Karva translation; in 5-8, by prefix translation.

translation was used; the runs for the third and fourth pairs used prefix translation. With each chromosome is shown the algebraic equivalent of its expression tree, together with the domain in which that tree matched. The relation between the algebra and the domain is: if and only if the inputs satisfy the domain expression, the algebraic expression will compute to a value greater than zero (i.e., the classifier matches).

Many of the expressions simplify easily. For instance if the first is prepended to ">0" (i.e., $b-(b+a)b>0$), the result can be seen to be the same as the domain expression. However, other algebraic expressions are harder to simplify, and it seems clear that for interpretation of XCSF-GEP conditions in general, automated editing is called for. Further, it was quite remarkable how many different but correct algebraic expressions appeared in the total of forty runs of the experiment. Even though many conditions evolved to precisely delineate the two regularities of this environment, each of those regularities clearly has a multitude of algebraic descriptions. XCSF-GEP was prolific in finding these, but the evolutionary process, even in this simple problem, did not reduce them to two, or even a small number.

5 Discussion and Conclusion

The experiment with the tent environment showed it was possible to use GEP for the conditions of XCSF, but the learning was fairly slow and the evolved populations were not compact. However, it was the case—fulfilling one of the main objectives—that the high-numerosity classifiers, if somewhat obscurely, corresponded precisely in their conditions to this environment's oblique regularities.

Many questions ensue. On speed, it must be noted that the experimental system did not use the full panoply of genetic operators, transposition in particular, that are available in GEP, so that search was in some degree limited. Also, the conditions did not have the multigenic structure that is believed important [8]

to efficiency. Nor was GEP's facility for random numerical constants used; the constants needed in the current problem were evolved algebraically, e.g., $1 = b/b$.

The match threshold may matter for speed. If it is set too low, more classifiers match, and in this sense the generality of all classifiers is increased. In effect, the generality of an XCSF-GEP classifier is defined in relation to the match threshold (which could conceivably be adaptive). Since over-generality leads to error, too low a threshold will increase the time required to evolve accurate classifiers. There is a substantial theory [3] of factors, including generality, that affect the rate of evolution in XCS-like systems; it should be applicable here.

On compactness the situation is actually much improved by the fact that regularity-fitting classifiers *do* evolve, in contrast to the poor fit of rectilinear classifiers for all but rectilinear environments. Considerable work (e.g. [23,6,9,18,4]) exists on algorithms that reduce evolved populations down to minimal classifier sets that completely and correctly cover the problem environment. If the population consists of poorly fitting classifiers, the resulting minimal sets are not very small. However, if, as with XCSF-GEP in this experiment, two classifiers are evolved that together cover the environment, compaction methods should produce sets consisting of just those two. Thus XCSF-GEP plus postprocessing compaction (plus condition editing) should go quite far toward the goals of conciseness and transparency. All this of course remains to be tested in practice.

In conclusion, defining classifier conditions using GEP appears from this initial work to lead to a slower evolution than traditional rectilinear methods, but captures and gives greater insight into the environment's regularities. Future research should include implementing more of the functionality of GEP, exploring the effect of the match threshold, testing compaction algorithms, and extending experiments to more difficult environments.

Acknowledgement

The author acknowledges helpful and enjoyable correspondence with Cândida Ferreira.

References

1. Bull, L., O'Hara, T.: Accuracy-based neuro and neuro-fuzzy classifier systems. In: Langdon, et al. (eds.) [12], pp. 905–911
2. Martin, V.: Kernel-based, ellipsoidal conditions in the real-valued XCS classifier system. In: Beyer, H.-G., O'Reilly, U.-M., Arnold, D.V., Banzhaf, W., Blum, C., Bonabeau, E.W., Cantu-Paz, E., Dasgupta, D., Deb, K., Foster, J.A., de Jong, E.D., Lipson, H., Llora, X., Mancoridis, S., Pelikan, M., Raidl, G.R., Soule, T., Tyrrell, A.M., Watson, J.-P., Zitzler, E. (eds.) GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation, 25-29 June 2005, vol. 2, pp. 1835–1842. ACM Press, Washington (2005)
3. Martin, V.: Rule-Based Evolutionary Online Learning Systems: A Principled Approach to LCS Analysis and Design. Springer, Berlin (2006)

4. Butz, M.V., Lanzi, P.L., Wilson, S.W.: Function approximation with XCS: hyperellipsoidal conditions, recursive least squares, and compaction. In: IEEE Transactions on Evolutionary Computation (in press)
5. Butz, M.V., Wilson, S.W.: An Algorithmic Description of XCS. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 2000. LNCS (LNAI), vol. 1996, pp. 253–272. Springer, Heidelberg (2001)
6. Dixon, P.W., Corne, D.W., Oates, M.J.: A ruleset reduction algorithm for the XCS learning classifier system. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 2003. LNCS (LNAI), vol. 2661, pp. 20–29. Springer, Heidelberg (2003)
7. Ferreira, C.: Gene expression programming: a new algorithm for solving problems. Complex Systems 13(2), 87–129 (2001)
8. Ferreira, C.: Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence, 2nd edn. Springer, Germany (2006)
9. Fu, C., Davis, L.: A modified classifier system compaction algorithm. In: Langdon, et al. (ed.) [12], pp. 920–925.
10. Holland, J.H., Reitman, J.S.: Cognitive systems based on adaptive algorithms. In: Waterman, D.A., Hayes-Roth, F. (eds.) Pattern-directed Inference Systems. Academic Press, New York (1978); Reprinted In: Fogel D.B.(ed.). Evolutionary Computation. The Fossil Record. IEEE Press (1998) ISBN: 0-7803-3481-7
11. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
12. Langdon, W.B., Cantú-Paz, E., Mathias, K.E., Roy, R., Davis, D., Poli, R., Balakrishnan, K., Honavar, V., Rudolph, G., Wegener, J., Bull, L., Potter, M.A., Schultz, A.C., Miller, J.F., Burke, E.K., Jonoska, N. (eds.): Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002). Morgan Kaufmann, San Francisco (2002)
13. Lanzi, P.L.: Extending the Representation of Classifier Conditions Part II: From Messy Coding to S-Expressions. In: Banzhaf, W., Daida, J., Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M., Smith, R.E. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999), pp. 345–352. Morgan Kaufmann, San Francisco (1999)
14. Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.): Learning Classifier Systems. From Foundations to Applications. LNCS (LNAI), vol. 1813. Springer, Berlin (2000)
15. Lanzi, P.L., Wilson, S.W.: Using convex hulls to represent classifier conditions. In: Keijzer, M., Cattolico, M., Arnold, D., Babovic, V., Blum, C., Bosman, P., Butz, M.V., Coello, C.C., Dasgupta, D., Ficici, S.G., Foster, J., Hernandez-Aguirre, A., Hornby, G., Lipson, H., McMinn, P., Moore, J., Raidl, G., Rothlauf, F., Ryan, C., Thierens, D. (eds.) GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation, 8-12 July 2006, vol. 2, pp. 1481–1488. ACM Press, Washington (2006)
16. Li, X., Zhou, C., Xiao, W., Nelson, P.C.: Prefix gene expression programming. In: Rothlauf, F. (ed.) Late breaking paper at Genetic and Evolutionary Computation Conference (GECCO 2005), Washington, D.C., USA, 25-29 June (2005)
17. Luke, S.: A comparison of bloat control methods for genetic programming. Evolutionary Computation 14(3), 309–344 (2006)
18. Tamee, K., Bull, L., Pinngern, O.: Towards clustering with XCS. In: Thierens, D., Beyer, H.-G., Bongard, J., Branke, J., Clark, J.A., Cliff, D., Congdon, C.B., Deb, K., Doerr, B., Kovacs, T., Kumar, S., Miller, J.F., Moore, J., Neumann, F., Pelikan, M., Poli, R., Sastry, K., Stanley, K.O., Stutzle, T., Watson, R.A., Wegener, I. (eds.) GECCO 2007: Proceedings of the 9th annual conference on Genetic and

evolutionary computation, London, 7-11 July 2007, vol. 2, pp. 1854–1860. ACM Press, New York (2007)
19. Wilson, S.W.: Classifier Fitness Based on Accuracy. Evolutionary Computation 3(2), 149–175 (1995)
20. Wilson, S.W.: Get Real! XCS with Continuous-Valued Inputs. In: Lanzi, et al. (eds.) [14], pp. 209–219
21. Wilson, S.W.: Function approximation with a classifier system. In: Spector, L., Goodman, E.D., Wu, A., Langdon, W.B., Voigt, H.-M., Gen, M., Sen, S., Dorigo, M., Pezeshk, S., Garzon, M.H., Burke, E. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), 7-11 July 2001, pp. 974–981. Morgan Kaufmann, San Francisco (2001)
22. Wilson, S.W.: Classifiers that approximate functions. Natural Computing 1(2-3), 211–233 (2002)
23. Wilson, S.W.: Compact rulesets from XCSI. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 2001. LNCS (LNAI), vol. 2321, pp. 196–208. Springer, Heidelberg (2002)
24. Wilson, S.W.: Classifier systems for continuous payoff environments. In: Deb, K., Poli, R., Banzhaf, W., Beyer, H.-G., Burke, E., Darwen, P., Dasgupta, D., Floreano, D., Foster, J., Harman, M., Holland, O., Lanzi, P.L., Spector, L., Tettamanzi, A., Thierens, D., Tyrrell, A. (eds.) GECCO 2004. LNCS, vol. 3103, pp. 824–835. Springer, Heidelberg (2004)
25. Wilson, S.W.: Three architectures for continuous action. In: Kovacs, T., Llorà, X., Takadama, K., Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 2003. LNCS (LNAI), vol. 4399, pp. 239–257. Springer, Heidelberg (2007)

Evolving Classifiers Ensembles with Heterogeneous Predictors

Pier Luca Lanzi[1,2], Daniele Loiacono[1], and Matteo Zanini[1]

[1] Artificial Intelligence and Robotics Laboratory (AIRLab),
Politecnico di Milano. P.za L. da Vinci 32, I-20133, Milano, Italy
[2] Illinois Genetic Algorithm Laboratory (IlliGAL),
University of Illinois at Urbana Champaign,
Urbana, IL 61801, USA
lanzi@elet.polimi.it, loiacono@elet.polimi.it, mat.zanini@tiscali.it

Abstract. XCS with computed prediction, namely XCSF, extends XCS by replacing the classifier prediction with a parametrized prediction function. Although several types of prediction functions have been introduced, so far XCSF models are still limited to evolving classifiers with the same prediction function. In this paper, we introduce XCSF with heterogeneous predictors, XCSFHP, which allows the evolution of classifiers with different types of prediction function within the same population. We compared XCSFHP to XCSF on several problems. Our results suggest that XCSFHP generally performs as XCSF with the most appropriate prediction function for the given problem. In particular, XCSFHP seems able to evolve, in each problem subspace, the most adequate type of prediction function.

1 Introduction

XCS with computed prediction [22], namely XCSF, extends the typical idea of classifiers by replacing the prediction parameter with a prediction function $p(s_t, \mathbf{w})$. The prediction function, typically defined as a linear combination of s and $\mathbf{w}$, is used to compute the classifier prediction on the basis of the current state s_t and a parameter vector $\mathbf{w}$ associated to each classifier. Since its introduction, XCSF has been extended with several techniques for computing the classifier prediction, ranging from polynomial functions [11] to Neural Networks [10] and Support Vector Machines [18]. However the experimental results reported so far [12,10,18] suggest that none of such techniques outperforms the others in every respect. In particular, as discussed in [10], a powerful predictor may solve complex problems but may learn more slowly or be unnecessarily expensive in the simple ones. Therefore, the choice of the predictor to use in each problem still requires human expertise and good knowledge of the problem. In this paper we introduce XCSF with heterogeneous predictor, dubbed XCSFHP that extends XCSF by evolving an ensemble of classifiers with different types of prediction function. In XCSFHP, the predictors are not specified at design time, instead the system is in charge of evolving the most adequate predictor for each problem subspace.

J. Bacardit et al. (Eds.): IWLCS 2006/2007, LNAI 4998, pp. 218–234, 2008.

The idea of evolving ensembles of classifiers has been widely studied in the LCS research. Early examples include Dorigo and Colombetti [8] that used a hierarchy of learning classifier systems for controlling an autonomous robot; a similar approach has been used in [1,3] for modeling an economic process. More recently, Llora and Wilson [16] proposed to evolve an ensemble of classifiers with an heterogeneous knowledge representations. Finally, Bull et al. showed in [5,4] that an ensemble of learning classifier systems may learn faster and more reliably than a single one. Our work differs from the previous ones basically in two respects: (i) previous works usually evolve classifiers of the same type (e.g., [5,4]) or focus on different representations (e.g., [16]), while we propose a framework for evolving an ensemble of classifiers with different predictor types; (ii) previous frameworks usually deal with evolving an ensemble of complete problem solutions, while our approach aims to find *one* problem solution by evolving an ensemble of classifiers in each problem subspace.

2 The XCSF Classifier System

XCSF extends XCS in three respects [22]: (i) classifiers conditions are extended for numerical inputs, as done for XCSI [21]; (ii) classifiers are extended with a vector of weights w, that are used to compute classifier's prediction; finally, (iii) the weights w are updated instead of the classifier prediction.

2.1 Classifiers

In XCSF, classifiers consist of a condition, an action, and four main parameters. The condition specifies which input states the classifier matches; it is represented by a concatenation of interval predicates, $int_i = (l_i, u_i)$, where l_i ("lower") and u_i ("upper") are reals (whereas in the original XCSF they were integers [22]). The action specifies the action for which the payoff is predicted. The four parameters are: the weight vector $\mathbf{w}$, used to compute the classifier prediction as a function of the current input; the prediction error ε, that estimates the error affecting classifier prediction; the fitness F that estimates the accuracy of the classifier prediction; the numerosity num, a counter used to represent different copies of the same classifier. The weight vector $\mathbf{w}$ has one weight w_i for each possible input, and an additional weight w_0 corresponding to a constant input x_0, that is set as a parameter of XCSF.

2.2 Performance Component

XCSF works as XCS. At each time step t, XCSF builds a *match set* [M] containing the classifiers in the population [P] whose condition matches the current sensory input s_t; if [M] contains less than θ_{mna} actions, *covering* takes place; covering is controlled by the parameter r_0 and it works as in XCSI [21,22] but considers real values instead of integers. The weight vector w of covering classifiers is initialized to zero; all the other parameters are initialized as in XCS.

For each action a_i in [M], XCSF computes the *system prediction*. As in XCS, in XCSF the *system prediction* of action a is computed by the fitness-weighted average of all matching classifiers that specify action a. In contrast with XCS, in XCSF classifier prediction is computed as a function of the current state s_t and the classifier vector weight w. Accordingly, in XCSF system prediction is a function of both the current state s and the action a. Following a notation similar to [7], the system prediction for action a in state s_t, $P(s_t, a)$, is defined as:

$$P(s_t, a) = \frac{\sum_{cl \in [M]|_a} cl.p(s_t) \times cl.F}{\sum_{cl \in [M]|_a} cl.F} \tag{1}$$

where cl is a classifier, $[M]|_a$ represents the subset of classifiers in [M] with action a, $cl.F$ is the fitness of cl; $cl.p(s_t)$ is the prediction of cl in state s_t, which is computed as:

$$cl.p(s_t) = cl.w_0 \times x_0 + \sum_{i>0} cl.w_i \times s_t(i)$$

where $cl.w_i$ is the weight w_i of cl. The values of $P(s_t, a)$ form the *prediction array*. Next, XCSF selects an action to perform. The classifiers in [M] that advocate the selected action are put in the current *action set* [A]; the selected action is sent to the environment and a reward r is returned to the system together with the next input state s_{t+1}.

2.3 Reinforcement Component

XCSF uses the incoming reward to update the parameters of classifiers in action set $[A]_{-1}$ corresponding to the previous time step, $t-1$. At time step t, the expected payoff P is computed as:

$$P = r_{-1} + \gamma \max_{a \in A} P(s_t, a) \tag{2}$$

where r_{-1} is the reward received at the previous time step. The expected payoff P is used to update the weight vector w of the classifier in $[A]_{-1}$ using a *modified delta rule* [19]. For each classifier $cl \in [A]_{-1}$, each weight $cl.w_i$ is adjusted by a quantity Δw_i computed as:

$$\Delta w_i = \frac{\eta}{|s_{t-1}(i)|^2}(P - cl.p(s_{t-1}))s_{t-1}(i) \tag{3}$$

where η is the correction rate and $|s_{t-1}|^2$ is the norm the input vector s_{t-1}, (see [22] for details). The values Δw_i are used to update the weights of classifier cl as:

$$cl.w_i \leftarrow cl.w_i + \Delta w_i \tag{4}$$

Then the prediction error ε is updated as:

$$cl.\varepsilon \leftarrow cl.\varepsilon + \beta(|P - cl.p(s_{t-1})| - cl.\varepsilon)$$

Finally, classifier fitness is updated as in XCS.

2.4 Discovery Component

In XCSF, the genetic algorithm works as in XCSI [21]; in the version used here the mutation of classifier conditions, controlled by parameter r_0, is based on real values as in [23] instead of integers as in [21].

3 XCSF with Heterogeneous Predictors

XCSF with Heterogeneous Predictors, dubbed XCSFHP, extends XCSF by evolving classifiers with different predictors within the same population. To develop XCSFHP, XCSF has to be modified basically in two respects: (i) when a new classifier is created by the covering operator, the type of its prediction function is randomly chosen among the available ones; (ii) the GA is applied also to the prediction function of classifiers.

3.1 Covering Operator

Extending the covering operator in XCSFHP is straightforward: the predictor type of the new classifier is randomly selected among the available functions. Then the classifier predictor is initialized according to the type and the other classifiers parameters are initialized as in XCSF.

3.2 Discovery Component

In XCSFHP, the genetic algorithm works basically as in XCSF but it is extended also to the prediction function. The selected parent classifiers are copied and with probability χ are recombined as in XCSF. In XCSFHP, after the usual mutation operator is applied to the classifier condition and action, the type of classifier predictor can be mutated, with probability μ, into a new one. In the offspring, the predictor is initialized on the basis of the parent predictor. When the predictor type is the same, i.e., no mutation occurred, the offspring simply inherits the parent predictor. Otherwise the initialization depends on the predictor types involved.

3.3 Predictor Ensembles

In principle, XCSFHP can evolve ensembles of classifiers with any type of predictors. However, in this paper we focus on the two examples of predictor ensembles described in the following.

Polynomial Prediction Functions. As the first predictors ensemble, we used three polynomial prediction functions: a linear, quadratic and a cubic prediction function [11]. Polynomial predictors compute the classifier prediction as a linear combination between a parameter vector w and the input vector or an extended input vector. In particular, the linear predictor has a parameter vector with one weight w_i for each possible inputs and an additional weight w_0 corresponding

to a constant input x_0. Quadratic predictor extends the input vector with one quadratic term for each possible input. Accordingly also the weight vector is extended with one new weight for each inputs introduced. Finally, cubic predictor extends further the input vector and the weight vector adding cubic terms.

The initialization of the offspring predictor is straightforward. When the parent predictor is a function with an higher degree, the offspring predictor is initialized by copying the weight vector of the parent, ignoring the weights corresponding to the higher degree inputs. In the opposite case, the weight vectors is still copied and the new weights (i.e., the one corresponding to the higher degree inputs) are initialized to zero.

Ensembles of Constant, Linear and Neural Predictors. The second ensemble studied involves three really heterogeneous predictor types: (i) a constant prediction, that is the one used in XCS; (ii) a linear prediction function, that is the one used in XCSF; (iii) the neural prediction function, introduced in [10].

In this ensemble the initialization of the offspring predictors is straightforward only when the types involved are the constant and the linear ones; in such case the initialization is performed as previously described for the case of polynomial predictors. When, instead, the neural predictors is involved, we followed a different approach: a training set is built by sampling the inputs matched by the offspring and defining as target the corresponding outputs computed by the parent predictor. Then such a set is used for training the offspring predictor. Notice that this approach is very general and can be applied to manage any type of predictors. On the other hand it might be computationally expensive, requiring a data sampling and a complete training of the predictor. However in XCSFHP, as in XCSF, the mutation operator is applied with a very low probability and therefore does not affect so much the overall computational performance.

4 XCSFHP for Function Approximation

We compared XCSFHP to XCSF on several function approximation problems. In the first set of experiments we focus on XCSFHP with ensembles involving polynomial predictors, i.e. linear, quadratic and cubic predictors. In the second set of experiments we study XCSFHP with the ensemble of predictors involving constant, linear and neural predictors.

All the experiments discussed in this section follow the standard design used in the literature [20,22]. In each experiment the system has to learn to approximate a target function $f(x)$; each experiment consists of a number of problems that the system must solve. For each problem, an example $\langle x, f(x)\rangle$ of the target function $f(x)$ is randomly selected; x is input to the system, which computes the approximated value $\hat{f}(x)$ as the expected payoff of the only available dummy action; the action is virtually performed (the action has no actual effect), and XCSF receives a reward equal to $f(x)$. The system learns to approximate the target function $f(x)$ by evolving a mapping from the inputs to the payoff of the only available action. Each problem is either a *learning* problem or a *test* problem. In *learning* problems, the genetic algorithm is enabled; during *test*

problems it is turned off. Classifier parameters are always updated. The covering operator is always enabled, but operates only if needed. Learning problems and test problems alternate.

The performance of the compared systems is measured as the average system prediction error. In addition, we analyzed the size of the populations evolved as a measure of the systems generalization capabilities. All the experiments reported have been conducted on `xcslib` [9].

4.1 Experiments with Polynomial Predictors

In the first experiment we applied XCSFHP with polynomial predictors to the approximation of the f_{fl} function (see Figure 1a), defined as follows:

$$f_{tl}(x) = \begin{cases} 5x + 2 & \text{for } x < 0.5 \\ 4.5 + 0.1 \cdot (x - 0.5) & \text{for } 0.5 \leq x \leq 1 \end{cases} \tag{5}$$

We compared XCSFHP with the ensemble of polynomial predictors and three versions of XCSF with a single type of predictor, that is respectively with linear prediction, with quadratic prediction and with cubic prediction. Experiments were performed with the following parameters setting: $N = 200$, $\beta = 0.2$, $\delta = 0.2$, $\gamma = 0.7$, $\theta_{GA} = 50$, $\chi = 0.8$, $\mu = 0.04$, $\epsilon_0 = 0.01$, $\nu = 5$, $\alpha = 0.1$, $\theta_{del} = 50$, GA subsumption is on with $\theta_{sub} = 50$; action-set subsumption is not used; $m_0 = 0.5$, $r_0 = 0.5$ [22]; the classifier weight vector is updated using the Recursive Least Squares, with $\delta_{rls} = 100$ [15]. Figure 1c shows that all the three polynomial predictors compared (linear, quadratic and cubic) perform almost in the same way and so does XCSFHP. Figure 1d shows that in simple function, as f_{tl} function, all the predictors compared lead almost to the same population size. However it is interesting to study what type of predictors have been evolved by XCSFHP. To this purpose, Figure 1b shows the frequencies of the matching classifiers types over the whole input space. In the first half of the input space cubic and quadratic predictors are evolved more frequently than the linear ones, especially near slope changing point, where high quadratic and cubic terms can be exploited for providing a smooth approximation. Instead, in the second half of the input space the target function is almost constant and high degree terms of cubic and quadratic predictors may easily cause big prediction error; thus, the linear predictors is evolved more frequently. In summary, on the approximation of a simple function as f_{tl}, all the polynomial predictors provide almost the same performance and the same generalization capabilities. Nonetheless XCSFHP is able to evolve a solution as accurate and compact as the one evolved by XCSF with a single predictor type. In addition we also showed that XCSFHP evolved with higher probability the predictors type most suitable for approximating the target function in each subspace of the input domain.

In the second experiment we applied XCSF and XCSFHP systems to the more complex function f_{abs} (see Figure 2a) introduced in [11] and defined as follows:

$$f_{abs}(x) = |\sin(2\pi x) + |\cos(2\pi x)|| \tag{6}$$

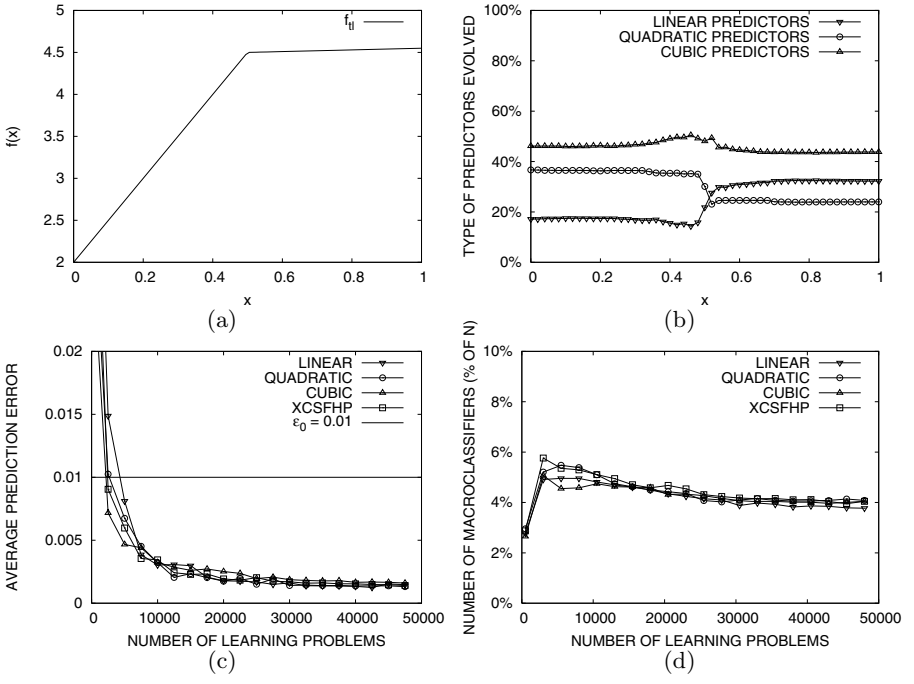


Fig. 1. XCSFHP and XCSF with linear, quadratic and cubic predictors applied to $f_{tl}(x)$ function: (a) the target function, (b) frequencies of predictor types evolved, (c) performance, and (d) population size. Curves are averages over 50 runs.

We used the same parameters setting of the previous experiment except for $N = 800$. Figure 2c shows that all the systems evolve an accurate approximation, but XCSF with linear prediction is slightly slower to converge than XCSF with both quadratic and cubic predictors. On the other hand, XCSFHP converges as fast as XCSF with quadratic and cubic predictors. For what concerns the generalization capabilities, Figure 2d shows that XCSF with linear predictors evolves a slightly bigger population, than XCSF with quadratic and cubic predictors. Notice that XCSFHP evolves a population as compact as the ones evolved by XCSF with quadratic and cubic predictors. It is worthwile to see what type of predictors are evolved by XCSFHP. Figure 2b shows the frequencies of the three predictor types evolved over the whole input domain. As expected, the cubic and quadratic predictors are more frequently evolved being the most suitable for approximating the target function as previously showed by the analysis of the performance and the generalization. In particular it is interesting to underline that the linear predictors are almost always not present in the final population, except for a few ones evolved for $x > 0.6$, where linear approximator can be exploited for approximating accurately the slope of the target function.

In conclusion, XCSFHP performs almost as XCSF with the most suitable predictor both in terms of prediction accuracy and generalization capability.

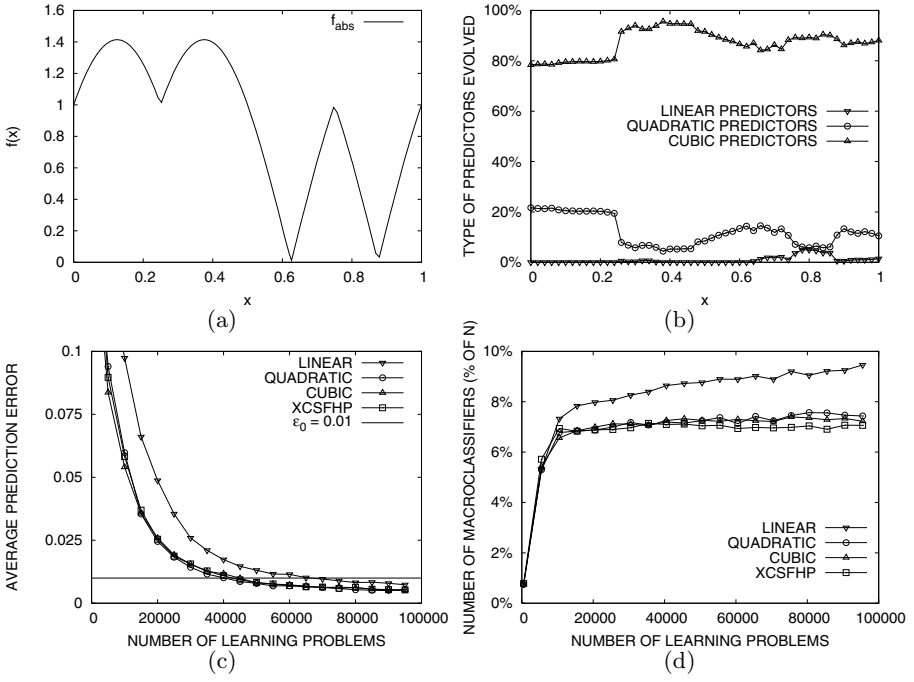


Fig. 2. XCSFHP and XCSF with linear, quadratic and cubic predictors applied to $f_{abs}(x)$ function: (a) the target function, (b) frequencies of predictor types evolved, (c) performance, and (d) population size. Curves are averages over 50 runs.

Altough so far we applied XCSFHP to very simple experiments, our results suggested that XCSFHP is capable of evolving the most appropriate predictor type for each subspace of the input domain.

4.2 Experiments with Constant, Linear and Neural Predictors

We now move to the analysis of XCSFHP with constant, linear and neural predictors ensemble. In the first experiments we compared XCSFHP to three XCSF versions respectively with constant, linear and neural predictors. Notice that XCSF with constant predictors turns out to be equal to XCSI [21] and XCSF with neural predictor has been introduced in [10] as XCSFNN.

In the first experiment we compared XCSFHP to the XCSF systems on the approximation of the f_3 function [6] defined as,

$$f_3(x_1, x_2) = \sin(2\pi(x_1 + x_2)). \tag{7}$$

The parameters are set according to [6] as follows: $N = 2000$, $\beta = 0.5$, $\delta = 0.5$, $\gamma = 0.9$ $\theta_{GA} = 50$, $\chi = 1.0$, $\mu = 0.05$, $\epsilon_0 = 0.01$, $\nu = 5$, $\alpha = 0.1$, $\theta_{del} = 20$, GA subsumption is on with $\theta_{sub} = 20$; action-set subsumption is not used; $m_0 = 1.0$, $r_0 = 0.5$ [22]. Figure 3 compares the performances, in terms of predictive accu-

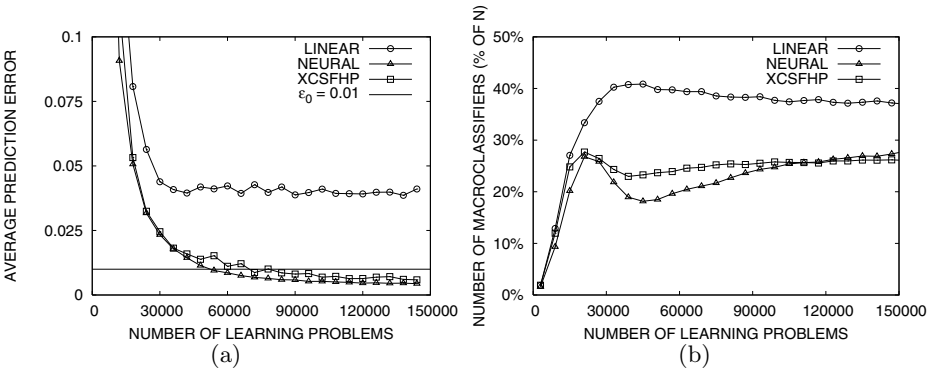


Fig. 3. XCSFHP compared to XCSF with constant, linear and neural network predictors on the approximation of $f_3(x_1, x_2)$: (a) performance and (b) population size. Curve are averaged over 20 runs.

racy (Figure 3a) and generalization capability (Figure 3b), of the four systems studied. Figure 3a shows that XCSF is able to evolve an accurate solution only with neural predictors (XCSF with constant predictors, not reported in the figure, reaches an average system error of approximately 0.2). It is worthwhile to say that XCSFHP is not only able to evolve an accurate solution but also converge toward it almost as fast as XCSF with neural predictors. Figure 3b compares the size of the populations evolved by the system compared. XCSF with neural predictors is able to evolve a more compact population than XCSF with linear prediction. Again, XCSFHP evolves a population almost as compact as XCSF with neural predictors. To understand how XCSFHP is able to evolve an accurate and compact solution, we analyzed the type of predictors evolved over the input space. Figure 4 shows that XCSFHP mainly evolved neural predictors being the most suitable type of predictor for approximating accurately the target function. Is interesting to note that (see Figure 4c) XCSFHP also evolves linear predictors in two regions of the input space: when both x_1 and x_2 approach to 0 and, especially, when they both approach to 1. In such regions, in fact, the target function can be effectively approximated with a linear function.

In the second experiment we applied XCSFHP and the XCSF systems to the the four variable function $f_4(x_1, x_2, x_3, x_4)$ [6], defined as

$$f_4(x_1, x_2, x_3, x_4) = x_2 + x_4 + 2\pi \sin x_1 + 2\pi \sin x_3, \tag{8}$$

with the same parameters setting of the previous experiment.

Figure 5a compares the performances of XCSFHP, XCSF with linear and with neural predictors. XCSF with constant predictors (not reported in the figure) converges to an average prediction error around to 0.3. On the function f_4 XCSF with linear predictors is not only able to evolve an accurate solution but converges also faster than XCSF with neural predictors. As discussed in [17], this result is due to the fact that in the f_4 function the variables contributions are linearly separable. It is interesting to note that, also in this case, XCSFHP

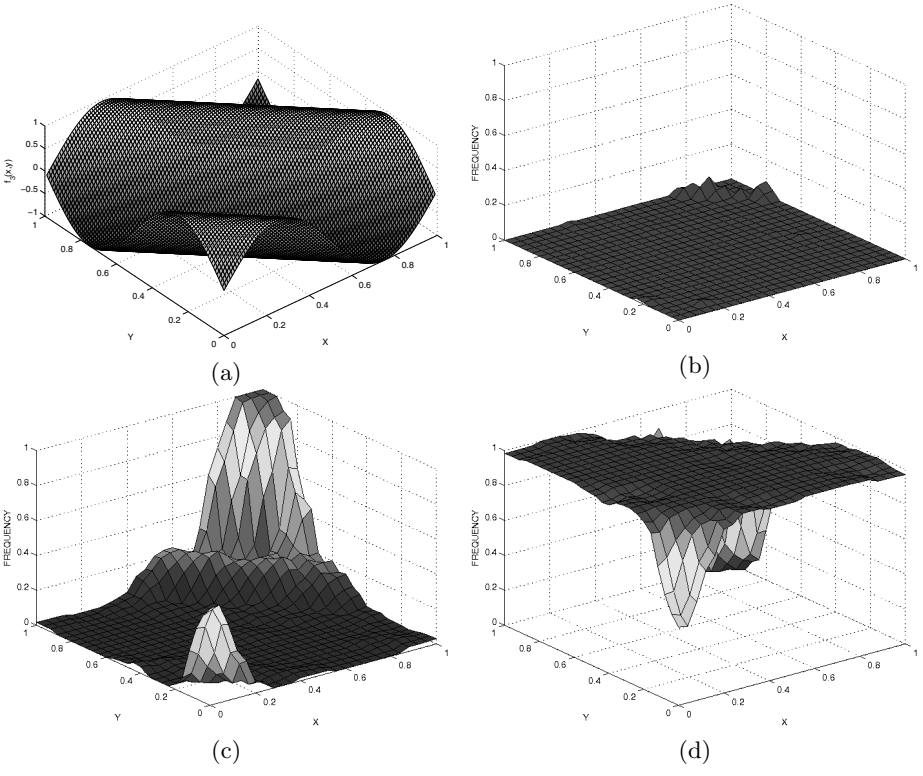


Fig. 4. Frequencies of the predictors types evolved by XCSFHP applied to $f_3(x_1, x_2)$: (a) the target function, (b) constant predictors, (c) linear predictors, (d) neural predictors

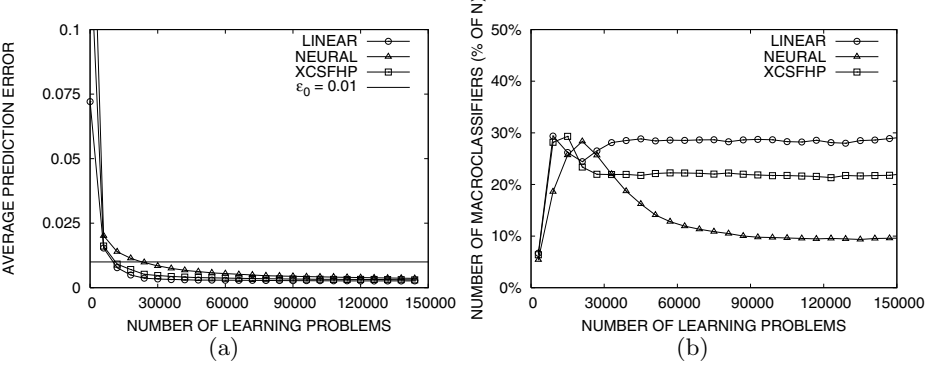


Fig. 5. XCSFHP compared to XCSF with constant, linear and neural network predictors on the approximation of $f_4(x_1, x_2, x_3, x_4)$: (a) performance and (b) population size. Curve are averaged over 20 runs.

is able to exploit the most appropriate predictors type and to converge almost as fast as XCSF with the best predictor for the function f_4 that is the linear one. Concerning the generalization capabilities, Figure 5b shows the size of the populations evolved by the systems. Notice that, although neural predictors are slower, they let XCSF evolve a more compact solution than linear ones. On the other hand XCSFHP evolves a solution not as compact as XCSF with neural predictors, but it is also able to exploit some neural predictors resulting in population smaller than the one evolved with linear predictors. In this case it is not possible to show the type of predictor evolved in each region of the input space due to its high dimensionality. However, XCSFHP evolved overall the 77.1% of linear predictors, the 22.8% of neural predictors, and the 0.1% of constant predictors. This confirms our previous discussions.

In summary we showed that XCSFHP performances are almost always comparable with the one of XCSF with the best predictor. This is due to the capability of XCSFHP to evolve autonomously the most appropriate predictor type in each subspace of the input domain.

5 XCSFHP on Multistep Problems

So far, we studied the XCSFHP when applied to function approximation problems. Now we apply XCSFHP to *multistep problems*, following the standard experimental design used in the literature [20,13,14]. In this set of experiments we used only ensembles involving polynomial predictors. In fact, it is well known [2], that complex predictors, like neural predictors, might not converge when applied to multistep problems.

5.1 2D Continuous Gridworld

In first experiments, we applied XCSFHP to a class of real valued multistep environments, the *2D Continuous Gridworld* [2]. The *empty continuous gridworld*,

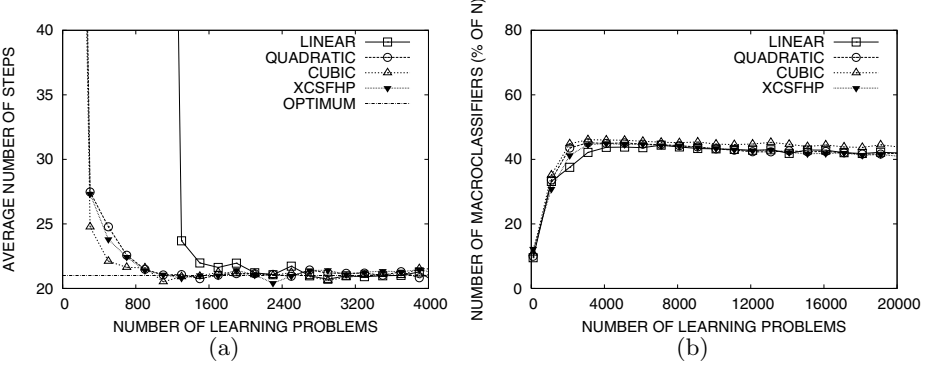


Fig. 6. XCSFHP and XCSF with linear, quadratic and cubic predictors applied to `Grid(0.05)`: (a) performance and (b) population size. Curves are averaged over 10 runs.

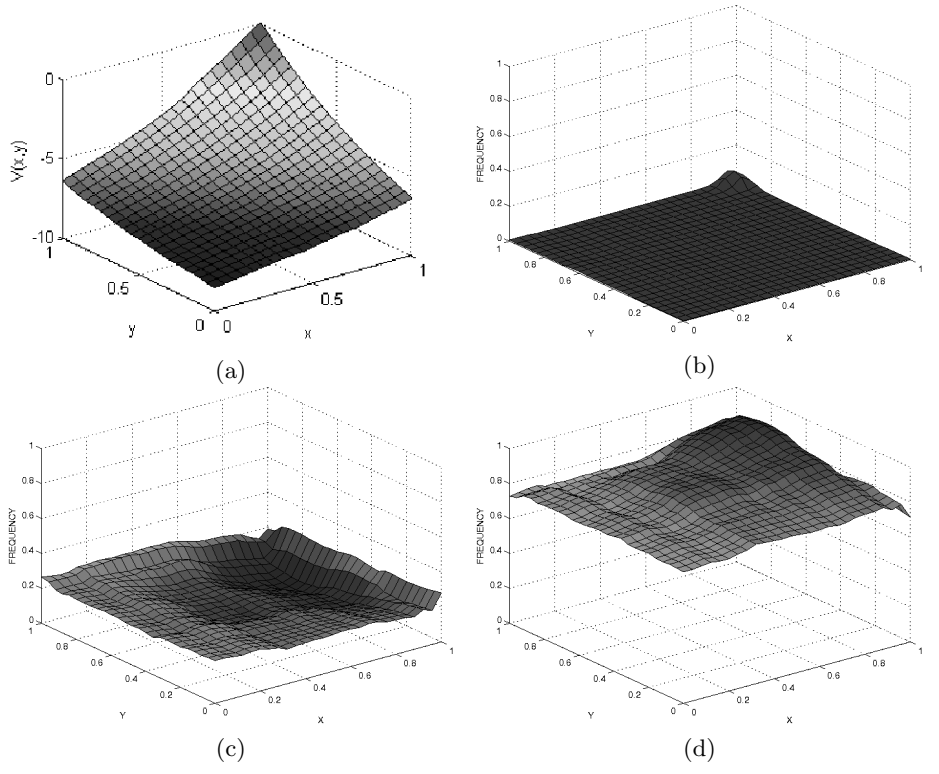


Fig. 7. Frequencies of predictor types evolved by XCSFHP: (a) the optimal value function of `Grid(0.05)`, (b) linear predictors, (c) quadratic predictors, and (d) cubic predictors. Statistics are averaged over 10 runs.

`Grid(`s`)` in brief, is a two dimensional environments in which the current state is defined by a pair of real valued coordinates $\langle x, y\rangle$ in $[0, 1]^2$, the only goal is in position $\langle 1, 1\rangle$, and there are four possible actions (left, right, up, and down) coded with two bits. Each action corresponds in a step of size s in the corresponding direction; actions that would take the system outside the domain $[0, 1]^2$ take the system to the nearest position of the grid border. The system can start *anywhere* but in the goal position and it reaches the goal position when *both* coordinates are equal or greater than one. When the system reaches the goal it receives a reward of 0, in all the other cases it receives -0.5. The performance is computed as the average number of steps needed to reach goal or food positions during the last 100 test problems. Given the step-size s, the optimal number of steps to reach the goal in `Grid(`s`)` is in the average equal to $(s+1)/s$ [20,13]. In this paper we set $s = 0.05$ a rather typical value used in the literature [20,13].

In the first experiments we applied XCSFHP with polynomial predictors and XCSF with linear, quadratic and cubic predictors to `Grid(0.05)` with the following parameter setting. $N = 5000$, $\beta = 0.2$, $\delta = 0.2$, $\gamma = 0.95$ $\theta_{GA} = 50$, $\chi =$

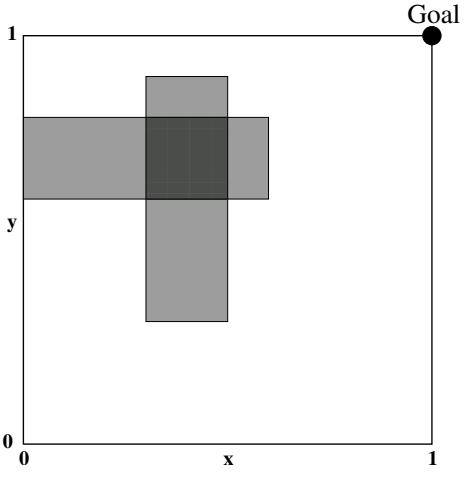


Fig. 8. The `Puddles(`s`)` problem. The goal is placed in $\langle 1, 1 \rangle$. Gray zones represent the "puddles".

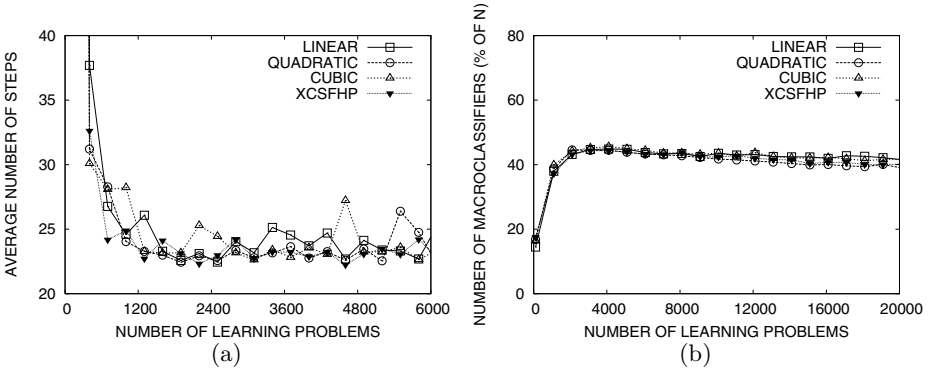


Fig. 9. XCSFHP and XCSF with linear, quadratic and cubic predictors applied to `Puddles(`0.05`)`: (a) performance and (b) population size. Curves are averaged over 10 runs.

0.8, $\mu = 0.04$, $\epsilon_0 = 0.005$, $\nu = 5$, $\alpha = 0.1$, $\theta_{del} = 50$, GA subsumption is on with $\theta_{sub} = 50$; action-set subsumption is not used; $m_0 = 0.5$, $r_0 = 0.5$ [22]. Figure 6a compares the performances of XCSFHP and of XCSF with different predictors. All the systems compared converge toward an optimal solution, but XCSF with linear predictor learns slower than XCSF with quadratic and cubic predictors. On the other hand XCSFHP converges as fast as XCSF with quadratic predictors and it is only slightly slower than XCSF with cubic predictors. When considering the size of the evolved population, reported in Figure 6b, all the systems seems to perform the same. As we did in previous problems, we analyzed the type of predictors evolved by XCSFHP over the problem space, reported in Figure 7. As expected from the previous results, XCSFHP evolves mainly cubic

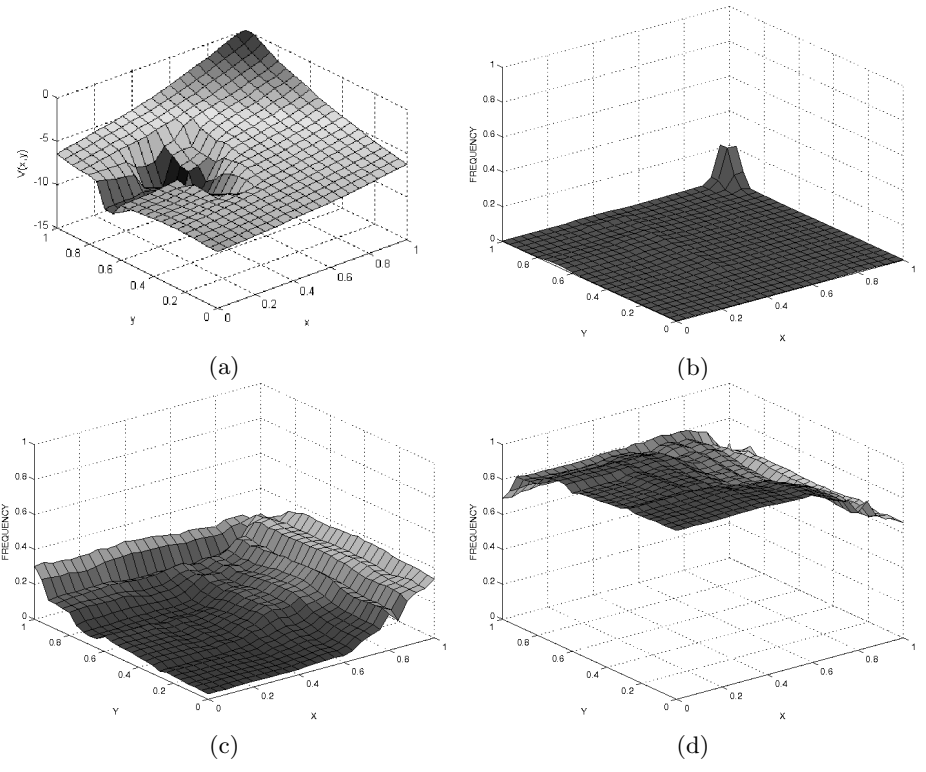


Fig. 10. Frequencies of predictor types evolved by XCSFHP: (a) the optimal value function of `Puddles(0.05)`, (b) linear predictors, (c) quadratic predictors, and (d) cubic predictors. Statistics are averaged over 10 runs.

predictors, that are more than the 80% of the all predictors evolved. Almost all the rest are quadratic predictors and only few linear predictors are evolved near the goal position. The results confirm what found in previous experiments on function approximation problems: XCSFHP evolves with higher probability the more appropriate predictors for solving the problem, i.e. the ones that allows XCSFHP to converge faster toward the optimal solution.

5.2 2D Continuous Gridworld with Puddles

In the second set of experiments we move to a slightly more complex problem, that is the 2D Continuous Gridworld with *Puddles*, `Puddles(`s`)` in brief. This problem adds some obstacles to 2D Continuous Gridworld, defined as areas involving an additional cost for moving. These areas are called "puddles" [2], since they actually create a sort of puddle in the optimal value function. We used the same experimental design followed in [13]. Figure 8 shows the `Puddles(`s`)` environment. In this paper we set $s = 0.05$, a rather typical value in the literature. Moving through a puddle results in an additional negative reward of -2; in the

area where the two puddles overlap, the darker gray region, the negative rewards sum up, i.e., the additional negative reward is -4.

Figure 9 shows that all the systems compared have almost the same performance and evolve almost the same number of macroclassifiers. The optimal value function of the `Puddles(0.05)`, reported in Figure 10a, does not allow the same generalization of the previous `Grid(0.05)` problem. Therefore the performance of XCSF with different predictors are very close. It is worthwhile to analyze what type of predictor does XCSFHP evolve over the problem space. Figure 10 shows that XCSFHP evolves prevalently cubic predictors, especially in the more critical region, i.e., the ones with the puddles; the number of evolved quadratic predictors is higher away from the puddles; a small number of linear predictor is also evolved only near the goal position, where the optimal value function has an almost linear slope.

6 Conclusions

In this paper, we introduced XCSF with heterogeneous predictors, namely XCSFHP, that extends XCSF by evolving classifiers with different type of predictors. Our approach differs from previous works on classifiers ensembles basically in two respects: (i) previous works usually evolve classifiers of the same type or focus on classifier representation, while we evolve classifiers with different type of predictors; (ii) previous works usually involve many populations or many systems, while we deal with evolving different type of classifiers in the *same* population. In order to investigate the capabilities of XCSFHP, we applied it to several problems. In principle, within XCSFHP it is possible to evolve classifiers with any type of predictors. However in the experimental analysis presented in this paper we focused on (i) ensembles involving polynomial predictors, i.e., linear, quadratic and cubic and on (ii) ensembles involving constant, linear and neural predictors. We compared XCSFHP to XCSF with a given type of predictor on several function approximation and multistep problems. Our results suggest that XCSFHP performs almost as XCSF with the best predictor type. In addition, the analysis of the type of predictors evolved by XCSFHP shows that it evolves the most appropriate type of predictor for each problem subspace, without any human intervention. Finally, relying on more than a single predictor type for solving a problem, we can also expect XCSFHP to be more robust than XCSF, even if further investigations in this direction are necessary.

References

1. Bagnall, A.J., Smith, G.D.: Using an Adaptive Agent to Bid in a Simplified Model of the UK Market in Electricity. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999), p. 774 (1999); One page poster paper
2. Boyan, J.A., Moore, A.W.: Generalization in reinforcement learning: Safely approximating the value function. In: Tesauro, G., Touretzky, D.S., Leen, T.K. (eds.) Advances in Neural Information Processing Systems 7, pp. 369–376. MIT Press, Cambridge (1995)

3. Bull, L.: On using ZCS in a Simulated Continuous Double-Auction Market. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999), pp. 83–90 (1999)
4. Bull, L., Studley, M., Bagnall, A., Whittley, I.: Learning classifier system ensembles with rule-sharing. IEEE Transactions on Evolutionary Computation 11(4), 496–502 (2007)
5. Bull, L., Studley, M., Bagnall, T., Whittley, I.: On the use of rule-sharing in learning classifier system ensembles. In: The 2005 IEEE Congress on Evolutionary Computation, 2005, vol. 1, pp. 612–617 (2005)
6. Butz, M.V.: Kernel-based, ellipsoidal conditions in the real-valued XCS classifier system. In: Beyer, H.-G., O'Reilly, U.-M., Arnold, D.V., Banzhaf, W., Blum, C., Bonabeau, E.W., Cantu-Paz, E., Dasgupta, D., Deb, K., Foster, J.A., de Jong, E.D., Lipson, H., Llora, X., Mancoridis, S., Pelikan, M., Raidl, G.R., Soule, T., Tyrrell, A.M., Watson, J.-P., Zitzler, E. (eds.) GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation, 25-29 June 2005, vol. 2, pp. 1835–1842. ACM Press, Washington (2005)
7. Butz, M.V., Wilson, S.W.: An Algorithmic Description of XCS. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 2000. LNCS (LNAI), vol. 1996, pp. 253–272. Springer, Heidelberg (2001)
8. Dorigo, M., Colombetti, M.: Robot Shaping: An Experiment in Behavior Engineering. MIT Press/Bradford Books (1998)
9. Lanzi, P.L.: The XCS library (2002), `http://xcslib.sourceforge.net`
10. Lanzi, P.L., Loiacono, D.: XCSF with neural prediction. In: Evolutionary Computation, 2006. CEC 2006. IEEE Congress on, pp. 2270–2276 (2006)
11. Lanzi, P.L., Loiacono, D., Wilson, S.W., Goldberg, D.E.: Extending XCSF beyond linear approximation. In: Genetic and Evolutionary Computation – GECCO-2005, pp. 1859–1866. ACM Press, Washington (2005)
12. Lanzi, P.L., Loiacono, D., Wilson, S.W., Goldberg, D.E.: XCS with computed prediction for the learning of boolean functions. In: Proceedings of the IEEE Congress on Evolutionary Computation – CEC-2005, Edinburgh, UK, September 2005, pp. 588–595. IEEE, Los Alamitos (2005)
13. Lanzi, P.L., Loiacono, D., Wilson, S.W., Goldberg, D.E.: XCS with computed prediction in continuous multistep environments. In: Proceedings of the IEEE Congress on Evolutionary Computation – CEC-2005, Edinburgh, UK, September 2005, pp. 2032–2039. IEEE, Los Alamitos (2005)
14. Lanzi, P.L., Loiacono, D., Wilson, S.W., Goldberg, D.E.: XCS with computed prediction in multistep environments. In: GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation, pp. 1859–1866. ACM Press, New York (2005)
15. Lanzi, P.L., Loiacono, D., Wilson, S.W., Goldberg, D.E.: Generalization in the XCSF classifier system: Analysis, improvement, and extension. Evolutionary Computation 15(2), 133–168 (2007)
16. Llora, X., Wilson, S.W.: Mixed decision trees: Minimizing knowledge representation bias in LCS. In: Deb, K., Poli, R., Banzhaf, W., Beyer, H.-G., Burke, E.K., Darwen, P.J., Dasgupta, D., Floreano, D., Foster, J.A., Harman, M., Holland, O., Lanzi, P.L., Spector, L., Tettamanzi, A., Thierens, D., Tyrrell, A.M. (eds.) GECCO 2004. LNCS, vol. 3103, pp. 797–809. Springer, Heidelberg (2004)
17. Loiacono, D., Lanzi, P.L.: Evolving neural networks for classifier prediction with XCSF. In: Proceedings of the Second Italian Workshop on Evolutionary Computation, ISSN 1970–5077 (2006)

18. Loiacono, D., Marelli, A., Lanzi, P.L.: Support vector regression for classifier prediction. In: GECCO 2007: Proceedings of the 9th annual conference on Genetic and evolutionary computation, pp. 1806–1813. ACM Press, New York (2007)
19. Widrow, B., Hoff, M.E.: Neurocomputing: Foundation of Research. In: Adaptive Switching Circuits, pp. 126–134. MIT Press, Cambridge (1988)
20. Wilson, S.W.: Classifier Fitness Based on Accuracy. Evolutionary Computation 3(2), 149–175 (1995)
21. Wilson, S.W.: Mining Oblique Data with XCS. In: Proceedings of the International Workshop on Learning Classifier Systems (IWLCS-2000), in the Joint Workshops of SAB 2000 and PPSN 2000, pp. 158–174 (2000)
22. Wilson, S.W.: Classifiers that approximate functions. Journal of Natural Computing 1(2-3), 211–234 (2002)
23. Wilson, S.W.: Classifier systems for continuous payoff environments. In: Deb, K., et al. (eds.) GECCO 2004. LNCS, vol. 3103, pp. 824–835. Springer, Heidelberg (2004)

Substructural Surrogates for Learning Decomposable Classification Problems

Albert Orriols-Puig[1,2], Kumara Sastry[2],
David E. Goldberg[2], and Ester Bernadó-Mansilla[1]

[1]Grup de Recerca en Sistemes Intel·ligents, Enginyeria i Arquitectura La Salle, Universitat Ramon Llull, Quatre Camins 2, 08022 Barcelona (Spain)
[2]Illinois Genetic Algorithms Laboratory, Department of Industrial and Enterprise Systems Engineering, University of Illinois at Urbana-Champaign
aorriols@salle.url.edu, ksastry@uiuc.edu, deg@uiuc.edu, esterb@salle.url.edu

Abstract. This paper presents a learning methodology based on a substructural classification model to solve decomposable classification problems. The proposed method consists of three important components: (1) a structural model, which represents salient interactions between attributes for a given data, (2) a surrogate model, which provides a functional approximation of the output as a function of attributes, and (3) a classification model, which predicts the class for new inputs. The structural model is used to infer the functional form of the surrogate. Its coefficients are estimated using linear regression methods. The classification model uses a maximally-accurate, least-complex surrogate to predict the output for given inputs. The structural model that yields an optimal classification model is searched using an iterative greedy search heuristic. Results show that the proposed method successfully detects the interacting variables in hierarchical problems, groups them in linkages groups, and builds maximally accurate classification models. The initial results on non-trivial hierarchical test problems indicate that the proposed method holds promise and also shed light on several improvements to enhance the capabilities of the proposed method.

1 Introduction

Nearly decomposable functions play a central role in the design, analysis, and modeling of complex engineering systems [28,6,8]. A design decomposition principle has been proposed for the successful design of scalable genetic algorithms (GAs) [8,18,20], genetic programming [25], and learning classifier systems and genetics based machine learning (GBML) [4,16]. For example, in [4], estimation of distribution algorithms (EDAs) were applied over the rule-based knowledge evolved by XCS [32,33] to discover linkages between the input variables, permitting XCS to solve hierarchical problems that were intractable with first-generation XCS.

Nonetheless, previous approaches used the probabilistic models built by EDAs—GAs that replace variation operators by building and sampling probabilistic models of promising solution—for recombination. However, the

J. Bacardit et al. (Eds.): IWLCS 2006/2007, LNAI 4998, pp. 235–254, 2008.

probabilistic models can also be used to induce the form of surrogates which can be used for efficiency enhancement of GAs [27,19,26] and GBML [17]. In this paper, we use the substructural surrogates for learning from decomposable problems with nominal attributes. Similar to Sastry, Lima, and Goldberg [26], we use the structural model of EDAs to induce the form of the surrogate and linear regression for estimating the coefficients of the surrogate. The surrogate is subsequently used to predict the class of unknown input instances.

In this paper, we discuss the critical components of the proposed methodology and outline several ways to implement it. We then propose a greedy search heuristic for discovering the structural model that minimizes the test error of the classification model constructed from it. We address this method as *greedy Extraction of the Structural Model for Classification* (gESMC). We artificially design a set of hierarchical problems by means of concatenating essential blocks whose output, provided by a boolean function, serves as the input of another function that determines the global output of the example. Thus, these problems may be decomposed and essential blocks should be correctly processed to predict the correct output. gESMC is able to detect the interactions between variables and build accurate classification models. Moreover, the system is compared to C4.5 and SMO. The comparison highlights that extracting the problem structure is essential to solve hierarchical problems. Finally, we review the limitations of applying a greedy search to obtain the best structural model, show in which circumstances these limitations may appear, and propose approaches to overcome them.

The paper is organized as follows. Section 2 discusses the proposed methodology followed by a description of gESMC. The test problems designed and used in this study are discussed in Sect. 4. Section 5 compares the results of gESMC with C4.5 and SMO on the hierarchical problems. Section 6 discusses some enhancements that are yet to be investigated. Section 7 provides summary and conclusions.

2 Methodology for Learning χ-Ary Input Problems

In this section, we discuss a methodology for learning the structural and the classification model from a set of labeled examples. The methodology consists of three layers: (1) the *structural model layer*, (2) the *surrogate model layer*, and (3) the *classification model layer*. The *structural model layer* extracts the dependencies between the attributes of the examples in the dataset. These dependencies can be expressed in form of linkage groups [9,10], matrices [35], or Bayesian networks [18]. However the dependencies are represented, the key idea is that the salient interactions between attributes are used as a basis for determining the output. The *surrogate model layer* uses the *structural model* to infer the functional form of the surrogate, and the coefficients of the surrogate are determined using linear regression methods. The resulting surrogate is a function that approximates the output of each input instance. Finally, the *classification model layer* uses the surrogate function to predict the class of new input instances.

In essence, we infer the structure of the surrogate from the structural models of attribute interactions and then use linear regression methods to estimate

the value of the coefficients (or the partial contributions of subsolutions to the output) of the resulting surrogate function. Finally the surrogate is used to predict the class of new input instances. Details of each of the three components are discussed in the following sections.

2.1 Structural Model Layer

The *structural model layer* is responsible for identifying salient interactions between attributes, which need to be processed together to determine their contribution to the output. For example, consider a problem with two binary attributes (x_1, x_2) and whose output is determined by the *x-or* boolean function. If we considered each of the attributes independently, we cannot evolve a function that computes the output accurately for all possible inputs. However, when we consider the two attributes together, we can easily create a function that accurately predicts the output for all possible input sequences.

A number of linkage-learning methods [8] can be used to implement the structural model layer. Here, we use estimation of distribution algorithms (EDAs) [18,20], which learn the salient interactions between decision variables by building probabilistic models of promising candidate solutions. These probabilistic models can be expressed in different forms such as (i) linkage groups [9,10], i.e., groups of variables that have a salient interaction; matrices [35], which express the relationship between pairs or groups of variables, permitting to detect overlaps in the groups of interacting variables; or Bayesian networks [18], in which the nodes represent variables and the connections denote salient interactions between variables. The implementation proposed in the next section uses linkage groups to express the salient interactions, although it can be extended to other representations.

In the realm of learning classifier systems (LCSs) or genetics-based machine learning (GBML), EDAs have been successfully combined with LCSs to extract the linkages between classifiers' alleles [4,16,17]. However, unlike previous studies which used the structural model as a replacement of recombination, in this study we integrate the structural model and learning with the use of substructural surrogates.

In order to achieve this integration, the first step is to find the structural model of the given data. This can be done in several ways. As with EDAs, given a class of permissible structural models, we can search for the best structural model. Prior and domain-specific knowledge can also be used to propose the structural model, and a search mechanism could be used to refine it [1]. In this study we use a greedy search heuristic that searches for the model structure that results in the most accurate surrogate model, details of which are given in Section 3.

2.2 Surrogate Model Layer

The *surrogate model layer* preprocesses the input examples according to the structural model and builds a regression model from these preprocessed examples, as described in [26]. In this section we summarize the procedure of building

such a surrogate. Consider a matrix D of dimension $n \times \ell$ that contains all the input examples (where n is the number of examples and ℓ the number of attributes). Once the structural model is built, every linkage group is treated as a *building block* [11]. Then, we consider all possible input combinations within each linkage group to process the input examples.

For example, consider the following structural model of a binary problem of 3 variables: $\{[x_1, x_3], [x_2]\}$. That is, there is salient interaction between variables x_1 and x_3, which are independent from the variable x_2. In this case, we consider the following schemata: {0*0, 0*1, 1*0, 1*1, *0*, *1*}. In general, given m linkage groups, the total number of schemata m_{sch} to be considered is given by:

$$m_{sch} = \sum_{i=1}^{m} \left[\Pi_{j=1}^{k_i} \chi_{i,j} \right], \tag{1}$$

where $\chi_{i,j}$ is the alphabet cardinality of the j^{th} variable of the i^{th} linkage group, and k_i is the size of the i^{th} linkage group.

Then, each example in D is mapped to a vector of size m_{sch}, creating the matrix A of dimensions $n \times m_{sch}$:

$$\mathbf{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m_{sch}} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m_{sch}} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,m_{sch}} \end{pmatrix}, \tag{2}$$

where $a_{i,j}$ will have value '1' if the ith example belongs to the jth schemata, and '0' otherwise. Note that, given an example, only one of the schemata for each linkage group can have value '1'.

We map different labels or classes of the examples to numeric values. For example: $\{class_1, class_2, \cdots, class_k\} \longrightarrow \{\mathbb{Z}_1, \mathbb{Z}_2, \cdots, \mathbb{Z}_k\}$. The label or class c_i of each example is also kept in a matrix C of dimensions $n \times 1$:

$$\mathbf{C} = \begin{pmatrix} c_1 & c_2 & \cdots & c_n \end{pmatrix}^{\tau}. \tag{3}$$

Now, the task of designing the surrogate can be formulated into a linear system of equations and our objective is to compute the coefficients of the matrix $\mathbf{x}$ of dimensions $m_{sch} \times 1$ that satisfy the following equality:

$$\mathbf{Ax} = \mathbf{C}. \tag{4}$$

In practice, we may not find an $\mathbf{x}$ that satisfies this expression. For this reason, we use a multi-dimensional least squares fitting approach. That is, the problem is reformulated by estimating the vector of coefficients $\mathbf{x}$ that minimize the square error function χ:

$$\chi^2 = (Ax - C)^T \cdot (Ax - C). \tag{5}$$

The problem of least-squares fitting is well-known, and so we do not provide insight in the resolution methodology herein. The interested reader is referred to [5,23]. Here, we used the multi-dimensional least squares fitting routine available with Matlab [24].

2.3 Classification Model Layer

Once we obtain the matrix x with the regression coefficients, the output for a new example is computed as follows. The example is mapped to a vector $\vec{e}$ of size m_{sch}. The mapping procedure used is identical to that used to create matrix $\mathbf{A}$ and as outlined in the previous section, the elements of $\vec{e}$ will have a value '1' if the example belongs to the corresponding schemata and '0' otherwise. Then, the predicted output is given by:

$$output = \vec{e} \cdot x. \tag{6}$$

Note that the *output* is a continuous value, and has to be transformed to one of the possible class labels. Therefore, we convert the continuous output to the closer integer $\mathbb{Z}_i$ in $\{\mathbb{Z}_1, \mathbb{Z}_2, \cdots, \mathbb{Z}_k\}$, and then, return the class label that corresponds to $\mathbb{Z}_i$.

In essence, the proposed method relies on the structural and the surrogate models extracted from the data to build the classification model. Therefore, we note that if this structural model does not reflect the variable interactions accurately, the accuracy of the classification model will be limited. Thus, a critical task for the success of the proposed methodology is our ability to find reasonably accurate structural models. In the next section we propose an implementation of the methodology that searches iteratively for the best structural model, and uses the classification model to evaluate its quality. We call this implementation *greedy extraction of the structural model for classification* (gESMC).

3 Implementing the Methodology: gESMC

The pseudocode of the implementation of our proposed method is shown in Algorithm 1. In the initialization stage, the algorithm divides the data into training and test sets. We start with a structural model where all variables are treated as independent and build a surrogate function via regression over the training set as explained in the previous section (see Section 2.2). The quality of the classification model is evaluated with the test set and stored in the variable *mdl*.

Similar to the extended compact genetic algorithm (eCGA) [10], in gESMC we use a greedy search heuristic to partition the set of attributes into non-overlapping clusters such that the classification error is (locally) minimized. That is, starting from a model where variables are treated as independent, we continue to merge substructures till either (1) the *mdl* measure becomes less than a user set threshold θ, or (2) the search produces no improvement. In every iteration of the inner loop (lines 10 to 13), we merge two linkage groups from the current best model, create the surrogate and the classification model, and evaluate it. That is, $\binom{m}{2}$ new structural models are formed (where m is the number of substructures in the current best model), and their surrogate functions created and evaluated. Among the evaluated $\binom{m}{2}$ models, the one with the lowest classification error is chosen as the current best model for the next iteration if it *significantly improves*

Algorithm 1. Building of structural and classification model via a greedy search.

```
   Data: dataset is the set of labeled examples.
   Result: function is the classification function and bestModel the structural
           model.
 1 begin
 2   i ← 0
 3   count ← 0
 4   [train, test] ← divideData ( data )
 5   bestModel ←  [1], [2], ..., [n]
 6   function ←— createSurrogateFunction ( bestModel, train ) ▷ See Sect. 2.2
 7   mdl ← evaluateModel ( bestModel, test )
 8   isImproving ← true
 9   while mdl > θ and isImproving do
10     for i ∈ {1, ..., length(bestModel) − 1} do
11       for j ∈ {i + 1, ..., length(bestModel) − 1} do
12         newModel[count] ← joinLinkages( bestModel, i, j)
13         newFunction[count] ← createSurrogateFunction (
           newModel[count], train )
14         newMdl[count] ← evaluateModel ( newModel[count], test )
15         count ← count + 1
16       end
17     end
18     best ← position min. mdl( newMdl )          ▷ Selects the best model
19     if newMdl[best] significantly improves mdl then
20       bestModel = newModels[best]
21       mdl = newMdl[best]
22     else
23       isImproving=false
24     end
25   end
26 end
```

the current best model; otherwise, we terminate the search, and the current best surrogate and classification models are returned as the (locally) best models.

Three elements of the implementation need further explanation: (1) procedure to divide the data into training and test sets (line 2), (2) evaluation of the model (lines 5 and 12), and (3) procedure for comparing two models and choosing the best one (line 17). Each of the three elements are discussed in the following paragraphs.

Partition of the data. The procedure used to partition the data into training and test sets affects the estimation of classification error. A number of approaches such as holdout validation, k-fold cross validation, and leave-one-out cross-validation methods can be used. Here, we use a k-fold cross validation [29] with $k = 10$.

Evaluation of the model. The quality of a structural model depends on (1) the complexity of this model, and (2) the test error of the classification model created from it. Again a number of measures such as minimum description length metrics and multiobjective approaches could be used to measure the relative quality of a given surrogate and classification model. We use the k-fold cross validation which provides a measure of both the test error and the model complexity in terms of overfitting the training data. That is, if the structural model is more complex than necessary, the surrogate function will tend to overfit the training instances, and the test error will increase. Nonetheless, we acknowledge that direct measures for model complexity could be included in the evaluation.

Comparison of models. Given a current-best model, in gESMC we consider all pairwise merges of the substructures of the current-best model. We need to choose the best model among all the models created via the pairwise merges and compare it to the current-best model. Again, this could be done in a number of ways. For example, we could accept the new model if its classification error is lower than that of the current-best model. However, this might lead to spurious linkages and more complex models might be accepted, especially if the data set is noisy. To avoid getting unnecessarily complicated structural models, we can say that a model m_1 is significantly better than a model m_2 if:

$$error_{m_1} < error_{m_2} - \delta, \tag{7}$$

where δ is a user-set threshold. Alternatively, we can use different statistical tests as well. In our implementation, we use a paired t-test to determine if a new, more complex, structural model is better than the current best model [29]. That is, we feed the t-test with the errors corresponding to the ten different folds obtained with the current best model and the new model. We use a significance level of $\alpha = 0.01$.

Before proceeding with a description of the test functions, we note two important properties of gESMC. First, in the current implementation gESMC, the structural model is a partition of the variables into non-overlapping groups. However, this limitation can easily be relaxed by using other structural models [35,18]. Second, because of the greedy procedure, we need some guidance from lower-order, sub-optimal structural models toward an optimal structural model. This limitation can be alleviated by replacing the greedy search heuristic with another optimization method such as genetic algorithms [11,7]

4 Test Problems

In this section, we present a general set of decomposable problems to investigate the capabilities of gESMC in correctly identifying salient substructures and building an accurate classification model. Following the idea of designing hierarchical artificial problems proposed in [4,3], we design a class of two-level

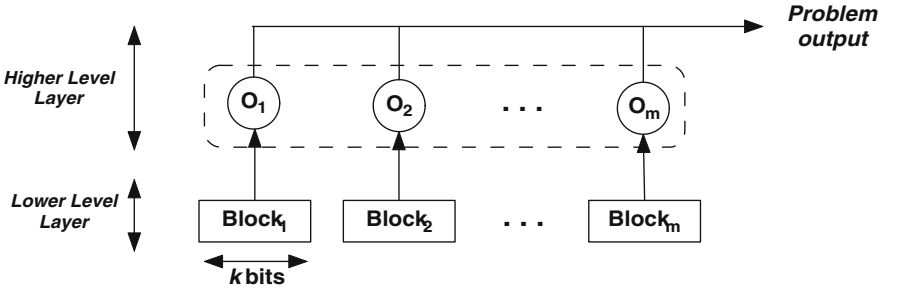


Fig. 1. Example of the design of a two-level hierarchical problem. In the low level layer, m blocks of k bits are concatenated. Each block is evaluated resulting in the correspondent output. All the outputs are groups in an input string that is used to determine the global output.

hierarchical problems where the lower level consists of functions that operate on a set of binary-input blocks and the upper level consists of functions that operate on the lower-level function values to produce the output (see a schematic illustration of these types of problems in Fig. 1). The lower- and upper-level function used in the study are explained in Section 4.1 and 4.2, respectively.

4.1 Lower Level of the Hierarchy

At the lower level of the hierarchy, we considered the following two binary functions which operate independently on m blocks with k variables in each block. Moreover the variables within a block interact with each other and determine the output of the function.

The position problem. The *position* problem [2] is defined as follows. Given a binary input of length ℓ, the output is the position of the left-most one-valued bit. For example, f(1̲00)=3, f(01̲0)=2, f(001̲)=1, and f(000)=0. Note that every variable is linked to all the variables on its left.

The parity problem. The *parity* problem [15] is a two-class binary problem defined as follows. Given a binary input of length ℓ, the output is the number of one-valued bits modulo two. For example, f(110)=0, f(100)=1, and f(111)=1. To predict the output accurately for the parity problem, all the variables have to be jointly processed. Additionally, for a k-bit parity problem, the structural model that represents that all the variables are independent yields a classification model of the same accuracy as the one that contains substructures of size $k-1$ or less. That is, till we get a structural model that groups all k variables together, the accuracy of the classification model does not increase.

4.2 Higher Level of the Hierarchy

At the higher level of the hierarchy, we use the following problems where each variable contributes independently to the output. That is, the structural

information is contained in the lower-level of the hierarchy and the upper level function affects the salience of the substructures. Notice that χ-ary strings are permitted in the higher level. That is, both problems defined as follows for the higher level can deal with χ-ary strings.

The decoder problem. The decoder problem [2] is a binary-input multi-class problem defined as follows. Given an input of length ℓ, the output is determined by the decimal value of the input. For example, $f(111) = 7$, $f(101) = 5$, and $f(000) = 0$. Note that each variable independently contributes to the output. That is, starting with class equal to zero, a '1' in i^{th} position adds 2^i to the output, irrespective of other variable values.

The count-ones problems. The *count-ones* is defined as follows. Given a binary input of size ℓ, the output is the number of one-valued bits. Again, the output of the count-ones problems can be predicted by treating the input variables independently.

As mentioned earlier, we concatenated m blocks of k bits of the two lower level problems with the two higher level problems to create four different hierarchical test problems. Specifically, we used the *position* at the lower level with the *decoder* (HPosDec) and the *count-ones* (HPosCount) in the higher level. Similarly, low order *parity* blocks were combined again with the *decoder* (HParDec) and the *count-ones* (HParCount). Additionally, we added some irrelevant bits, which do not contribute to the output, to see if our method was capable of ignoring them. Therefore, in our case, $\ell \geq m \cdot k$, where ℓ is the length of the input string.

With the above description of the test problems, the following section presents the results of gESMC and compares them with those of C4.5 and SMO.

5 Results

This section analyzes the behavior of gESMC for learning hierarchical problems, and compares the results to those obtained with two highly competitive learners in terms of performance and interpretability.

5.1 Experimental Methodology

We use the four hierarchical problems designed in the previous section to analyze the performance of gESMC. We start with concatenations of three minimum-order blocks in the lower level hierarchy (that is, k=2) and add 9 irrelevant bits to the input. Our aim is to analyze the capabilities of gESMC in (i) identifying salient substructures of interacting variables, and (ii) ignoring irrelevant variables. Next, we increase the order of the lower-order blocks with a two-fold objective. For the problem with position blocks, we analyze if the system is able to identify and efficiently handle larger groups of linked variables. For the problems with parity blocks, we want to investigate the behavior of gESMC when there is a lack of guidance toward an accurate substructural model.

Table 1. Test error and standard deviation obtained with gESMC, SMO and C4.5 on the problems HPosDec, HPosCount, HParDec, HParCount with ℓ=15, $m = 3$, and $k = 2$. Results are averaged over ten runs with different holdouts and random seeds.

	gESMC	**C4.5**	**SMO**
HPosDec	0.00% ± 0.00%	0.00% ± 0.00%	0.00% ± 0.00%
HPosCount	0.00% ± 0.00%	0.00% ± 0.00%	21.89% ± 0.13%
HParDec	0.00% ± 0.00%	3.32% ± 2.90%	89.11% ± 0.94%
HParCount	0.00% ± 0.00%	5.15% ± 4.43%	62.72% ± 0.27%

To illustrate the need for detecting linkage groups in classification tasks, we compare the results obtained with gESMC to those of two widely used learners: C4.5 [22], and SMO [21]. C4.5 is a decision tree, derived from ID3, which has been widely used because of its ability to tackle a wider range of problems and because of the interpretability of the extracted knowledge. SMO is a *support vector machine* [30] that implements the *Sequential Minimal Optimization* algorithm. Although the interpretability is more difficult since it represents the knowledge as function weights, its competence has been demonstrated in different kinds of problems. Both methods were run using WEKA [34]. Unless otherwise noted, for C4.5 we used the default configuration, and for SMO, we used a polynomial kernel of order 1.

The three methods are compared in terms of performance (that is, test accuracy) and comprehensibility of the knowledge generated by the learner. As the datasets had a large number of instances, we used the *holdout* methodology[1] to estimate the test accuracy; that is, 70% of the instances were randomly selected and placed in the training set, and the rest formed the test set. We repeated the experiments with ten different holdouts. To compare the performance of each pair of learners on a given problem, we applied a paired Student t-test [29] on the results. We fed the results for each different seed to the t-test. To study the interpretability of each method, we qualitatively compared the structural and the classification models evolved by gESMC to the decision trees generated by C4.5, and the weights extracted by SMO.

5.2 Results with 2-Bit Low Order Blocks

We first show performances for gESMC, C4.5, and SMO on the problems HPosDec, HPosCount, HParDec, and HParCount with $\ell = 15$, $m = 3$, and $k = 2$. Therefore, the problems were formed by three lower level blocks of two bits and 9 irrelevant bits at the end of the binary input. Next, we compare the results in terms of performance and interpretability.

Comparison of the Performance. Table 1 summarizes the test errors resulting of applying gESMC, C4.5, and SMO on the four hierarchical problems. All

[1] A holdout is the simplest cross-validation approach where the data is divided in two sets, the train and the test set.

Table 2. Structural models and surrogate functions build by gESMC for the problems HPosDec, HPosCount, HParDec, HParCount with ℓ=15, $m = 3$, and $k = 2$

HPosDec	*link. groups*	$[x_0x_1][x_2x_3][x_4][x_5][x_6][x_7][x_8][x_9][x_{10}][x_{11}][x_{12}][x_{13}][x_{14}]$
	surr. func.	$16.75 + 9(1 - \overline{x}_0x_1) - 6\overline{x}_2\overline{x}_3 - 3\overline{x}_2x_3 - 1.5x_4 + 0.5x_5$
HPosCount	*link. groups*	$[x_0x_1][x_2x_3][x_4][x_5][x_6][x_7][x_8][x_9][x_{10}][x_{11}][x_{12}][x_{13}][x_{14}]$
	surr. func.	$0.75 + \overline{x}_0x_1 + (1 - \overline{x}_2x_3) + 0.5\overline{x}_4 + 0.5x_5$
HParDec	*link. groups*	$[x_0x_1][x_2x_3][x_4x_5][x_6][x_7][x_8][x_9][x_{10}][x_{11}][x_{12}][x_{13}][x_{14}]$
	surr. func.	$2 + 4(\overline{x}_0x_1 + x_0\overline{x}_1) - 2(\overline{x}_2\overline{x}_3 + x_2x_3) - (\overline{x}_4x_5 + x_4\overline{x}_5)$
HParCount	*link. groups*	$[x_0x_1][x_2x_3][x_4x_5][x_6][x_7][x_8][x_9][x_{10}][x_{11}][x_{12}][x_{13}][x_{14}]$
	surr. function	$\overline{x}_0x_1 + x_0\overline{x}_1 + \overline{x}_2x_3 + x_2\overline{x}_3 + \overline{x}_4x_5 + x_4\overline{x}_5$

the results were averaged over ten runs, each with a different holdout partition and random seed.

The results show that gESMC obtained 0% test error for all the problems tested. This indicates that the method is able to process the variable linkages and build maximally accurate classification models. None of the other learners could achieve 0% error in all the problems. C4.5 achieved 0% test error for the problems HPosCount and HPosDec, the ones formed by position blocks. Nonetheless, on the problems that consist of parity blocks, C4.5 was significantly outperformed by gESMC according to a paired t-test on a confidence level of 0.99. Finally, SMO presents the worst behavior of the comparison. The learner could accurately generalize over the input data only on the HPosDec problem. For the problems HPosCount, HParDec, and HParCount, the results of SMO significantly degraded those obtained with gESMC and C4.5. Note the big difference in the test errors; for HParDec, SMO has 89.11% test error, C4.5 has 3.32%, and gESMC is maximally accurate. We repeated the experiments with a Gaussian kernel [13] to promote the discovery of the linkage groups, but no significant improvement was found.

These results highlight the importance of learning and incorporating the structural model into the classification model. gESMC found highly accurate classification models only after discovering the problem structure (examples of some structural and classification models are shown in the next section). However C4.5 and SMO failed since they were not able to identify this structure. Note that the problems formed by parity blocks resulted more problematic for both learners than the problems based on position blocks. This could be explained as follows. The variables linkages in the position are weaker than in the parity. That is, in the position problem every variable processed from left to right reduces the uncertainty of the output. In the parity, looking at a single variable does not reduce the uncertainty, and so, processing the linkages is crucial. We hypothesize that, for this reason, problems formed by parity are more difficult to learn for C4.5 and SMO.

Comparison of the Interpretability. We now analyze the interpretability of the models created by gESMC, and qualitatively compare them to those obtained by C4.5 and SMO. Table 2 shows the structural models and the associated

surrogate functions built for each problem. For HPosDec and HPosCount, gESMC correctly detects the linkages between the groups of variables $[x_0, x_1]$ and $[x_2, x_3]$; all the other variables are considered independent. Variables x_4 and x_5 are incorrectly identified as independent because gESMC reaches the termination criteria of 0% test error. For the problems HParDec and HParCount, gESMC discovers the linkage groups $[x_0, x_1]$, $[x_2, x_3]$, and $[x_4, x_5]$. Differently from the position problem, now gESMC needs to discover all the existing parity groups to remove the uncertainty, and so, build the most accurate classification model.

The availability of the structural model with gESMC is another advantage over other conventional classification techniques in terms of interpretability. The structural model facilitates easy visualization of the salient variable interactions; moreover, it permits a better understanding of the resulting surrogate function. Note that for all the problems, gESMC built easily interpretable functions and also efficiently ignored irrelevant variables. For example, consider the problem HParCount, in which the output is the number of '1s' resulting from the evaluation of each low-order parity block. The function evolved clearly indicates that if any of the linkage groups has the schemata '01' or '10' (values from which the parity would result in '1'), the output is incremented by one.

Let us now compare this knowledge representation to those obtained with C4.5 and SMO. For this purpose, we consider the size of the trees built by C4.5, and the machines constructed by SMO. For HPosDec and HPosCount,

```
at1 = 0
|   at2 = 0
|   |   at3 = 0
|   |   |   at4 = 0
|   |   |   |   at5 = 0
|   |   |   |   |   at6 = 0: 0 (153.0)
|   |   |   |   |   at6 = 1: 1 (150.0)
|   |   |   |   at5 = 1: 2 (314.0)
|   |   |   at4 = 1
|   |   |   |   at5 = 0
|   |   |   |   |   at6 = 0: 3 (154.0)
|   |   |   |   |   at6 = 1: 4 (139.0)
|   |   |   |   at5 = 1: 5 (311.0)
|   |   at3 = 1
|   |   |   at5 = 0
|   |   |   |   at6 = 0: 6 (277.0)
|   |   |   |   at6 = 1: 7 (299.0)
|   |   |   at5 = 1: 8 (632.0)
            . . .
```

Fig. 2. Portion of the tree built by C4.5 for the HPosDec problem

C4.5 built a tree with 53 nodes, from which 27 were leaves. The resulting trees specified the output for each combination of the six relevant bits (see a portion of a tree for HPosDec in Fig. 2). Although these trees detail the output given the value of the first six variables, they do not show the variable interactions. For HParCount, C4.5 built trees that, on average, had 136 leaves and 270 nodes. For HParDec, the trees had 142 leaves and 283 nodes. This high number of nodes makes the interpretability of tree very hard. Additionally, all the trees had some irrelevant attributes in the decision nodes. That is, C4.5 was overfitting the training instances to reduce the training error, resulting in more complicated trees, further hindering the interpretability of the classification model.

In contrast to gESMC and C4.5, SMO presented the less interpretable results. In general, SMO creates a machine for each pair of classes, and adjust $\ell + 1$ weights for each machine (where ℓ is the number of attributes of the problem). For HPosDec, SMO built 351 machines with 16 weights ranging from 0 to 1. For HPoscount, HParDec, and HParCount, 6, 6, and 28 machines were respectively created, all them with 16 weights ranging from 0 to 1. Although some of these weights were zero, the machines evolved could not be interpreted at all. Thus, the human expert would not be able to extract any information from these knowledge models.

Although both SMO and gESMC represent the knowledge in weights of functions that partition the search space, gESMC yields the structural model which permits easy visualization of salient variable interactions. Additionally, while SMO weights the input variables, gESMC weights the different subsolutions of the identified substructures. Classification models obtained via gESMC show the relative influence of subsolutions to the output and therefore gESMC's models are more easily interpretable than the ones created by SMO.

5.3 Results Increasing the Low Level Block Size

We now increase the interaction order of the lower-level blocks to analyze its effect on the performance of gESMC. Additionally, for the test problems with parity blocks, we also want to investigate the effect of having no guidance from lower-order substructures. Specifically, we want to analyze if this lack of guidance thwarts the search of gESMC toward the best structural form of the surrogate. For this purpose, we use HPosDec, HPosCount, HParDec, and HParCount with $\ell = 15$, $m = 2$, and $k = 3$, and compared gESMC to C4.5 and SMO.

Table 3 shows the test errors for gESMC, C4.5, and SMO. For HPosDec and HPosCount, gESMC obtained 0% error test and for both problems, two different structural models were created during independent runs:

$$Model_1 : [x_0x_1][x_3x_4x_5][x_2][x_6][x_7][x_8][x_9][x_{10}][x_{11}][x_{12}][x_{13}][x_{14}],$$
$$Model_2 : [x_0x_1x_2][x_3x_4][x_5][x_6][x_7][x_8][x_9][x_{10}][x_{11}][x_{12}][x_{13}][x_{14}].$$

In both the above models, the variables of one of the lower-level blocks are correctly identified, and only two variables of the other lower-level blocks form a linkage group. As observed in the previous section, this is because gESMC meets

Table 3. Test error and standard deviation obtained with gESMC, SMO and C4.5 on the problems HPosDec, HPosCount, HParDec, HParCount with ℓ=15, $m = 3$, and $k = 3$. Results are averages over ten runs with different holdouts and random seeds.

	gESMC	**C4.5**	**SMO**
HPosDec	0.00% ± 0.00%	0.00% ± 0.00%	0.00% ± 0.00%
HPosCount	0.00% ± 0.00%	0.00% ± 0.00%	14.24% ± 1.03%
HParDec	24.00% ± 25.50%	9.01% ± 5.86%	76.91% ± 2.01%
HParCount	49.99% ± 0.00%	12.25% ± 6.69%	49.94% ± 0.22%

the convergence criteria of 0% test errors even when one of the substructures is partially identified.

The surrogate functions evolved are qualitatively similar to those obtained in the previous section. In all cases, only the six relevant variables were taken in consideration, and specifically, some of their schemata. For example, one of the surrogate functions created for the HPosDec is

$$14.9\overline{x}_5 + 15x_5 - 2.5\overline{x}_3\overline{x}_4 + \overline{x}_3x_4 - 12\overline{x}_0\overline{x}_1\overline{x}_2 - 4\overline{x}_0x_1\overline{x}_2 - 8\overline{x}_0\overline{x}_1x_2 - 4\overline{x}_0x_1x_2, \tag{8}$$

which only contains the six relevant variables $x_0, x_1, \cdots, x_5$.

As expected, for HParDec and HParCount, gESMC yielded poorer results. For HParDec, the average test error was 24% with a high standard deviation. This high deviation is because gESMC yielded a maximally accurate classification model for 50% of the runs. For the rest 50%, gESMC could not discover an accurate structural model. Further investigation showed that this is due to the stochasticity of the holdout estimation. Since we randomly selected 70% of the instances as the training set, the symmetry of the parity problem may be broken leading the greedy search heuristic to yield the accurate structural model. This indicates that introduction of stochasticity might break symmetry of parity-like functions and render the accurate structural model hill-climbable. However, the efficacy of adding exogenous noise to break symmetry needs to be further investigated.

For HPosDec, gESMC was not able to discover the accurate structural model in any of the cases, and therefore yielded a test error of 50%. As mentioned earlier, the reason for this failure is due of the greedy search of the structural model. For the k-bit parity function, since all structural models with substructures of order $k-1$ or lower yield classification models with the same error, the optimal structural model is not hill-climbable. Therefore, the greedy search heuristic fails to identify the accurate structural model and so yields inaccurate classification models. This limitation can easily be alleviated in several number of ways, some of which are outlined in the next section.

Finally, we compare the results of gESMC to those obtained with C4.5 and SMO. All three algorithms perform equivalently in tackling HPosDec; differently, gESMC and C4.5 outperform SMO on HPosCount. However, on HParDec and HParCount, C4.5 outperforms both gESMC and SMO. Nonetheless, as with the

2-bit lower-order blocks, the trees of C4.5 had some irrelevant variables in the decision nodes indicating overfitting to the training data.

These results clearly show that gESMC can discover the accurate structural model provided that it is hill-climbable from lower-order structural models. In the following section, we discuss some approaches to relax this limitation of gESMC. We also discuss ways to represent structural models with overlapping substructures.

6 Discussion

The results presented in the previous section highlighted both the strengths and limitations of gESMC. In this section we discuss some approaches to overcome the limitations of gESMC which have to be further investigated. We discuss approaches to discover accurate structural models even when there is a lack of guidance from lower-level structural models. Moreover, we also address two new issues: how to deal with problems that present non-linearities in the high order function and also discuss ways to represent structural models with overlapping substructures.

6.1 Lack of Guidance from Lower-Order Substructures

As mentioned earlier, the greedy search used in gESMC needs some guidance from lower-order substructural models towards the optimal structural model. That is, in order to discover a k-variable substructure the greedy search needs a classification model built with at least one of the substructures of order 2 to be more accurate than that with substructures of order 1, and the classification model built with at least one of the substructures of order 3 has to be more accurate than those with substructures of order 2 and so on. In the absence of such a guidance, the greedy search may stop because it cannot find any structural model that decreases the classification error. To alleviate this limitation, we propose the following two approaches:

Increase the order of substructural merges. We can increase the order of the linkages that the greedy search does if the test error is high and no better structural model is found. That is, at each iteration, instead of pairwise merges, we could permit higher-order merges if the pairwise merges yield no improvement.

We implemented this approach and tested gESMC on the four hierarchical problems. The results show that gESMC obtained 0% test error in all the four problems, and the structural models were correctly evolved. However, the limitation of this approach is the increase in the complexity and cost of the algorithm which is dictated by the maximum order of linkages permitted (ℓ_{max}):

$$Cost = \binom{\ell}{2} \cdot s + \binom{\ell}{3} \cdot s + \cdots + \binom{\ell}{\ell_{max}} \cdot s, \tag{9}$$

where s is the cost of building a surrogate. Note that the cost of this approach increases with ℓ_{max}. For this reason, we do not consider this approach as a general solution, although it can be really useful in certain problem domains.

Select randomly one of the new structural models. If the test error is high, and the greedy search cannot find any structural model that significantly decreases this test error, a new structural model can be chosen randomly. More sophisticated approaches could be followed, such as using a technique based on *simulated annealing* [14]. In this case, we would accept a structural model with a higher error with the hope of getting a better model in the subsequent iterations.

Preliminary results using this strategy indicates that gESMC yields maximally accurate classification models for problems consisting of lower-order parity blocks with $k > 2$. However, the structural models evolved are slightly more complicated and contain spurious interactions between variables. Nevertheless, these spurious linkages can be removed by analyzing the classification model and the relative contribution of different schemata to the output.

6.2 Non-linearities in the High Order Functions

The problems designed for the experimentation consisted in higher order functions in which each variable contributed independently to the output. The search procedure could be easily replaced to be able to tackle problems with non-linearities in the higher order functions [4] more efficiently. For example, the greedy search of gESMC could be easily replaced by population-based search methods such as *genetic algorithms* [11,7]. This would permit to solve non-linearities in the higher order of the hierarchy at the cost of slightly increasing the computational time, since a population of candidate solutions should be evaluated and evolved.

6.3 Creating Structural Models with Overlapping Substructures

Finally, we look at problems with overlapping linkages where some variables interact with different groups of variables depending on the input. A widely used test problem with overlapping linkages is the *multiplexer* problem [12,31], which is defined as follows. Given a bit string of length ℓ, where the first $\log_2 \ell$ bits are the *address bits* and the remaining bits are the *position bits*, the output is the value of the position bit referred by the decimal value of the address bits. For example, for the 6-bit multiplexer, f(<u>00</u> <u>0</u>101)=0 and f(<u>10</u> 10<u>1</u>1)=1. Thus, a surrogate with a group formed by all the address bits and the corresponding position bit as a basis accurately determines the output.

We tested gESMC on the 6-bit and 11-bit multiplexer problems. The structural models evolved contained all the address and the position bits in the same linkage group. For example, we obtained the following structural model for the 6-bit multiplexer:

$$[x_0 x_1 x_2 x_3 x_4 x_5],$$

which resulted in a 0% test error. Since gESMC builds structural models with non-overlapping substructures, one way to handle overlapping substructures is by grouping the substructures together. However, such a merger is unnecessary and other methods which can build structural model with overlapping surrogates such as the design structure matrix genetic algorithm (DSMGA) [35,17], can evolve a structural model such as:

$$[x_0x_1x_2][x_0x_1x_3][x_0x_1x_4][x_0x_1x_5],$$

The above structural model also yields a surrogate with 0% test error, and gives more information than the former one. Therefore, we will investigate the use of DSMGA and other similar methods that can discover structural models with overlapping variables.

7 Summary and Conclusions

In this paper, we proposed a methodology for learning by building a classification model that uses the structural and surrogate model of a data set. First, we discover the structural model of a set of examples, identifying salient groups of interacting variables to determine the output. Then, the structural model is used to infer the functional form of a surrogate function and the coefficients of the surrogate are estimated using linear regression. Finally, using the substructural surrogate, we build a classification model to predict the class of a given new set of inputs.

We presented gESMC, an implementation of the methodology which uses a greedy search heuristic to search for the structural, surrogate, and classification models that minimize the classification error. Without any problem knowledge, gESMC starts with a simplest model of independent variables and proceeds to explore more complex structural models untill the classification error no longer improves or is below a user-defined threshold.

We ran gESMC on four hierarchical test problems. We compared the models evolved by gESMC with those created by C4.5 and SMO. The empirical observations evidenced that gESMC significantly outperforms C4.5 and SMO in problems that consisted of 2-bit low order blocks in terms of learning accuracy and interpretability. Moreover, one of the main differences between gESMC and other learners is highlighted: gESMC detects the structure of the data and uses it to predict the class of given inputs. In essence, gESMC not only yields accurate classification models, but also the classification models evolved are *interpretable*. That is, gESMC not only provides the classification model, but also the structure of the data, making it amenable to human interpretation.

Along with these strengths, the results also highlighted some limitations of the particular implementation of the methodology, gESMC. Specifically, the accuracy of the structural model to capture salient variable interactions depends on the guidance from lower-order substructures. Therefore, the accuracy of the structural model and consequently the accuracy of the classification model suffers when there is no guidance from lower-order substructures. This limitation

is expected provided that we use a minimum description length style metric and also a greedy search heuristic that only considers pairwise merges of the substructures. Several approaches were outlined to overcome this limitation, serving as a basis for further research on substructural surrogates for learning decomposable classification problems.

Acknowledgments

We thank the support of *Enginyeria i Arquitectura La Salle*, Ramon Llull University, *Ministerio de Ciencia y Tecnología* under project TIN2005-08386-C05-04, and *Generalitat de Catalunya* under Grants 2005FI-00252 and 2005SGR-00302.

This work was also sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-03-1-0129, the National Science Foundation under grant ITR grant DMR-03-25939 at the Materials Computation Center. The U.S. Government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the National Science Foundation, or the U.S. Government.

References

1. Baluja, S.: Incorporating a priori Knowledge in Probabilistic-Model Based Optimization. In: Pelikan, M., Sastry, K., Cantú-Paz, E. (eds.) Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications ch. 9, pp. 205–219. Springer, Berlin (2006)
2. Bernadó-Mansilla, E., Garrell, J.M.: Accuracy-Based Learning Classifier Systems: Models, Analysis and Applications to Classification Tasks. Evolutionary Computation 11(3), 209–238 (2003)
3. Butz, M.V.: Rule-Based Evolutionary Online Learning Systems: A Principled Approach to LCS Analysis and Design. In: Studies in Fuzziness and Soft Computing, vol. 109. Springer, Heidelberg (2006)
4. Butz, M.V., Pelikan, M., Llorà, X., Goldberg, D.E.: Automated Global Structure Extraction for Effective Local Building Block Processing in XCS. Evolutionary Computation 14(3), 345–380 (2006)
5. Drapper, N.R., Smith, H.: Applied Regression Analysis. John Wiley & Sons, New York (1966)
6. Gibson, J.J.: The Ecological Approach to Visual Perception. Lawrence Erlbaum Associates, Mahwah (1979)
7. Goldberg, D.E.: Genetic Algorithms in Search, Optimization & Machine Learning, 1st edn. Addison Wesley, Reading (1989)
8. Goldberg, D.E.: The Design of Innovation: Lessons from and for Competent Genetic Algorithms, 1st edn. Kluwer Academic Publishers, Dordrecht (2002)
9. Harik, G.: Linkage Learning via Probabilistic Modeling in the ECGA. Technical report. University of Illinois at Urbana-Champaign, Urbana, IL (January 1999) (IlliGAL Report No. 99010)

10. Harik, G.R., Lobo, F.G., Sastry, K.: Linkage Learning via Probabilistic Modeling in the ECGA. In: Pelikan, M., Sastry, K., Cantú-Paz, E. (eds.) Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications ch. 3, pp. 39–61. Springer, Berlin (2006) (Also IlliGAL Report No. 99010)
11. Holland, J.H.: Adaptation in Natural and Artificial Systems. The University of Michigan Press (1975)
12. De Jong, K.A., Spears, W.M.: Learning Concept Classification Rules Using Genetic Algorithms. In: Proceedings of the International Joint Conference on Artificial Intelligence, Sidney, Australia, pp. 651–656 (1991)
13. Keerthi, S.S., Lin, C.J.: Asymptotic Behaviors of Support Vector Machines with Gaussian Kernel. Neural Computation 15(7), 1667–1689 (2003)
14. Korst, J., Aarts, E.: Simulated Annealing and Boltzmann Machines. Wiley-Interscience, New York (1997)
15. Kovacs, T.: Deletion Schemes for Classifier Systems. In: GECCO 1999: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 329–336. Morgan Kaufmann, San Francisco (1999)
16. Llorà, X., Sastry, K., Goldberg, D.E., de la Ossa, L.: The χ-ary extended compact classifier system: Linkage learning in Pittsburgh LCS. In: Proceedings of the 2006 Genetic and Evolutionary Computation Conference Workshop Program. ACM Press, Berlin (2006) (Also IlliGAL Report No. 2006015)
17. Llorà, X., Sastry, K., Yu, T.-L., Goldberg, D.E.: Do not match, inherit: Fitness surrogates for genetics-based machine learning. In: Proceedings of the 2007 Genetic and Evolutionary Computation Conference, vol. 2, pp. 1798–1805 (2007)
18. Pelikan, M.: Hierarchical Bayesian Optimization Algorithm: Toward a new Generation of Evolutionary Algorithms. Springer, Berlin (2005)
19. Pelikan, M., Sastry, K.: Fitness inheritance in the Bayesian optimization algorithm. In: Proceedings of the 2004 Genetic and Evolutionary Computation Conference, vol. 2, pp. 48–59 (2004) (Also IlliGAL Report No. 2004009)
20. Pelikan, M., Sastry, K., Cantú-Paz, E. (eds.): Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications. Studies in Computational Intelligence, vol. 33. Springer, Heidelberg (2006)
21. Platt, J.: Fast Training of Support Vector Machines using Sequential Minimal Optimization. In: Advances in Kernel Methods - Support Vector Learning, pp. 557–563. MIT Press, Cambridge (1998)
22. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, San Mateo (1995)
23. Rao, C.R., Toutenburg, H.: Linear Models: Least Squares and Alternatives. Springer, Berlin (1999)
24. Recktenwald, G.: Numerical Methods with MATLAB: Implementations and Applications. Prentice Hall, Englewood Cliffs (2000)
25. Sastry, K., Goldberg, D.E.: Probabilistic Model Building and Competent Genetic Programming. In: Riolo, R.L., Worzel, B. (eds.) Genetic Programming Theory and Practise, ch. 13, pp. 205–220. Kluwer, Dordrecht (2003)
26. Sastry, K., Lima, C.F., Goldberg, D.E.: Evaluation Relaxation Using Substructural Information and Linear Estimation. In: GECCO 2006: Proceedings of the 8th annual Conference on Genetic and Evolutionary Computation, pp. 419–426. ACM Press, New York (2006)
27. Sastry, K., Pelikan, M., Goldberg, D.E.: Efficiency enhancement of genetic algorithms via building-block-wise fitness estimation. In: Proceedings of the IEEE International Conference on Evolutionary Computation, pp. 720–727 (2004) (Also IlliGAL Report No. 2004010)

28. Simon, H.A.: Sciences of the Artificial. MIT Press, Cambridge (1969)
29. Dietterich, T.G.: Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms. Neural Comp. 10(7), 1895–1924 (1998)
30. Vapnik, V.: The Nature of Statistical Learning Theory. Springer, New York (1995)
31. Wilson, S.W.: Quasi-Darwinian Learning in a Classifier System. In: 4th IWML, pp. 59–65. Morgan Kaufmann, San Francisco (1987)
32. Wilson, S.W.: Classifier Fitness Based on Accuracy. Evolutionary Computation 3(2), 149–175 (1995)
33. Wilson, S.W.: Generalization in the XCS Classifier System. In: 3rd Annual Conf. on Genetic Programming, pp. 665–674. Morgan Kaufmann, San Francisco (1998)
34. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques, 2nd edn. Morgan Kaufmann, San Francisco (2005)
35. Yu, T.-L.: A matrix approach for finding extrema: Problems with modularity, hierarchy, and overlap. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL (2006)

Empirical Evaluation of Ensemble Techniques for a Pittsburgh Learning Classifier System

Jaume Bacardit[1,2] and Natalio Krasnogor[1]

[1] Automated Scheduling, Optimization and Planning research group, School of Computer Science, University of Nottingham, Jubilee Campus, Wollaton Road, Nottingham, NG8 1BB, UK
{jqb,nxk}@cs.nott.ac.uk
[2] Multidisciplinary Centre for Integrative Biology, School of Biosciences, University of Nottingham, Sutton Bonington, LE12 5RD, UK

Abstract. Ensemble techniques have proved to be very successful in boosting the performance of several types of machine learning methods. In this paper, we illustrate its usefulness in combination with GAssist, a Pittsburgh-style Learning Classifier System. Two types of ensembles are tested. First we evaluate an ensemble for consensus prediction. In this case several rule sets learnt using GAssist with different initial random seeds are combined using a flat voting scheme in a fashion similar to bagging. The second type of ensemble is intended to deal more efficiently with ordinal classification problems. That is, problems where the classes have some intrinsic order between them and, in case of misclassification, it is preferred to predict a class that is close to the correct one within the class intrinsic order. The ensemble for consensus prediction is evaluated using 25 datasets from the UCI repository. The hierarchical ensemble is evaluated using a Bioinformatics dataset. Both methods significantly improve the performance and behaviour of GAssist in all the tested domains.

1 Introduction

Ensemble learning, a family of techniques established for more than a decade in the Machine Learning community, provides performance boost and robustness to the learning process by integrating the collective predictions of a set of models in some principled fashion [1]. This family of techniques covers many different approaches, the two most representative methods being Bagging [2] and Boosting [3].

This paper presents the empirical evaluation of two types of ensemble techniques that integrate the collective predictions of models generated using Learning Classifier Systems (LCS) methods. Specifically, we will use the GAssist [4] LCS, a system belonging to the Pittsburgh approach of LCSs, that has shown to generate very compact and accurate solutions for a variety of datasets [5,4,6,7,8].

The first of these two approaches will consist in a simple consensus voting of an ensemble of rule sets generated by running GAssist several times on the

J. Bacardit et al. (Eds.): IWLCS 2006/2007, LNAI 4998, pp. 255–268, 2008.

same dataset with different initial random seeds. This is conceptually similar to Bagging, but its implementation is even simpler.

The second type of ensemble is designed to solve problems of ordinal classification [9]. That is, when the classes of the problem have some intrinsic order. We do this with two goals in mind: (1) improving the performance of GAssist for these datasets and (2) when a misclassification occurs, to try to keep the errors localized by attempting to minimize the distance between the actual and predicted classes according to the intrinsic class order. Our ensemble approach takes a N classes dataset and generates N-1 hierarchically structured binary datasets from it. As an example, from a 10 classes dataset, first we would learn how to separate between the examples with a class ≤ 5 and examples with class > 5. Then, using the examples of class ≤ 5 we would learn how to separate between ≤ 2 and > 2. The same would happen with examples with class > 5. At the end of this process we would have a hierarchy of 9 binary classifiers. GAssist aim is to learn these N-1 binary datasets. Afterwards, the ensemble will integrate the predictions of the N-1 binary models into a final prediction of the N classes domain. This kind of ensemble has been shown to be useful for a series of Bioinformatics datasets [6,7,10,11,8,12] which give raise to ordinal classification problems.

The rest of the paper is structured as follows: First, section 2 describes some related work and compares it to the ensemble mechanisms studied in this paper. Next, section 3 contains the main characteristics of GAssist, the Pittsburgh LCS used in this paper. Section 4 describes and evaluates the first type of ensemble studied in this paper, the consensus voting ensemble, and section 5 the second type, the hierarchical ensemble for ordinal classification. Finally, section 6 discusses our findings and suggests possible directions for future research. Will describe the conclusions and further work of the paper.

2 Related Work

Usually, there are two questions that have to be addressed when building and using an ensemble that integrates the predictions of several models:

- What data is used to train each model?
- How are the individual model predictions integrated to produce a final ensemble prediction?

In the case of Bagging [2], N views of the training set are generated by a sampling with replacement procedure, and each of the N models in the ensemble is trained with one of these views. After that, the ensemble prediction process follows a simple majority voting: the ensemble will predict the most frequent class from the ones predicted by the N members of the ensemble. The first of the two types of ensembles studied in this paper shares the same decision mechanism as Bagging, a consensus voting of the generated models. The difference with Bagging lays in the way that the models are generated. In our ensemble we have a single dataset, and the different models are generated by learning this dataset

feeding the GAssist LCS with different random seeds, which results in a simpler implementation.

The aim of bagging is to generate models that complement each other. This is achieved implicitly by the sampling process used to generate the models. On the other hand, Boosting [3] achieves the same aim in an explicit way. This method generates the models and the dataset in an iterative way. The first model uses the original training data, and the later models focus on learning the examples that were mis-classified by the previous models. This is achieved by weighting the instances based on their mis-classification rate on previous models. Finally, the ensemble prediction is a weighted voting process, where the weight of a model is based on its error over the training data used to generate it.

There are few examples of the use of ensembles in the LCS community. Llorà et al. [13] studied several policies to select the representative candidates from the final population of a Pittsburgh LCS, but did not integrate individuals from several populations. Also, Bull et al. [14] investigated the use of an ensemble of complete LCS populations using an island model. Their ensemble took place during the learning process, not afterwards unlike the approach of this paper.

There are several methods reported in the literature to perform ordinal classification by means of an ensemble. For instance, the method proposed by Frank and Hall [9] takes advantage of learning techniques that can produce a probability of an instance belonging to a certain class and divides the learning process of an N class ordinal domain into N-1 binary domains in the following way:

1. This method needs models that can produce class probability estimates
2. For a given domain D with ordered classes ranging from 1 to k
3. $k-1$ binary domains D_i .. D_{k-1} are generated, where the class definition for domain i will be defined by the predicate $D > i$. That is, the subdomain D_1 will predict if the class of the examples is greater than 1, D_2 will predict if the class of the examples is greater than 2, ...
4. Models for these $k-1$ domains are generated
5. For each new unseen instances, the probability that this instance belongs to each of the k classes is computed as follows:
   - $P(D = 1) = 1 - P(D > 1)$
   - $P(D = i) = P(D > i - 1) - P(D > i), 1 < i < k$
   - $P(D = k) = P(D > k - 1)$
6. The ensemble predicts the class with higher probability

This method generates $k-1$ datasets as in our hierarchical ensemble method. However all the binary datasets of this method have the same number of instances as the original dataset, while in our method some of the datasets only need to learn a subpart of the domain (a certain sub-range of the k ordinal classes) and thus only need to contain the instances of the relevant classes.

An alternative way of doing ordinal classification was proposed by Kramer et al. [15]. Instead of dividing the problem in several sub-problems and combining the models learned from them, they treat the dataset as a regression problem, using the S-CART [16] regression trees induction method, and then map the continuous predictions provided by S-CART into some of the discrete ordinal

classes. Two types of policies are studied. The first of them is a simple rounding of the outputs of the unmodified S-CART into the nearest class. The second policy is to modify internally S-CART so that it produces integer predictions corresponding to the discrete ordinal classes.

3 The GAssist Learning Classifier System

GAssist [4] is a Pittsburgh Genetic–Based Machine Learning system descendant of GABIL [17]. The system applies a near-standard generational GA that evolves individuals that represent complete problem solutions. An individual consists of an ordered, variable–length rule set.

Using the rules of an individual as an ordered set to perform the match process allows the creation of very compact rule sets by the use of default rules. We use an existing mechanism [18] to explicitly exploit this issue and determine automatically the class for the default rule.

We have used the GABIL [17] rule-based knowledge representation for nominal attributes and the adaptive discretization intervals (ADI) rule representation [4] for real-valued ones. To initialize each rule, the system chooses a training example and creates a rule that guarantees to cover this example [19].

A fitness function based on the Minimum Description Length (MDL) principle [20] is used. The MDL principle is a metric applied to a theory (a rule set

Table 1. GAssist configuration for the tests reported in the paper

Parameter	Value
General parameters	
Crossover probability	0.6
Selection algorithm	Tournament
Tournament size	3
Population size	400
Individual-wise mutation probability	0.6
Initial #rules per individual	20
Rule Deletion operator	
Iteration of activation	5
Minimum number of rules	#active rules + 3
MDL-based fitness function	
Iteration of activation	25
Initial theory length ration	0.075
Weight relax factor	0.9
ADI rule representation	
Split and merge probability	0.05
Initial reinitialize probability	0.02
Final reinitialize probability	0
#bins of uniform-width discretizers	4,5,6,7,8,10,15,20,25
Maximum number of intervals	5

here) which balances its complexity and accuracy. Our specific MDL formulation promotes rule sets with as few rules as possible as well as rules containing predicates as simple as possible. The details and rationale of this fitness formula are explained in [4].

The system also uses a windowing scheme called ILAS (incremental learning with alternating strata) [21] to reduce the run-time of the system. This mechanism divides the training set into several non-overlapping strata and chooses a different stratum at each GA iteration for the fitness computations of the individuals. ILAS empirically showed in previous experiments not only to reduce the computational cost of GAssist but also to apply generalization pressure (complementary to the one applied by the MDL-based fitness function) that helped generating more compact and accurate solutions.

Parameters of the system are described in table 1.

4 Ensembles for Consensus Prediction

The evaluated ensemble technique follows these steps:

1. GAssist is run N times on the unmodified training set, each time using a different seed to initialize the pseudo-random numbers generator
2. From each of these N runs a rule set is extracted. This rule set corresponds to the best individual of the population, evaluated using the training set
3. For each instance in the test set, the N members of the ensemble produce a prediction. The majority class of these predictions is used

This ensemble technique is very similar to Bagging, with just one difference: all models (rule sets) are learned using the same training data: the original training set.

Each rule set produced by GAssist is stored in text format as its phenotype representation: the actual ordered predicates in conjunctive normal form that constitute a rule set. Moreover, when these rule sets are dumped to text format, only the relevant attributes are expressed. This means that the ensemble code will not make unnecessary calculations for the match process of the irrelevant attributes. To illustrate the text format used to express the rules generated by GAssist, figure 1 contains an example of a rule set for the Wisconsin Breast Cancer domain generated by GAssist.

4.1 Empirical Evaluation

In order to evaluate the performance of the ensemble method described in this paper we have used a test suite of 25 datasets that represent a broad range of domains in respect to number of attributes, instances, type, etc. These problems were taken from the University of California at Irvine (UCI) repository [22], and their features are summarized in table 2.

The datasets are partitioned using the standard stratified ten-fold cross-validation method. Three different sets of 10-cv folds have been used. Also, the

1:Att Clump_Thickness is [<9.4] and Att Cell_Size_Uniformity is [<4.6] and Att Cell_Shape_Uniformity is [<6.4] and Att Marginal_Adhesion is [<7.75] and Att Single_Epi_Cell_Size is [<5.5][>7.75] and Att Bare_Nuclei is [<5.5] and Att Normal_Nucleoli is [<4][5.5,8.5] and Att Mitoses is [<6.76][>7.12] → benign
2:Att Cell_Size_Uniformity is [<1.9] and Att Single_Epi_Cell_Size is [<7.75] and Att Normal_Nucleoli is [<4][5.5,8.5] → benign
3:Default rule → malignant

Fig. 1. Rule set generated by GAssist for the Wisconsin Breast Cancer dataset

Table 2. Features of the datasets used in this paper. #Inst. = Number of Instances, #Attr. = Number of attributes, #Real = Number of real-valued attributes, #Nom. = Number of nominal attributes, #Cla. = Number of classes, Dev.cla. = Deviation of class distribution.

Dataset Properties						
Code	#Inst.	#Attr.	#Real	#Nom.	#Cla.	Dev.cla.
bal	625	4	4	—	3	18.03%
bpa	345	6	6	—	2	7.97%
bre	286	9	—	9	2	20.28%
cmc	1473	9	2	7	3	8.26%
col	368	22	7	15	2	13.04%
cr-a	690	15	6	9	2	5.51%
gls	214	9	9	—	6	12.69%
h-c	303	13	6	7	2	4.46%
hep	155	19	6	13	2	29.35%
h-h	294	13	6	7	2	13.95%
h-s	270	13	13	—	2	5.56%
ion	351	34	34	—	2	14.10%
irs	150	4	4	—	3	—
lab	57	16	8	8	2	14.91%
lym	148	18	3	15	4	23.47%
pim	768	8	8	—	2	15.10%
prt	339	17	—	17	21	5.48%
son	208	60	60	—	2	3.37%
thy	215	5	5	—	3	25.78%
vot	435	16	—	16	2	11.38%
wbcd	699	9	9	—	2	15.52%
wdbc	569	30	30	—	2	12.74%
wine	178	13	13	—	3	5.28%
wpbc	198	33	33	—	2	26.26%
zoo	101	16	—	16	7	11.82%

experiments were repeated 15 times with different random seeds. This means that the GAssist results for each dataset included 450 runs, either by averaging the test accuracy of each of these 450 runs or by using the ensemble technique.

Student t-tests with a confidence interval of 95% were used to determine whether significant differences between the performance of the individual runs of GAssist and the ensemble of these same runs can be measured. The input data for the t-test will be the test accuracy obtained in each of the 30 test sets that we have (3x10-cv). The ensemble code produced one accuracy measure for each test set. The test accuracy of the individual runs of GAssist (15 repetitions for each data set) were computed by averaging these accuracies. The parameters of the system are the ones defined in [19].

Table 3 contains the results of the experiments performed to evaluate the ensemble technique studied in this paper. The table contains, for each dataset, the average accuracy of the 450 individual runs and the average accuracy of the 30 ensembles produced from the individual runs. We can observe how, for all datasets, the ensemble produces higher accuracy than the individual GAssist runs. The average accuracy increase is 2.5%. Also, the accuracy difference was significant in 10 of the 25 datasets, according to the t-tests.

Table 3. Results of the experiments to evaluate the consensus ensemble applied over GAssist runs. A • symbol in a row means that the ensemble was able to significantly outperform the GAssist individual runs according to the t-tests.

Dataset	GAssist acc.	Ensemble acc.
bal	79.0±4.0	82.5±3.8•
bpa	62.4±7.8	65.7±7.7
bre	70.5±7.9	73.0±7.6
cmc	54.4±3.9	55.7±3.6
col	93.3±4.3	96.2±3.2•
cr-a	85.1±4.1	86.0±3.7
gls	66.8±9.5	71.9±8.0•
h-c1	80.4±5.9	83.0±5.2•
h-h	95.7±3.4	96.7±2.7
h-s	80.2±7.6	82.0±6.9
hep	89.8±8.0	93.6±5.5•
ion	92.0±5.2	93.1±5.1
irs	95.3±5.6	95.8±5.6
lab	98.1±5.4	100.0±0.0•
lym	80.8±11.2	84.4±9.9
pim	74.7±4.8	75.6±4.0
prt	47.5±6.7	52.7±6.8•
son	76.6±9.3	84.0±7.5•
thy	92.0±5.7	93.8±5.4
vot	97.1±3.3	97.6±2.7
wbcd	96.1±2.5	96.2±2.3
wdbc	94.3±3.1	95.2±2.7
wine	93.4±5.5	96.3±3.9•
wpbc	75.3±8.3	80.4±7.6•
zoo	92.1±8.0	94.1±6.3
ave	82.5±13.7	85.0±12.9

After showing the benefits of using this kind of ensemble to boost the performance of GAssist, we would like to perform some simple tests to illustrate the impact of the ensemble size in its performance. To this extend, we have reused the 15 rule sets that were previously integrated into a single ensemble to produce alternative ensembles of 5 and 10 rule sets. Three ensembles of 5 rule-sets each were created with non-overlapped rule sets (using rule sets 1-5 for ensemble 1, rule sets 6-10 for ensemble 2 and rule sets 11-15 for ensemble 3). Three ensembles of 10 rules-sets were created, this time with overlapped rule sets (using rule sets 1-10 for ensemble 1, 6-10 for ensemble 2 and 1-5,11-15 for ensemble 3). The performance of the 3 rule sets for each tested ensemble size were averaged.

Table 4 shows the results of these experiments comparing the performance of the ensembles of 5, 10 and 15 rule sets. This time we applied a different statistical tests, the Friedman test, because it is suited to compare multiple methods across different datasets. We have used the test as suggested in [23]. The

Table 4. Comparing ensembles of different sizes (5, 10 and 15 rule-sets per ensemble) applied over rule-sets generated by GAssist

Dataset	5 rule sets	10 rule sets	15 rule sets
bal	81.6±2.9	82.3±3.0	82.5±3.8
bpa	64.4±6.7	65.5±6.6	65.7±7.7
bre	71.7±6.5	73.1±7.2	73.0±7.6
cmc	55.1±3.5	55.2±3.5	55.7±3.6
col	95.7±3.0	96.0±2.9	96.2±3.2
cr-a	85.9±3.8	85.7±3.8	86.0±3.7
gls	70.1±7.3	71.3±7.7	71.9±8.0
h-c1	82.5±5.0	83.0±4.7	83.0±5.2
h-h	96.2±2.6	96.3±2.6	96.7±2.7
h-s	81.6±6.7	82.0±6.5	82.0±6.9
hep	92.1±5.5	92.9±5.0	93.6±5.5
ion	93.1±4.6	93.0±4.9	93.1±5.1
irs	95.6±5.0	95.7±5.3	95.8±5.6
lab	99.3±2.1	99.7±1.3	100.0±0.0
lym	83.7±9.6	84.3±10.1	84.4±9.9
pim	75.5±3.5	75.8±3.5	75.6±4.0
prt	51.6±5.5	52.3±6.1	52.7±6.8
son	81.7±6.5	82.9±7.0	84.0±7.5
thy	93.4±5.0	93.3±5.4	93.8±5.4
vot	97.5±2.6	97.7±2.7	97.6±2.7
wbcd	96.3±2.3	96.2±2.3	96.2±2.3
wdbc	95.1±2.5	95.2±2.8	95.2±2.7
wine	95.9±3.4	96.3±3.3	96.3±3.9
wpbc	79.2±7.1	79.7±7.3	80.4±7.6
zoo	94.1±5.8	93.8±6.2	94.1±6.3
ave	84.4±13.1	84.8±12.9	85.0±12.9

test indicate that indeed there are significant performance differences between the three sizes of ensemble (with a probability of error of $1.1e^{-6}$). The Holm post-hoc test was used to compare a control method (the best method, the 15-rule-sets ensemble) against the other methods. With a confidence level of 95%, the Holm test indicated that the differences between the best ensemble and the other two ensembles are significant. Nevertheless, the average accuracy difference between the 5-rule-sets ensemble and the 15-rule-sets ensemble is only 0.6, and 1.9% over the average GAssist accuracy. This shows how with very few rule sets we can significantly boost the performance of GAssist.

5 Ensembles for Ordinal Classification

5.1 Motivation

The motivation for developing this kind of ensembles comes from our research in Bioinformatics, specially in Protein Structure Prediction (PSP). In this research area there are many features related to different properties of the complex 3D structure of proteins, some of these features are continuous like solvent accessibility [24]. Other features are defined as an integer, potentially having a high cardinality, such as contact number [6] or recursive convex hull [8]. Predicting these features can help improving the general problem of predicting the full 3D structure of a protein. If we have to predict these features using classification techniques we need to discretize them into a certain number of states. Therefore,

we end up generating a problem of ordinal classification and, depending on the chosen number of states, potentially having a high number of classes.

5.2 Ensemble Definition

Our ensemble-based approach at ordinal classification is divided in two parts: (1) decomposition of the original N classes dataset into several binary sub-datasets and (2) integration of the models generated for each dataset to produce a final N-classes prediction.

Hierarchical datasets generation. The generation of the binary datasets works as follows:

1. We have an original dataset with N ordinal classes
2. We select a certain cut-point between the classes with the following criterion: we will select the cut-point that produces the most balanced sets in terms of number of instances at left and right of the cut-point
3. We generate a binary dataset from the N classes dataset: instances belonging to classes below the cut-point will be labelled as class 0. Instances belonging to classes over the cut-point will be labelled as class 1
4. The steps 2 and 3 will be repeated recursively for the instances to the left and to the right of the cut-point, until we arrive to the trivial case: having a binary dataset
5. Finally, GAssist is run several times on each of the N-1 datasets. The binary model at each node of the hierarchy will be a consensus prediction ensemble as defined in the previous section.

This process will effectively convert a N classes ordinal classification domain into N-1 binary classification domains organized in a hierarchical way. The structure of this hierarchical ensemble is represented in figure 2 for a 10 classes dataset. The root node separates between examples with a class < 5 and class ≥ 5. This node will be trained with the whole dataset, assigning class 0 to the examples of the original classes 0 through 4 and class 1 to examples with an original class ranging from 5 to 9. Then, we will take the examples with classes ranging from 0 to 4 (the examples that had class 0 at the root node) and we will train a new node with them. This node will learn how to separate between examples with a class < 3 and a class ≥ 3. Again, a binary dataset will be generated to do so. Examples of classes 0 to 2 will have class 0. Examples of classes 3 and 4 will have class 1. Afterwards, we will use the examples from the original classes 0 through 2 to learn how to separate between classes 0..1 and class 2. Class 2 is a leaf of the hierarchy and no model is needed for this branch. Finally, we would train another model using examples from classes 0 and 1 to learn how to distinguish them. The rest of the hierarchy of the ensemble would be generated and trained in a similar way.

Unlike the Frank and Hall approach [9], these N-1 datasets will not have the same number instances as the original training set: The first dataset, corresponding to the root of the hierarchy will have the same number of examples as the

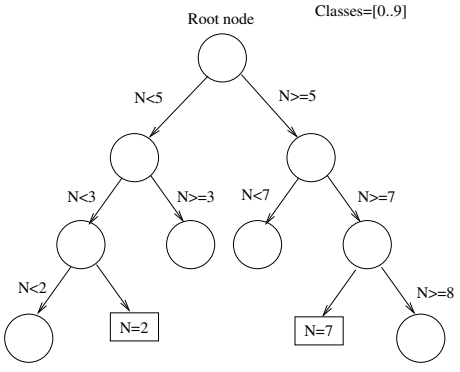


Fig. 2. Representation of the hierarchical ensemble for ordinal classification. Nodes (circles) with no descendants are already binary problems.

original training set. Then, the dataset corresponding to the left branch of the root node will contain the examples having a class less than the cut point of the root node. The dataset corresponding to the right branch of the root will have the examples having a class greater than the cut point of the root node, etc.

Integration of the models into a single prediction. The ensemble predictions will be performed as follows:

1. For each test instance, we will query the model at the root of the hierarchy to determine if its class is lower or higher than the first cut-point.
2. If the root model predicts class 0, the next step is to predict this instance using the model generated for the left branch of the root node
3. If the root model predicts class 1, the next step is to predict this instance using the model generated for the right branch of the root node
4. The process will continue until we reach a leaf node

This process will also be faster than the Frank and Hall method, as only $log_2(k)$ models will be queried, instead of all $k-1$ of them. Moreover, the model used at each node to predict if a given instances is lower or higher than the cut point is an ensemble itself, using the method described in the previous sections.

5.3 Empirical Evaluation of the Hierarchical Ensemble

We will use only one domain to illustrate the performance of this hierarchical ensemble. This domain belongs to the Bioinformatics field, and specifically to protein structure prediction (PSP) [6]. Proteins are heterogeneous molecules that have a complex 3D structure very difficult to determine experimentally and therefore needs to be predicted. Several different features can be predicted from a proteins 3D structure. The domain used in this paper is called prediction of average solvent accessibility [25]. This property is real-valued and therefore we

Table 5. Confusion matrix of the hierarchical classifier on a test set of the average solvent accessibility domain. Each cell contains the percentage of instances of the class in the row predicted as the class in the column.

		Predicted class									
		0	1	2	3	4	5	6	7	8	9
Real class	0	50.0	0.0	0.0	33.3	16.7	0.0	0.0	0.0	0.0	0.0
	1	0.0	11.1	0.0	55.6	22.2	0.0	0.0	0.0	11.1	0.0
	2	18.2	0.0	9.1	36.4	18.2	0.0	18.2	0.0	0.0	0.0
	3	0.0	10.0	10.0	30.0	30.0	0.0	0.0	10.0	10.0	0.0
	4	7.7	0.0	0.0	7.7	30.8	23.1	15.4	15.4	0.0	0.0
	5	0.0	0.0	0.0	0.0	25.0	0.0	50.0	25.0	0.0	0.0
	6	0.0	0.0	0.0	25.0	0.0	0.0	37.5	25.0	12.5	0.0
	7	0.0	0.0	0.0	0.0	0.0	11.1	22.2	22.2	44.4	0.0
	8	0.0	0.0	0.0	0.0	0.0	5.6	11.1	44.4	27.8	11.1
	9	0.0	0.0	0.0	0.0	0.0	8.3	0.0	33.3	25.0	33.3

Table 6. Confusion matrix of the flat ensemble on a test set of the average solvent accessibility domain. Each cell contains the percentage of instances of the class in the row predicted as the class in the column.

		Predicted class									
		0	1	2	3	4	5	6	7	8	9
Real class	0	83.3	0.0	0.0	0.0	0.0	16.7	0.0	0.0	0.0	0.0
	1	77.8	11.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	11.1
	2	54.5	18.2	0.0	0.0	0.0	27.3	0.0	0.0	0.0	0.0
	3	50.0	30.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	20.0
	4	30.8	7.7	0.0	0.0	0.0	30.8	7.7	0.0	15.4	7.7
	5	25.0	0.0	0.0	0.0	0.0	25.0	0.0	0.0	50.0	0.0
	6	12.5	12.5	0.0	0.0	0.0	12.5	0.0	0.0	37.5	25.0
	7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	11.1	22.2	66.7
	8	0.0	0.0	0.0	0.0	0.0	11.1	0.0	0.0	27.8	61.1
	9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	16.7	83.3

need some criterion to convert it into an ordinal set of classes. We will use the equal frequency discretization algorithm [26] for this task, dividing the domain into 10 ordinal classes.

The hierarchical ensemble obtained an accuracy of 23.9±3.0, while a flat ensemble of GAssist models performing normal classification, without any specific knowledge of the intrinsic class order, obtained an accuracy of 22.1±4.6. For reference, please note that Solvent Accessibility prediction accuracy for 10 classes with the kind of input information used in this paper ranges from 20 to 24% [25].

The performance difference is not significant, but the behaviour of both approaches is quite different: Table 5 contains the confusion matrix on one of the test folds for the hierarchical classifier, while table 6 contains the confusion matrix for the flat ensemble. The predictions of this domain are usually fed back into another PSP prediction task. Therefore it is important that, in the case of a mis-classification, the wrong predicted class is close to the real class in the intrinsic class order. The hierarchical classifier achieves this objective much better that the flat ensemble, as most of the predictions appear quite close to the diagonal. Numerically, we can compute the behaviour difference as the average misclassification penalty (AMP), defining the misclassification penalty as

the distance in the intrinsic class order between the real and predicted classes of each test instance. The hierarchical ensemble obtained an AMP of 1.7±0.2, while the flat ensemble obtained an AMP of 2.0±0.2. In this case, the AMP difference between both systems was significant, according to the t-tests with 95% confidence level.

6 Conclusions and Further Work

This paper has empirically studied the use of ensemble techniques in combination with Learning Classifier Systems, specifically using GAssist, a Pittsburgh approach LCS. Two types of techniques are studied. The first kind of ensemble performs a Bagging-style consensus prediction, while the second one is an hierarchical ensemble intended to deal with ordinal classification domains.

These methods are not really new contributions, just variations of already existing techniques, but the experiments reported in the paper illustrate that they are very useful in combination with GAssist. The first one significantly boosts the performance of GAssist on several domains, even when using very few rule sets per ensemble, while the other one helps GAssist minimize the importance of the mis-classifications, which is an issue of concern in ordinal domains.

For future work, it would be interesting to determine how we can tweak GAssist to provide the correct models to the ensemble in order to maximize the accuracy of the consensus prediction. For the hierarchical ensemble, a possible future line is to study other policies to partition the ordinal domain into several binary sub-domains, such as the ones described in the related work section. A question always open in ensemble research and very difficult to address is the interpretability of the ensembles. We would like to investigate how we can improve this issue in our context. Finally, checking the accuracy-computational cost trade-off of these ensemble techniques would be very useful.

Acknowledgements

We acknowledge the support of the UK Engineering and Physical Sciences Research Council (EPSRC) under grants GR/T07534/01 and EP/E017215/1. We are grateful for the use of the University of Nottingham's High Performance Computer.

References

1. Various authors: Special issue on integrating multiple learned models. Machine Learning 36 (1999)
2. Breiman, L.: Bagging predictors. Machine Learning 24, 123–140 (1996)
3. Freund, Y., Schapire, R.E.: Experiments with a new boosting algorithm. In: International Conference on Machine Learning, pp. 148–156 (1996)

4. Bacardit, J.: Pittsburgh Genetics-Based Machine Learning in the Data Mining era: Representations, generalization, and run-time. PhD thesis, Ramon Lull University, Barcelona, Catalonia, Spain (2004)
5. Bacardit, J., Butz, M.V.: Data mining in learning classifier systems: Comparing XCS with GAssist. In: Advances at the frontier of Learning Classifier Systems, pp. 282–290. Springer, Heidelberg (2007)
6. Bacardit, J., Stout, M., Krasnogor, N., Hirst, J.D., Blazewicz, J.: Coordination number prediction using learning classifier systems: performance and interpretability. In: GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation, pp. 247–254. ACM Press, New York (2006)
7. Stout, M., Bacardit, J., Hirst, J.D., Krasnogor, N., Blazewicz, J.: From hp lattice models to real proteins: Coordination number prediction using learning classifier systems. In: Rothlauf, F., Branke, J., Cagnoni, S., Costa, E., Cotta, C., Drechsler, R., Lutton, E., Machado, P., Moore, J.H., Romero, J., Smith, G.D., Squillero, G., Takagi, H. (eds.) EvoWorkshops 2006. LNCS, vol. 3907, pp. 208–220. Springer, Heidelberg (2006)
8. Stout, M., Bacardit, J., Hirst, J.D., Krasnogor, N.: Prediction of recursive convex hull class assignments for protein residues. Bioinformatics (in press, 2008)
9. Frank, E., Hall, M.: A simple approach to ordinal classification. In: Proc 12th European Conference on Machine Learning, pp. 145–156. Springer, Heidelberg (2001)
10. Bacardit, J., Stout, M., Hirst, J.D., Sastry, K., Llora, X., Krasnogor, N.: Automated alphabet reduction method with evolutionary algorithms for protein structure prediction. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO2007), London, England, pp. 346–353. ACM Press, New York (2007)
11. Stout, M., Bacardit, J., Hirst, J.D., Blazewicz, J., Krasnogor, N.: Prediction of residue exposure and contact number for simplified hp lattice model proteins using learning classifier systems. In: Applied Artificial Intelligence, Genova, Italy, pp. 601–608. World Scientific, Singapore (2006)
12. Stout, M., Bacardit, J., Hirst, J.D., Smith, R.E., Krasnogor, N.: Prediction of topological contacts in proteins using learning classifier systems. Soft Computing, Special Issue on Evolutionary and Metaheuristic-based Data Mining (EMBDM) (in press, 2008)
13. Llorà, X., Bacardit, J., Bernadó, E., Traus, I.: Where to go once you have evolved a bunch of promising hypotheses? In: Advances at the frontier of Learning Classifier Systems (2006)
14. Bull, L., Studley, M., Whittley, A.J.B., I.: On the use of rule sharing in learning classifier system ensembles. In: Proceedings of the 2005 Congress on Evolutionary Computation (2005)
15. Kramer, S., Widmer, G., Pfahringer, B., de Groeve, M.: Prediction of ordinal classes using regression trees. Fundam. Inform. 47, 1–13 (2001)
16. Kramer, S.: Structural regression trees. In: Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI 1996), pp. 812–819. AAAI Press/MIT Press (1996)
17. DeJong, K.A., Spears, W.M., Gordon, D.F.: Using genetic algorithms for concept learning. Machine Learning 13, 161–188 (1993)
18. Bacardit, J., Goldberg, D.E., Butz, M.V.: Improving the performance of a pittsburgh learning classifier system using a default rule. In: Kovacs, T., Llorà, X., Takadama, K., Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 2003. LNCS (LNAI), vol. 4399, pp. 291–307. Springer, Heidelberg (2007)

19. Bacardit, J.: Analysis of the initialization stage of a pittsburgh approach learning classifier system. In: GECCO 2005: Proceedings of the Genetic and Evolutionary Computation Conference, vol. 2, pp. 1843–1850. ACM Press, New York (2005)
20. Rissanen, J.: Modeling by shortest data description. Automatica 14, 465–471 (1978)
21. Bacardit, J., Goldberg, D.E., Butz, M.V., Llorà, X., Garrell, J.M.: Speeding-up pittsburgh learning classifier systems: Modeling time and accuracy. In: Yao, X., Burke, E.K., Lozano, J.A., Smith, J., Merelo-Guervós, J.J., Bullinaria, J.A., Rowe, J.E., Tiňo, P., Kabán, A., Schwefel, H.-P. (eds.) PPSN 2004. LNCS, vol. 3242, pp. 1021–1031. Springer, Heidelberg (2004)
22. Blake, C., Keogh, E., Merz, C.: UCI repository of machine learning databases (1998), `http://www.ics.uci.edu/mlearn/MLRepository.html`
23. Demšar, J.: Statistical comparisons of classifiers over multiple data sets. J. Mach. Learn. Res. 7, 1–30 (2006)
24. Rost, B., Sander, C.: Conservation and prediction of solvent accessibility in protein families. Proteins 20, 216–226 (1994)
25. Richardson, C., Barlow, D.: The bottom line for prediction of residue solvent accessibility. Protein Eng. 12, 1051–1054 (1999)
26. Liu, H., Hussain, F., Tam, C.L., Dash, M.: Discretization: An enabling technique. Data Mining and Knowledge Discovery 6, 393–423 (2002)

Technology Extraction of Expert Operator Skills from Process Time Series Data

Setsuya Kurahashi[1] and Takao Terano[2]

[1] University of Tsukuba, 3-29-1 Otsuka, Bunkyo, Tokyo 112-0012, Japan
kurahashi@gssm.otsuka.tsukuba.ac.jp
[2] Tokyo Institute of Technology, 4259 Nagatsuda-Cho Midori-ku, Yokohama 226-8502, Japan
terano@dis.titech.ac.jp

Abstract. Continuation processes in chemical and/or biotechnical plants always generate a large amount of time series data. However, since conventional process models are described as a set of control models, it is difficult to explain complicated and active plant behaviors. To uncover complex plant behaviors, this paper proposes a new method of developing a process response model from continuous time-series data. The method consists of the following phases: (1) Reciprocal correlation analysis; (2) Process response model; (3) Extraction of control rules; (4) Extraction of a workflow; and (5) Detection of outliers. The main contribution of the research is to establish a method to mine a set of meaningful control rules from a Learning Classifier System using the Minimum Description Length criteria and Tabu search method. The proposed method has been applied to an actual process of a biochemical plant and has shown its validity and effectiveness.

1 Introduction

Although there is a lot of theoretical research on Learning Classifier Systems (LCSs), very few applications have been reported in the literature. This paper describes the practical application of LCSs in order to extract plant operation knowledge from actual operation data of a biochemical plant. So far, many kinds of automatic control systems have been established in such plants as chemical plants. Operator confirmation and manual procedures are essential for a wide variety of products used in small quantities requiring stringent quality control, such as advanced materials for Liquid Crystal Display (LCD), pharmaceutical products, and so on. The quality control of biochemical plants has also become one of the most important issues in the field of food-safety.

However, from 2007 onward, many expert operators, members of the "baby-boom generation", will be coming up for retirement. In addition, many manufacturing sites have recruited a lot of foreign laborers and temporary employees, due to global competition. These sites are faced with problems in terms of many less-skilled laborers. The time, when expert operators with excellent skills acquired through long experience will leave their manufacturing sites, one after

J. Bacardit et al. (Eds.): IWLCS 2006/2007, LNAI 4998, pp. 269–285, 2008.

another, is just around the corner. In many manufacturing areas, how to pass on expert operator skills to future generations has become a big issue.

In the past, transfer functions like the delay time function have built up a process model by describing an individual response process. The transfer function is used in analyses of input/output behaviors. It is derived using the Laplace transform in control theory. However, process circumstances might change significantly, according to variations of infused material or operating conditions. Thus, automated acquisition or data mining of processes from actual daily data is desirable to manage these changes. In this research, we propose a heuristic search method for plant operation rules, which could provide guidance on human operators, building up a process response model from a large amount of time series data. The basic principles of the model are 1) to maximize the correlation coefficient among time series data, and 2) to apply LCSs with Minimum Description Length (MDL) criteria [1]. The paper also describes results from applying the proposed method to actual operation data for a biochemical plant.

The remainder of this paper is organized as follows: In the next section, we describe the research objective and the problem description. In the third section, we propose principles of LCS with MDL and improvement rate based MDL criteria. In the forth and fifth sections, we present experimental results and compare them with conventional methods. In the sixth and seventh sections, we present extracting knowledge of workflow from workers and detection of outliers. In the last section, we conclude with a summary.

2 Research Objective

2.1 A Target Plant

We are dealing with a biochemical plant with a distillation tower as in Figure 1. In the distillation tower, low-pressure treatment performs constituent separation after the basic ingredient is infused into the tower. Example: Figure 2 is the normalized data of one biochemical plant. "Normalized" means that each process value is normalized with the same range between 0.0 and 1.0. As the figure shows, it is quite complicated, and seems to be difficult to obtain useful data from the data shown in the figure using conventional methods.

Simply observing normalized data, as shown in Figure 2, is not enough, and it is almost impossible to read what kind of relationship exists among process data. Operators, who are those who actually operate the plant and learn about the process characteristics empirically, control the whole plant by frequently adjusting control settings or by manual operation.

In contrast, with the proposed method it is possible to obtain simple but persuasive knowledge like Figure 3. We will explain this rule in the following sections.

2.2 Problem Description

The purpose of this research is to extract significant information from such time series data that appear to be complicated. The following phases in Figure 4 show how to build a model up in order to analyze the process data.

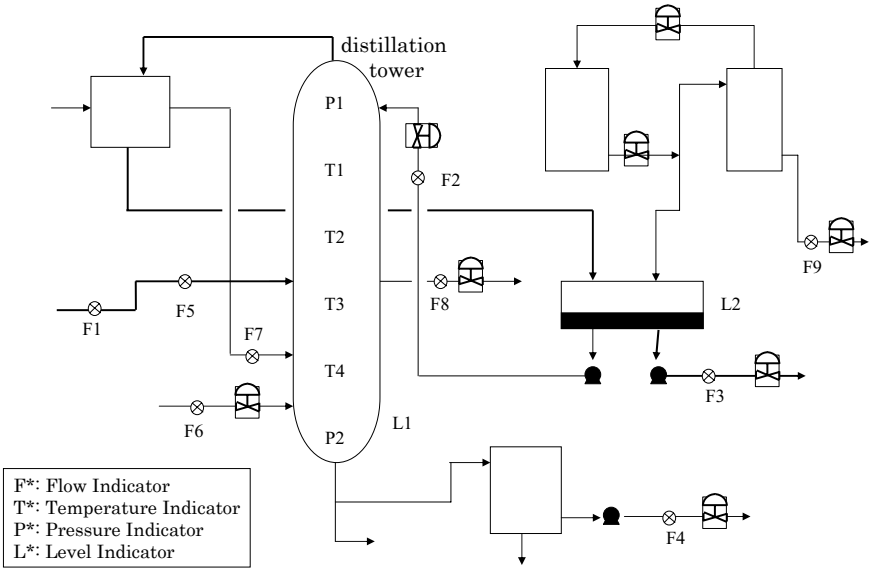


Fig. 1. Outline of the biochemical plant

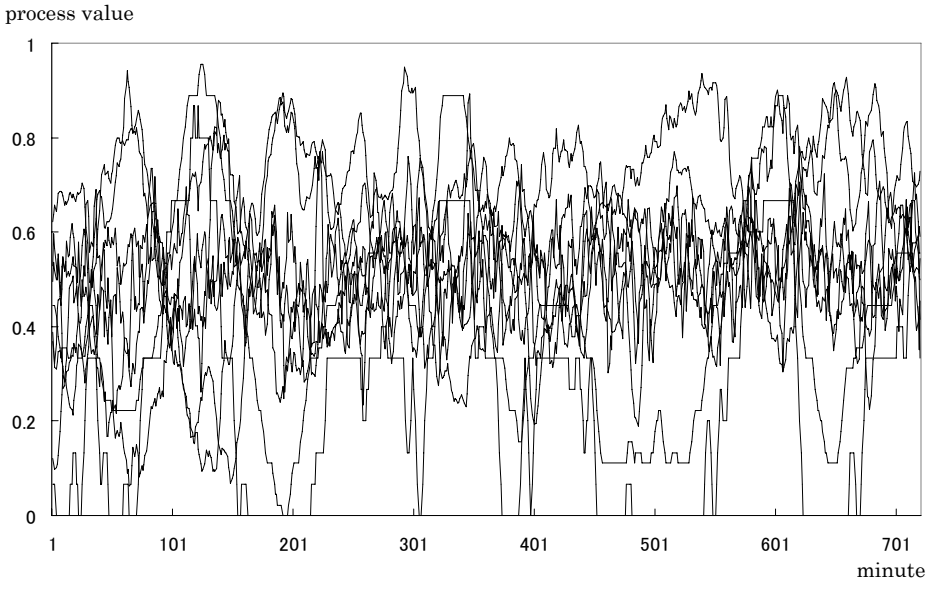


Fig. 2. Normalized process data

$25\% < F3 \leq 50\%$ and $75\% < F4$ and F3 is down then $75\% < T2$

"F3 flow is from 25% to 50%, and F4 flow is 75% or more, and F3 flow is decreasing, T2 temperature becomes 75% or more."

Fig. 3. Acquired knowledge

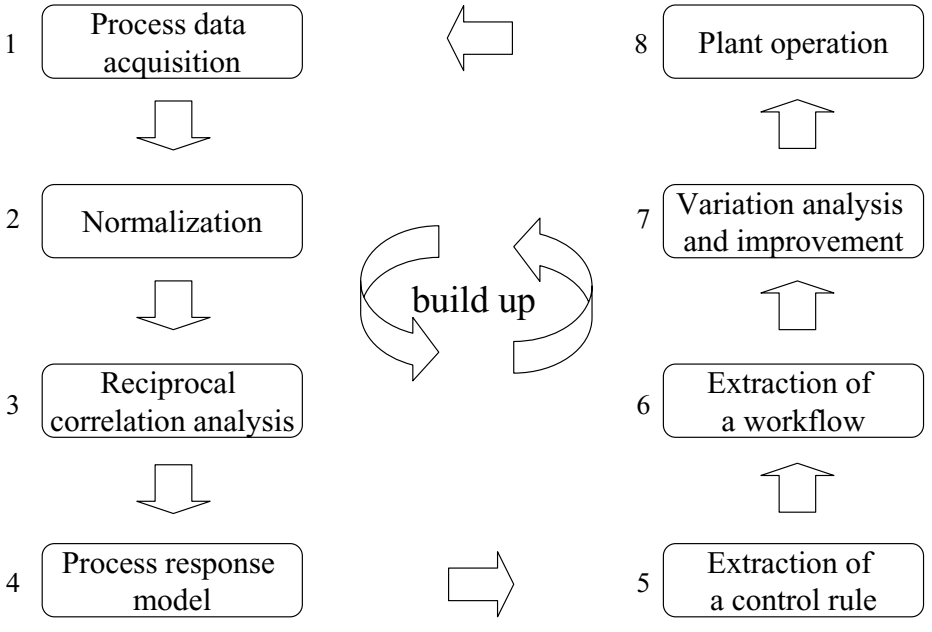


Fig. 4. Analyzing phases

1. Process data acquisition phase
   Various kinds of process data are collected and stored in a database.
2. Normalization phase
   Each piece of process data has a different range, such as 0.0 - 300.0 kPa or 0 - 1600 degrees C. All data are normalized between 0.0 and 1.0 to deal with the same range.
3. Reciprocal correlation analysis phase
   Two sets of normalized process data are selected and searched for the time difference that indicates the biggest correlation between each process value by gradually shifting the time.
4. Process response model phase
   Like the response model, the phase describes the relationship among the process data from the shifted time and the correlation coefficient.
5. Extraction of a control rule phase
   The process extracts the control rule by executing LCS that handles specified process data as a process response model class.

6. Extraction of workflow phase
   The LCS that counts occurring time of operational events and process data is executed so as to find the workflow.
7. Variation analysis and improvement phase
   A current workflow and an extracted workflow are compared, in order to analyze the difference. The old workflow is improved for the operation support system.
8. Plant operation phase
   A plant is operated with the new workflow, and process data is acquired again. This build-up cycle runs repeatedly.

3 Principles of LCS with MDL

3.1 MDL Criteria

Operation rules of such plants require simple and clear descriptions in order for operators to recognize the target process conditions. The concept of complexity determines the data and the model describes it [2]. MDL criteria minimize the complexity of the model and data [1,3]. We use it to minimize the complexity of LCS. MDL criteria are shown as follows [4]: Here, m_1 and m_0 are each occurrence numbers of $y = 1$ and $y = 0$, in data row $y^m = y_1, ..., y_m$ with length m. Here, $m = m_1 + m_0$. Also, c_i indicates 0 where each condition of the former part has a wild card #, or 1 in other cases. And t_i indicates the division number of the process data in each condition; k indicates the number of conditions. Then, the description length of data and model are as follows:

$$dataLength = mH(\frac{m_1}{m}) + \frac{1}{2}\log(\frac{m\pi}{2}) + o(1),$$
$$modelLength = \sum_{i=1}^{k} c_i(1 + \log t_i),$$
$$where, H(x) = -x\log(x) - (1-x)\log(1-x).$$

3.2 Improvement Rate Based MDL Criteria

Although the MDL principle generates a simple and safe model, this does not always means the model is easy to understand. Thus, we apply the improvement rate of association rules in data mining literature [5]. The following formula expresses the improvement rate:

$$improvement = \frac{P(r_i|\mathbf{p})}{P(r_i)},$$

where, $P(r_i)$ expresses the rate that the latter part r_i appears without condition, and $P(r_i|\mathbf{p})$ expresses the rate that the latter part r_i appears with the condition of the former part, $\mathbf{p}$.

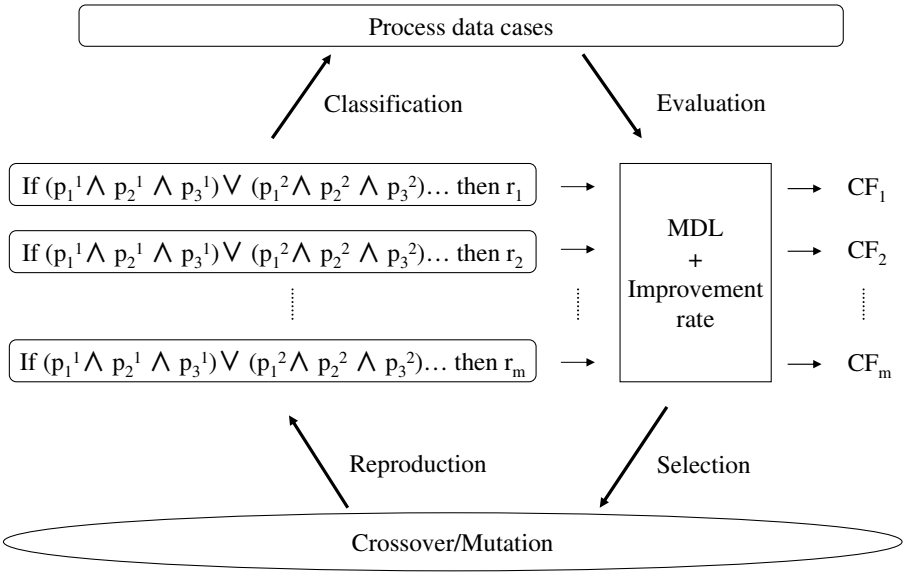


Fig. 5. Learning classifier system

The description length in MDL principles is calculated as follows: It is known that probability distribution $P(\cdot)$, on the assembly of data row $y^m = y_1, ..., y_m$ with length m, exists. Also, the length $L(y^m)$ of binary code string $\phi(y^m)$ can be expressed as

$$L(y^m) = -\log P(y^m).$$

Expressing the occurrence rate in marketing basket analysis through a logarithm with the same description length as the description length of MDL principles, it is possible that the improvement rate is the differential of the information amount between before-refining and after-refining, with certain conditions. Therefore, we handle the improved information amount and the description length in MDL principles at the same time. So, MDL criteria are expanded, in order to maximize the differential of the description length (model length + data length) obtained for classification by LCS. The following method shows the calculation of the learned classifier weight. Here, $dataLength_f$ and $modelLength_f$ express the initial description length of data and model, $dataLength_l$ and $modelLength_l$ express the final description length of data and model. The weight of classifier is calculated by

$$\begin{aligned} DL_{first} &= dataLength_f + modelLength_f, \\ DL_{last} &= dataLength_l + modelLength_l, \\ Weight &= DL_{first} - DL_{last}. \end{aligned}$$

When applying a rule and the rule is simple, the knowledge, which is unknown when there are no rules, reveals another unpredictable rule [6]. The expansion proposed here allows for detailed evaluation of the simple rule that can reveal a valuable fact with copious amounts of information. Evaluating all classifiers hit in the former part, counting the result of its classifier allows for calculation of the estimated value of a classification error. MDL criteria are used to get the weight, selecting the smallest dataLength in the classification of results. Then the learning classifier system is implemented to maximize the weight as a fitness function.

3.3 Learning Classifier System

As a learning classifier system, a modified system based on the original LCS is introduced [7,8,9,10]. This system corresponds to a lot of events, using the learning method it estimates event distribution by random sampling. Each individual consists of a condition part (as *disjunctive normal form*) and of a conclusion. Figure 5 shows the concept of the learning classifier system in this model. First of all, random generated rules, set as classifiers, classify the process data. It is possible to use other techniques such as rules generalized from input data. In this case, we used random generated rules for simplicity. MDL criteria and the improvement rate evaluate these rules and classification results, and set the result to CF_i with MDL or the improvement rate of MDL. Therefore, in the case of MDL criteria, each classifier is selected by MDL value as a maximization problem, and in the case of improvement MDL criteria, it is a minimization problem using the improvement rate of MDL. To each classifier, a new classifier is generated, conducting tournament selection based on the obtained CF_i, crossing and mutation.

Table 1 shows the parameters and CF_i of this Learning Classifier System. The number of classifiers: 200, data: 9 items * 300min, crossover probability: 0.7, mutation probability: 0.005. Each piece of numeric data was divided every 25%

Table 1. LCS Parameters

The number of process data items	9
Sampling time	1min
Collection time	300min
The division number of a condition value	4(25%)
The rate of change of a condition value	25%(up,stability,down)
The division number of a result value	4(25%)
The number of classifiers	200
Selection method	Tournament
Crossover probability	0.7
Mutation probability	0.005 on a gene
CF_i of MDL criteria	$DL_{last} = dataLength_{last} + modelLength_{last}$
CF_i of iMDL criteria	$DL_{first} - DL_{last}$
The number of Tabu lists(section 6)	10
The distance between solutions(section 6)	Hamming distance

after it normalized. It was also classified into downtrend, stability and uptrend, respectively. When **p** is set as a conditional expression of the former part and **r** is set as a conditional expression of the latter part, the structure of the classifier becomes

$$\mathbf{p} = (p_1^1 \wedge p_2^1 ... \wedge p_k^1) \vee (p_1^2 \wedge p_2^2 ... \wedge p_k^2)...,$$
$$\mathbf{r} = r_1, r_2, ..., r_n.$$

The following is an example rule:

$$((T1 \leq 0.25) \wedge (0.25 < F3 \leq 0.5) \wedge (F4\ is\ up)) \vee$$
$$((0.5 < T3 \leq 0.75) \wedge (F5\ is\ down) \wedge (F1\ is\ stable))$$
$$then\ T2 \leq 0.25.$$

The result r_i shows all the possible results that the target event would obtain, and counts the number of hits in all r_i that hit in the former part. This gives an estimated value of reliability for the latter part event, in accord with the agreed-upon event of the former part. The pressure toward generalization is preformed by the MDL metric.

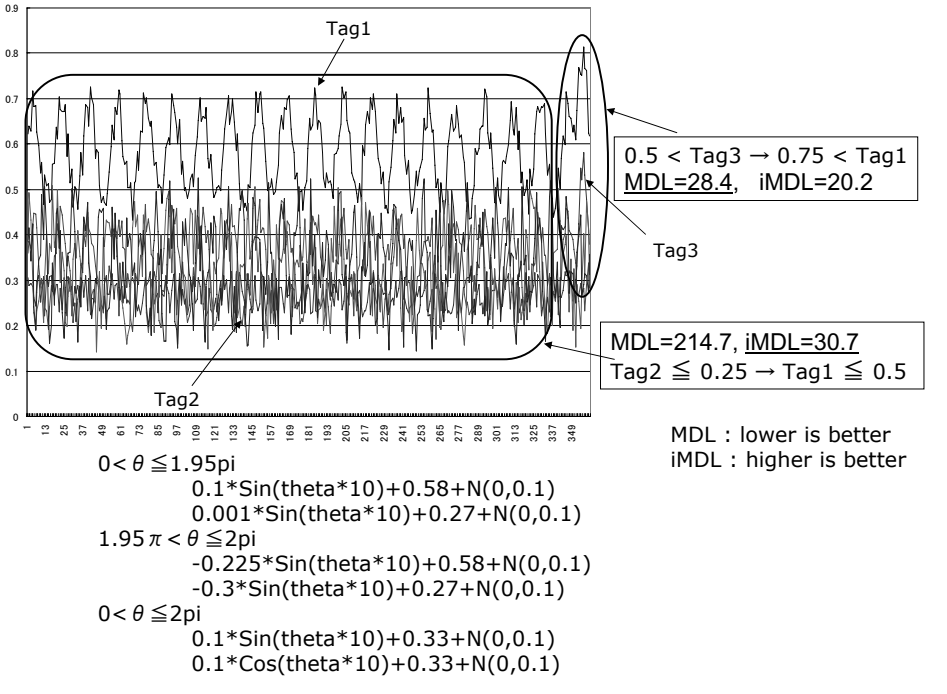


Fig. 6. Comparison between MDL and iMDL

3.4 Comparison between MDL and iMDL

Figure 6 shows experiments about LCS based on MDL(MDL-LCS) and Improvement MDL(iMDL-LCS). These test data were generated by functions of sine, cosine, and normal distribution. Tag1 is the target data of this analysis.

MDL-LCS detected phenomena in a right small circle in Figure 6. The rule was "$0.5 < Tag3$ then $0.75 < Tag1$" (If the process value of Tag3 is more than 0.5, then the process value of Tag1 is more than 0.75). The MDL value was 28.4 at the time.

Although MDL-LCS detected Tag3, it seems to be significantly related to Tag1. On the other hand, iMDL-LCS detected phenomena in a left large circle in Figure 6. This rule is different from the previous rule with MDL-LCS.

iMDL-LCS detected the following rule,"$Tag2 \leq 0.25$ then $Tag1 \leq 0.5$." MDL-LCS has the ability to detect an outlier, but it is easy to discover the rule just by looking, since it stands out. On the contrary, iMDL-LCS has the ability to detect hidden phenomena.

4 Experiments

The proposed method has been applied to practical plant operation data, by building a model according to the method shown in Figure 4.

4.1 The Response Model

In the case of the continuation process, a correlation is recognized among the data. The following operation produces a reciprocal correlation coefficient of time series data:

1. Select time series data **x** and **y** that of couple process tags. When k is set as time shift amount of each tag and $\bar{x}$, $\bar{y}$ are averages, the next formula calculates the next reciprocal correlation coefficient of k, $r_{xy}(k)$.

$$r_{xy}(k) = \frac{\sum_{t=k+1}^{T}(x_{t-k} - \bar{x})(y_t - \bar{y})}{\sqrt{\sum_{t=k+1}^{T}(x_{t-k} - \bar{x})^2}\sqrt{\sum_{t=1}^{T}(y_t - \bar{y})^2}}.$$

2. Obtain k that shows maximum correlation.

$$\max_k r_{xy}(k).$$

This operation, which consists of all data, produces a maximum correlation coefficient table and shift time table. Figure 7 is the result produced by the operation of a reciprocal correlation analysis. It shows coefficients of other process data to T2. Table 2 and Table 3 show a part of maximum correlation coefficients and their shift time.

The above operation leads to the building of a process response model, extracting tags with high correlation from a quantity of time series tag data. Figure 8 shows an example. It shows the structure of process response, by means of time series correlations among process tags and the time shift information.

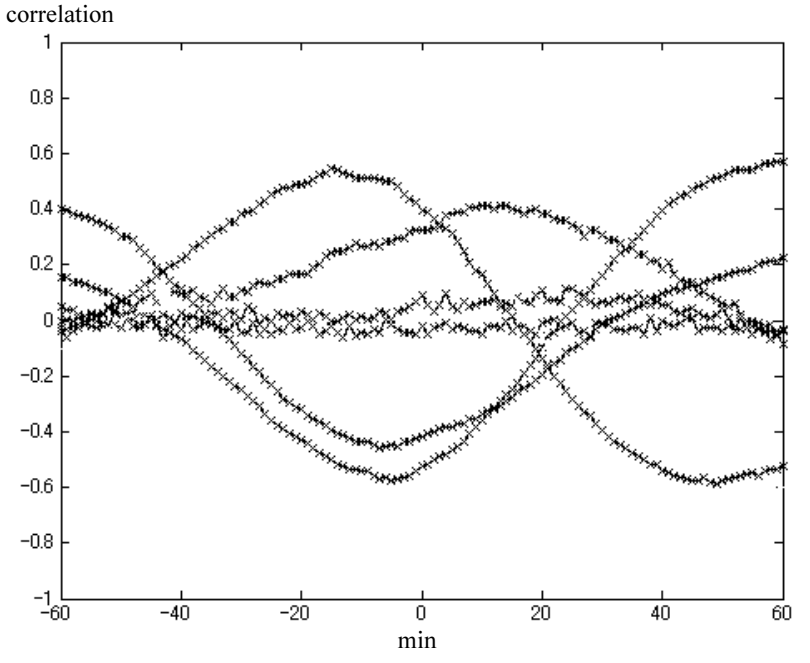


Fig. 7. Reciprocal Correlation Analysis

4.2 Heuristic Search for Operation Rules

In the actual operation, it is significant to discover a control point that makes the final quality stable. The LCS with MDL criteria and the improvement rate, searches for the control rule targeting tag data with high correlation obtained by the process response model. Figure 9 shows an example of the classifier obtained. At this moment, the improvement rate is 3.1, and the MDL value is 32.9 bit. The next example shows the classifier in the case that considers the improvement rate. In this case, the improvement rate is 6.6, and the MDL value is 54.8 bit. The former becomes a simpler model, although the result is close to "commonplace" with a low improvement rate. In the case that gives consideration to the improvement rate, in addition to the MDL value, an unpredictable rule is more easily revealed. As a result of an interview with the person in charge of the

Table 2. Maximum correlation coefficient

Maximum cor	F4	F2	F3	T2	F1
F4	1.00	0.41	-0.32	0.32	-0.28
F2	0.41	1.00	-0.57	-0.63	-0.46
F3	-0.32	-0.57	1.00	-0.80	-0.53
T2	0.32	-0.63	-0.80	1.00	0.66
F1	-0.28	-0.46	-0.53	0.66	1.00

Table 3. Maximum correlation shift time

Shift time	F4	F2	F3	T2	F1
F4	0	-10	-14	-22	-26
F2	10	0	-5	53	-7
F3	14	5	0	-5	-60
T2	22	-53	5	0	-55
F1	26	7	60	55	0

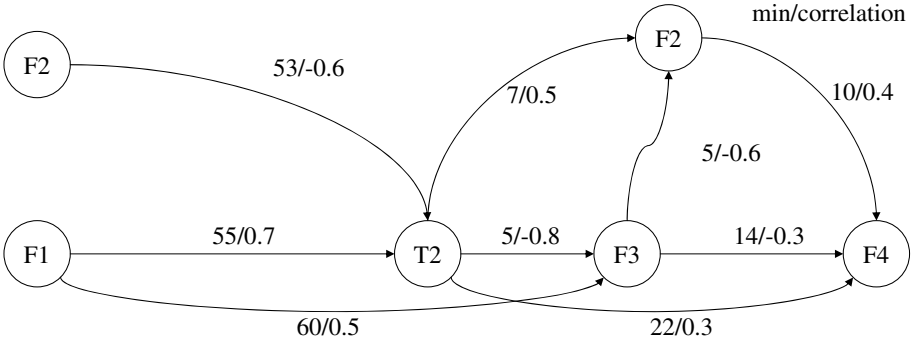


Fig. 8. Process response model

operation, T2(temperature of the tower) provides an important control point that greatly affects product constituent quality in this biochemical plant, and it is too difficult to control the temperature. In order to control it more accurately, the classification rule that operators find hard to be aware of becomes precious information. In the case of Figure 9, F2 flow is expected to be related to T2 temperature, but it is not noticeable that F3 flow transition rate away from T2, or F4 flow are also connected to T2. Furthermore, in Figure 10 too, comparing with the F1 ingredient flow which directly infused into the distillation tower and the F2 return flow, operators found it hard to see that the F3 flow away from T2 is related to T2 temperature. As mentioned above, the application of Minimum Description Length criteria considering the improvement rate results in such unpredictable information.

MDL:
$(75\% < F2)$ and $(75\% < F3)$ then $50\% < T2$
"F2 flow is 75% or more, and F3 flow is 75% or more, then T2 temperature becomes 50% or more."
MDL+Improvement:
$(25\% < F3 \leq 50\%)$ and $(75\% < F4)$ and ($F3$ is down) then $75\% < T2$
"F3 flow is from 25% to 50%, and F4 flow is 75% or more, and F3 flow is decreasing, T2 temperature becomes 75% or more."

Fig. 9. Control rules for Recipe-1

MDL:
(50% $< F1 \leq$ 75% and 25% $< F4 \leq$ 50% and $F4$ is up) or (75% $< F2$) then $T2 \leq$ 50%
MDL+Improvement:
(75% $< F2$ and 50% $< F3 \leq$ 75% and $F4$ is up) or (75% $< F2$) then $T2 \leq$ 25%

Fig. 10. Control rules for Recipe-2

5 Comparison with Conventional Methods

Time-series-data analysis and the time series model have been developed in the field of statistics. In the field of economics also, many models that estimate economic indicators have been reported. There has been a lot of research in data mining to obtain useful information from a large amount of data [11,12,13,14,15,16,17,18,19]. Also the MDL principle has been applied in many fields, for example, Genetic Programming and LCSs as *Bloat* control and generalization pressure [20,21,10]. Several indicators are proposed in order to measure how interesting the extracted knowledge is: J-Measure, i-Measure, I-Measure or *IShannon*-Measure for example [22,23,6]. However, these indicators do not directly represent the length of the model nor the classified data. Moreover, they do not give a human operator specific operation rules.

A simple application of decision tree methods also produces very large trees. For example, our results on actual process data using C4.5 [24] have generated a huge tree with 87 nodes before pruning, 43 nodes even after pruning. It is difficult for human operators to understand the meaning. They need a simple model to represent authentic information.

Table 4 shows results of the learning classifier system with decision tree C4.5, the learning classifier system on the basis of MDL + improvement rate criteria.

Table 4. Decision tree C4.5, LCS on MDL and MDL+Imprv

	MDL	Imprv-rate	Model-length	Errors
C4.5	410.4		297.2bit	12.0%
C4.5rule/pruning	177.0		50.8bit	14.3%
MDL criteria	121.3	3.9	9.8bit	6.7%
MDL+imprv	109.4	5.1	9.8bit	7.0%

The classification error rate indicates a lower value in MDL + improvement rate criteria, than decision tree C4.5 before and after pruning. In the rule description length, the MDL + improvement rate criterion classifier considerably decreases compared to C4.5.

6 Extracting Knowledge of Workflow from Workers

Using the proposed method, we also generated workflow processes for operations. First, we collect event data such as switching (on/off) and value setting operations with time stamps. Then, using the proposed methods, we search for rules

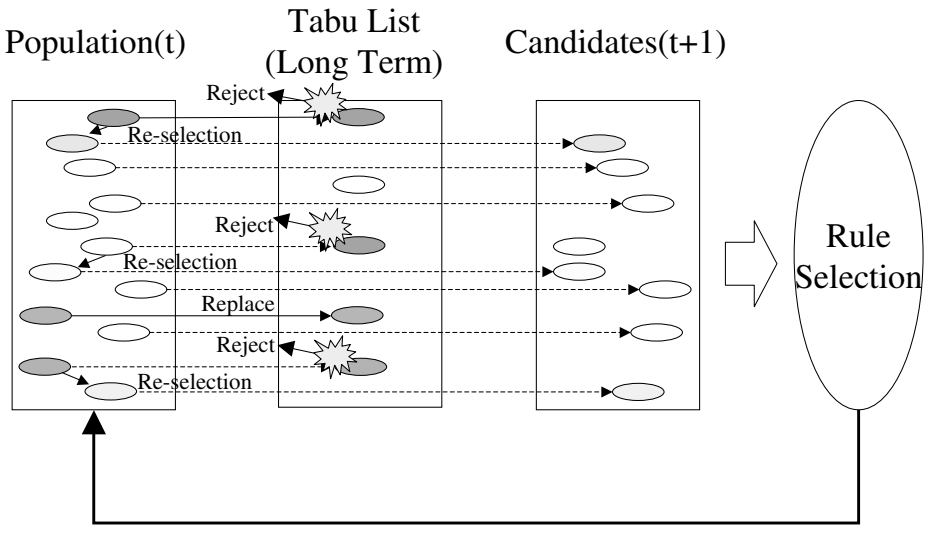


Fig. 11. TABU-LCS

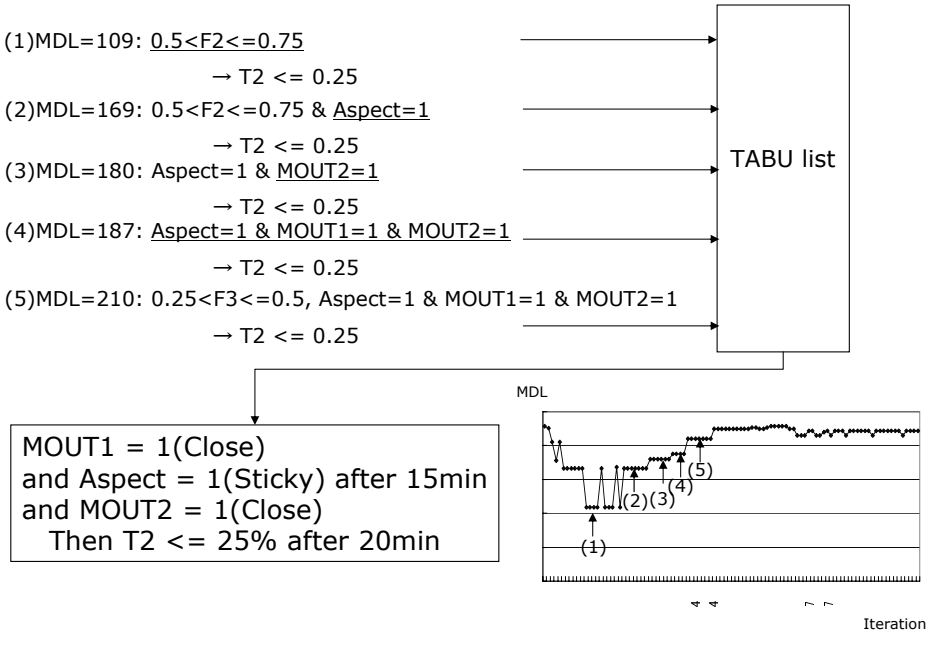


Fig. 12. The experiment of TABU-LCS

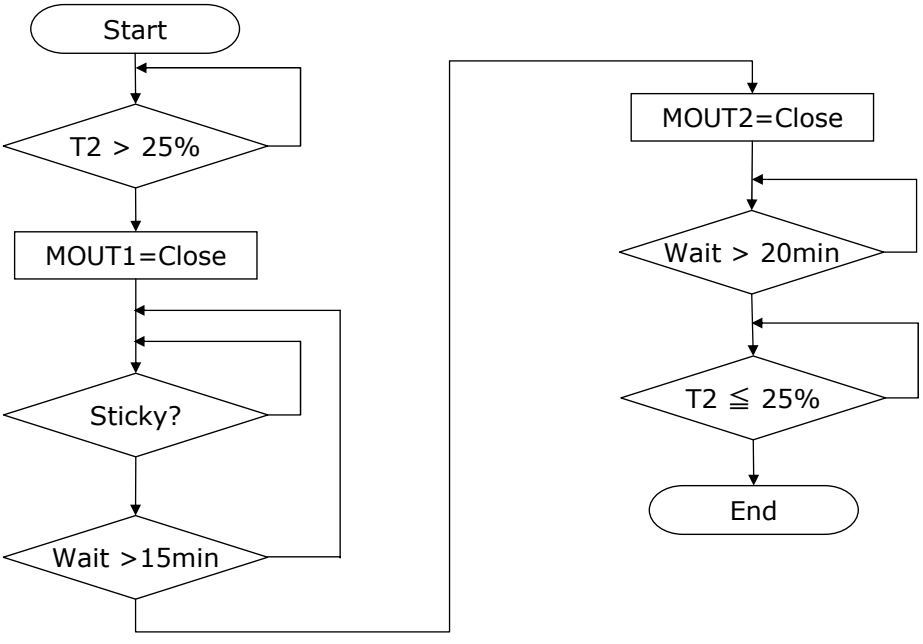


Fig. 13. Expert's workflow

during given time intervals. Finally, we sort the acquired rules with the time key, and then we get the corresponding workflow. However, a workflow consists of several rules that are operated simultaneously on shifted data. MDL-LCS actually extended multiple classifiers originated by Pittsburgh LCS [8,9] and found the following rule.

"$0.5 < F2 \leq 0.75$ then $T2 \leq 0.25$, # then #, ..." It did not find other rules in spite of LCSfs ability to detect sets of rules.

That is why the MDL principle reduces model length, and other rules at the same time are redundant for classification of abnormal conditions. In addition, it is important to find the second or third optimum classifier, since experts often operate various procedures that are different from a typical one.

Hence we utilize a TABU-Search method to solve the problem [25]. TABU-Search prohibits some operations from falling into local solutions. Figure tabu briefly explains LCS with TABU-Search. A TABU list is put in place between the Classifier Population and the Candidates Set. When selecting candidates from the population, if a classifier has the same gene as a classifier (TABU-CF) in the TABU list, it will be refused by the TABU list. Then another classifier is re-selected. Besides, if a classifier has more fitness value than a TABU-CF, it will be replaced as a new TABU-CF.

Figure 12 is the result of experiments with TABU-LCS. First, TABU-LCS found a simple rule whose MDL was 109. This rule was stored in the TABU List. Then TABU-LCS found a second rule whose MDL was 169. TABU-LCS eventually found five rules from the process data.

It's easy to compose a workflow using these rules. Figure 13 shows the workflow from these rules. Although the workflow in this figure is very small, it certainly occurs in this process. The method enables us to extract implicit plant operation knowledge from both manual operation data and process data. Such knowledge is useful in transferring experts' special skills to inexperienced operators.

6.1 Detection of Outliers

Outlier detection is a fundamental issue in data mining. It has been considered in [26], in which statistical outlier detection algorithms have been proposed. However, they were detection algorithms on statistical models. Process data includes a lot of event data, which are records of operations and process alarms, so it is difficult to assume a statistical model. Thus, we used the LCS to detect outliers. The data dealt with the following stages: (1) Deviation between process values and targeted values, called golden batch data, is measured, (2) Cases with deviation of more than a threshold value are selected, (3) Process conditions and workflow of the cases are detected using the LCS we proposed, and (4) Outliers are detected using process conditions and workflow. The results of the experiments have indicated that our model would detect errors of 0.5 percent or less. It was verified that the odds ratio, which was 3.3, was statistically significant.

7 Conclusion

This paper has proposed a new method of extracting plant operation knowledge from time series data using LCSs with the MDL principle and Tabu search. The method has generated useful but simple operation knowledge with high reliability. It enables us to extract implicit plant operation knowledge from both manual operation data and process data. Such knowledge is useful in transferring experts' special skills to inexperienced operators.

The method consists of the following phases: (1) Process data acquisition phase; (2) Normalization phase; (3) Reciprocal correlation analysis phase; (4) Process response model phase; (5) Extraction of a control rule phase; (6) Extraction of a workflow phase; (7) Variation analysis and improvement phase; and (8) Plant operation phase. Additionally, we have shown that the extracted workflow has the ability to detect outliers of process data.

The main contribution of the research is to establish a method of mining a set of meaningful control rules from the Learning Classifier System using the Minimum Description Length criteria with improvement rate and Tabu search method. The effectiveness of the proposed method has been demonstrated using actual plant data. We believe that the proposed method is one of the practical LCS applications.

References

1. Rissanen, J.: Modeling by shortest data description. Automatica 14, 465–471 (1978)
2. Adami, C.: Introduction to Artificial Life. Springer, NY (1998)
3. Mehta, M., Rissanen, J., Agrawal, R.: MDL-based decision tree pruning. In: Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD 1995), pp. 216–221 (1995)
4. Yamanishi, K.: A learning criterion for stochastic rules. Machine Learning 8, 165–203 (1992)
5. Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A.I.: Fast discovery of assosiation rules. In: Fayyad, U., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R. (eds.) Advances in Knowledge Discovery and Data Mining, pp. 307–328. AAAI Press and The MIT Press (1996)
6. Hilderman, R.J., Hamilton, H.J.: Knowledge Discovery and Measures of Interest. Kluwer Academic Publishers, Dordrecht (2001)
7. Holland, J.H., Reitman, J.S.: Cognitive systems based on adaptive algorithms. SIGART Bull. (63), 49 (1977)
8. Smith, S.: A learning system based on genetic adaptive algorithms. In: Ph.D thesis. University of Pittsburgh (1980)
9. Smith, S.: Flexible learning of problem solving heuristics through adaptive search. In: Proceedings 8th International Joint Conference on Artificial Intelligence (August 1983)
10. Butz, M.V., Pelikan, M., Llorà, X., Goldberg, D.E.: Extracted global structure makes local building block processing effective in XCS. In: GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation, pp. 655–662. ACM, New York (2005)
11. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Publishing Company, Inc., Reading (1989)
12. Adriaans, P., Zantinge, D.: Data Mining. Addison-Wesley, Reading (1996)
13. Weiss, S.M., Indurkhya, N.: Predictice Data Mining, A Practical Guide. Morgan Kaufmann Publishers, Inc., San Francisco (1997)
14. Berndt, D.J., Clifford, J.: Finding patterns in time series: a dynamic programming approach, 229–248 (1996)
15. Hetland, M.L., Saetrom, P.: Evolutionary rule mining in time series databases. Mach. Learn. 58(2-3), 107–125 (2005)
16. Harvey, A.C.: Time Series Models. Prentice Hall/Harvester (1993)
17. Stock, J.H., Watson, M.W.: A probability model of the coincident economic indicators. Working Paper 2772, National Bureau of Economic Research (November 1988)
18. Freitas, A.A.: Data mining and knowledge discovery with evolutionary algorithms. Springer, Heidelberg (2002)
19. Barry, A., Holme, J., Llora, X.: Data mining using learning classifier systems. In: Bull, L. (ed.) Applications of Learning Classifier Systems, pp. 15–67. Springer, Heidelberg (2004)
20. Iba, H., de Garis, H., Sato, T.: Genetic programming using a minimum description length principle. In: Kinnear Jr., K.E. (ed.) Advances in Genetic Programming, pp. 265–284. MIT Press, Cambridge (1994)
21. Bacardit, J., Garrell, J.M.: Bloat control and generalization pressure using the minimum description length principle for a pittsburgh approach learning classifier system. In: Kovacs, T., Llorà, X., Takadama, K., Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 2003. LNCS (LNAI), vol. 4399, pp. 59–79. Springer, Heidelberg (2007)

22. Smyth, P., Goodman, R.M.: An information theoretic approach to rule induction from databases. IEEE Transactions on Knowledge and Data Engineering 4(4), 301–316 (1992)
23. Hilderman, R.J., Hamilton, H.J.: Heuristic measures of interestingness. In: Proceedings of the Third European Conference on the Principles of Data Mining and Knowledge Discovery, pp. 232–241 (1999)
24. Quinlan, J.R.: C4.5:Programs for Machine Learning. Morgan Kaufman Publishers, Inc., San Francisco (1993)
25. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers, Dordrecht (1997)
26. Yamanishi, K., Takeuchi, J.: A unifying approach to detecting outliers and changepoints from nonstationary data. In: The Eighth ACM SIGKDD(KDD2002) (2002)

Analysing Learning Classifier Systems in Reactive and Non-reactive Robotic Tasks

Renan C. Moioli[1], Patricia A. Vargas[2], and Fernando J. Von Zuben[1]

[1] Laboratory of Bioinformatics and Bio-Inspired Computing - LBiC
School of Electrical and Computer Engineer - FEEC/Unicamp Campinas-SP, Brazil
[2] Centre for Computational Neuroscience and Robotics (CCNR)
Department of Informatics, University of Sussex,
Falmer, Brighton, BN1 9QH, United Kingdom
{moioli,vonzuben}@dca.fee.unicamp.br, p.vargas@sussex.ac.uk

Abstract. There are few contributions to robot autonomous navigation applying Learning Classifier Systems (LCS) to date. The primary objective of this work is to analyse the performance of the strength-based LCS and the accuracy-based LCS, named EXtended Learning Classifier System (XCS), when applied to two distinct robotic tasks. The first task is purely reactive, which means that the action to be performed can rely only on the current status of the sensors. The second one is non-reactive, which means that the robot might use some kind of memory to be able to deal with aliasing states. This work presents a rule evolution analysis, giving examples of evolved populations and their peculiarities for both systems. A review of LCS derivatives in robotics is provided together with a discussion of the main findings and an outline of future investigations.

1 Introduction

The growing interest in the study of intelligence, particularly in the context of *embodied cognitive science*, is responsible for the majority of sophisticated experiments employing mobile robots lately. The embodied cognitive science establishes as its main methodology the development of experiments in autonomous robot navigation and control. According to Pfeifer and Scheier [1], autonomous robotics is considered an appropriate paradigm for studying the principles of intelligence.

Nonetheless, a robotic control system that enables a robot to perform tasks in the way a human being, or even an insect, does, is far from being achieved. This is mainly because, first, there is a lack of high-quality information available for decision making processes, second, there are multiple conflicting objectives to be fulfilled, and third, usually the robot needs to tackle complex environment dynamics.

In an effort to overcome those problems, evolutionary and reinforcement learning techniques have been increasingly applied to robot autonomous navigation problems. One example of the application of such techniques is Evolutionary

J. Bacardit et al. (Eds.): IWLCS 2006/2007, LNAI 4998, pp. 286–305, 2008.

Robotics (ER). ER is a particularly novel field of research, which aims to apply evolutionary computation techniques to evolve the robot morphology (hardware design) and/or controllers for both real and simulated autonomous robots. In spite of being a well-established research area with many achievements reported in the literature [2], it has some intrinsic difficulties, mainly associated with the time spent while evaluating an individual. Thus, some researchers favour the use of reinforcement learning techniques, which seem to be more adequate to online learning [3].

Towards the aim of combining evolution and learning in an integrated way, Learning Classifier Systems (LCS), which were originally proposed by Holland [4], emerge as an alternative. Furthermore, these systems can be considered an appropriated framework for the synthesis of complex adaptive systems [5] and models of inference process in cognitive systems [6].

However, according to Holmes and collaborators [7], there are challenges still unexplored and limitations still to be alleviated in the field of LCS. Hence, variants of these systems are being investigated based on their adaptive features and also on problem peculiarities. Presently, the most investigated ones are: (i) accuracy-based LCS, XCS [8], and its variants, XCSF [9], XCSMH [10]; (ii) Anticipatory Classifier System, ACS [11]; (iii) Temporal Classifier System, TCS [12]. It is opportune to highlight that Learning Classifier Systems are being applied to diverse areas [13], from optimization problems [14] to time-series analysis [15]. Nonetheless, there are few contributions to robot autonomous navigation.

The primary objective of this work is to perform some performance analysis involving the strength-based LCS and the accuracy-based LCS, named EXtended Learning Classifier System (XCS). The rationale behind the choice of both systems, LCS and XCS, is threefold. Firstly, XCS is the most studied variant of the LCS. Secondly, XCS was proposed in an attempt to develop a new credit assignment procedure that could guide the classifier system to perform better in situations where the strength-based fitness is unsuitable. Thirdly, all classifier systems developed to date were inspired by the original LCS and thus it is believed that the comparison between the currently most studied (XCS) and the original one (LCS) could provide some insightful outcomes to the field of robotics and also to the LCS community.

Moreover, it is envisaged that this work is an opportunity to clearly understand the mechanisms of rule evolution and its peculiarities in the robotics domain for both systems, compiling the related work and consequently providing useful information for future research in the area. In addition, examples of evolved populations are presented, illustrating the rules dynamics and the resultant robot behaviour.

To embark on this endeavour two experiments are proposed. Experiment 1 presents a purely reactive task, while Experiment 2 has some aliasing states, thus the robot might need to use some sort of memory to solve the task, as will be further described in Section 5.

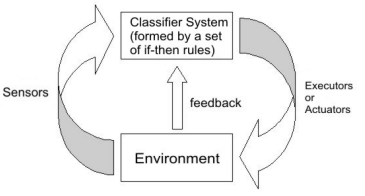


Fig. 1. Interaction of the classifier system with the environment

This paper is organized as follows. Section 2 introduces the fundamentals of the LCS and XCS. Section 3 presents a review of related works in the literature. Section 4 depicts the platforms used during the experiments. Section 5 describes the experiments and reveals the results together with an analysis of the evolution for each case. Finally, a discussion and future work are presented in Section 6.

2 Learning Classifier Systems Fundamentals

A LCS is composed of classifiers that represent the individuals of a population. Each classifier corresponds to if-then rules, which have two distinct parts: antecedent and consequent. The antecedent part defines which environmental condition should the classifier be associated with. In other words, for each input information (message) from the environment, there should be a classifier that matches itself with this input and thus propose an action expressed by its consequent part. Thereafter, depending on its *"strength"* (a sort of fitness based value) and similarity to the message from the environment, the classifier will be activated All activated classifiers compete in order to decide which one will act on the environment Depending on the result of this action, the classifier that originated this action can be penalized or rewarded (increasing or decreasing its strength). Periodically, the classifier set or population undergoes an evolutionary process, based on the strength of the classifiers, by means of a genetic algorithm [16]. Figure 1 illustrates the interaction between LCS and the environment.

Messages from the environment are coded as strings formed from a binary alphabet {0,1}. Each classifier is represented by a string formed from the alphabet {1,0, #}. The symbol "#" is used as a "don't care" symbol, i.e. it matches any other possible value in the message corresponding field. Another important feature is the classifier *specificity*. It is related to the proportion of "don't cares" in the message string: the more "#" symbols a classifier has, the larger the number of input messages it will be associated with. This measure may contribute to the reduction of the size of the population of classifiers at the end of the evolutionary process.

The aforementioned structure is a basis for all classifier systems here studied. On the next subsections, two types of classifier systems will be detailed, focusing on their peculiarities.

1: **BEGGINING OF AN EPOCH**
2: Receive message from the environment
3: Code message
4: Select classifiers that identify themselves with the message
5: Begin "Competition":
6: Calculate *bid* of every competitor
7: Point out winner
8: Charge competitors and winner
9: Act on the environment
10: Receive message from environment
11: Code message
12: Reward or punish winner classifier
13: Charge tax of life of every individual
14: If it is not the end of an "Epoch", return to **Step 2**
15: **END OF AN EPOCH**
16: Select most apt individuals from the population of classifiers
17: Apply crossover and mutation and generate descendants
18: Select weaker individuals in population
19: Insert descendants in population, replacing the weaker individuals
20: If it is not the end of the evolutionary process, go back to **Step 1**

Algorithm 1. LCS simplified algorithm (adapted from [17])

2.1 LCS

The classifier system originally proposed by Holland [4], named Learning Classifier System, is a methodology for creating and evolving rules, also known as classifiers. In a decision-making system, these classifiers code alternative actions designed to deal with the current state of the environment [18].

The implementation of the traditional LCS, was based on the work of [4], [17] and [19]. A compact version of the main procedures is presented in Algorithm 1. The system perceives the environment (Step 2). The message received allows the definition of the classifiers that most identify themselves with the message (Steps 3 and 4). Usually, the Euclidean distance or the Hamming distance are used in this process. Once defined the classifiers with higher degrees of matching, it should be decided which of them will act on the environment. This competition is done in the form of an auction (Equation 1), where each classifier presents a value (a bid) determined as a function of its strength and its specificity. The classifier that obtains the best bid will be chosen to act on the environment (Step 9). It is important to understand the role of the Gaussian noise function. This function is responsible for diversifying the choice of the winning classifier, allowing classifiers with low strength to be sporadically selected.

After acting on the environment, the system updates the input information, i.e. the message from the environment. Once coded (Step 11), this message will be used to establish the reward or punishment of this classifier (Step 12), increasing or decreasing the classifiers' strength. In the end, all classifiers are taxed for being on population (Step 13). This taxing is essential for replacing low-activated

Table 1. LCS and XCS Parameters Description

LCS Parameters		XCS Parameters	
n	: half-life of classifier	N	: Maximum population size
Energy(t)	: classifier energy at instant t	β	: learning rate$(0 < \beta < 1)$
T_{Winner}	: result of equation 1 if winner classifier; 0 otherwise	α, ϵ_0, ν	: precision control constants $(0 < \alpha < 1, \epsilon_0 > 0, \nu > 0)$
Bid	: result of equation 1	θ_{mna}	: minimum number of actions in $[M]$
T_{Bid}	: tax for participating in competition	γ	: discount factor in multistep problems
R_t	: reward at instant t	θ_{GA}	: GA threshold
k_0	: constant referred to the classifier energy $(0 < k_0 < 1)$	χ	: crossover probability
k_1	: constant referred to the non-specific part of the classifier $(0 < k_1 < 1)$	μ	: mutation probability
k_2	: constant referred to the specific part of the classifier $(0 < k_2 < 1)$	θ_{del}	: deletion threshold
Spec	: specificity of a classifier	δ	: fraction of the mean fitness of the population
SPow	: influence of specificity of a classifier	θ_{sub}	: subsumption threshold
S_t	: Classifier energy at instant t	$P_{\#}$	: *don't care* probability
σ	: standard deviation of the noise	p_I, ϵ_I, F_I	: initialization parameters
N_t	: noise with normal distribution		

classifiers by other new ones. Equation 2 describes the process of updating the strength of the classifier.

Steps 16 to 19 correspond to the evolutionary part of the process, when more active and adapted classifiers will have higher chances of being combined to produce new classifiers. These new classifiers will replace low-activated classifiers or classifiers with unsatisfactory performance. The parameters of the algorithm are further explained in Table 1, which also includes parameters of the XCS to be described in the following section.

$$Bid = k_0 * (k_1 + k_2 * Spec^{SPow}) * S_t + \sigma * N_t, \tag{1}$$

$$Energy(t+1) = (1 - \frac{1}{2}^{\frac{1}{n}}) * Energy(t) + R_t - T_{Winner} - T_{Bid} * Bid, \tag{2}$$

2.2 XCS

Wilson [8] proposed adaptations to the original classifier system, introducing the XCS. This approach differs from the original LCS mainly in two aspects. The first aspect is that the energy of each classifier is formed by prediction, prediction error and fitness. The fitness of each classifier is a number calculated by an inverse function of the prediction's mean error, that is, the fitness is based on the measure of how precise is a prediction. The second different aspect is that the genetic algorithm is applied in niches defined by the set of activated classifiers in each interaction, and not randomly anymore.

At each execution of the algorithm, a set M of classifiers that identify themselves with the actual environment is created. The initial population in XCS is null, and the system uses a technique called *covering*. This technique creates

classifiers on demand, instead of generating them randomly at the beginning. This approach prevents the proposition of classifiers with antecedent parts very different from those that are found in the environment, hence evolution and population gets simpler. When a new message from the environment is received, if no classifier matches the message, a new classifier is then created, with the antecedent equal to the environment reading. However *don't cares* may be introduced, with probability $P_{\#}$.

After performing covering, a vector containing the expected payment value prediction for each action in the set M is formed based on fitness and on prediction error of each classifier. The action set is then determined based on the values of this vector. Then, it acts over environment. The way that the action selection is done can vary, and different proposals were studied in literature [20]. In this work the technique known as 50/50 exploration/exploitation is adopted. Basically, the first 50 interactions have the action randomly chosen, amongst those present in M, role played by the Gaussian noise in the original LCS system.

The fitness updating of each classifier (see Equation 8) is made based on the reward ρ, the prediction p, the relative prediction error of the payment received (ϵ), taking into account the error of the other classifiers in the action set and the maximum value previously calculated in the prediction vector (*max(PA)*) (Equations 3 to 8). The parameters of each classifier are referred using the dot notation ($cl.\kappa$ refers to the precision κ of each classifier), and their description is given in Table 1.

$$p \leftarrow p + \beta(P - p), \tag{3}$$

$$\epsilon \leftarrow \epsilon + \beta(|P - p| - \epsilon), \tag{4}$$

$$P \leftarrow \rho_{-1} + \gamma * max(PA), \tag{5}$$

$$\kappa = \begin{cases} 1 & if \quad \epsilon < \epsilon_0 \\ \alpha(\epsilon_0/\epsilon)^{\nu} & otherwise \end{cases} \tag{6}$$

$$\kappa^{'} = \frac{\kappa}{\sum_{cl \in [A]} cl.\kappa}, \tag{7}$$

$$F = F + \beta(\kappa^{'} - F), \tag{8}$$

The genetic algorithm is applied in the action set only. It starts when the classifiers in the action set exceed a value θ_{GA}, which refers to the mean time since the last participation of each classifier in the GA. For being responsible for the discovery of new classifiers, it seems reasonable to apply the GA only in exploratory search trials, and not in tuning trials (exploitation). In each GA session, the new classifiers created are compared with the whole population. If the new classifier is already present in population, its numerosity is increased.

Numerosity stands for the number of microclassifiers (numerosity equal to one) that this macroclassifier represents. This approach allows the population to be reduced and also to improve the processing speed [8].

One can notice that the basic structure of the LCS is preserved. However, the classifier selection system and credit assignment have been changed. In this sense, the classifiers are not punished or rewarded by their immediate action, but by their performance in fulfilling the task. The fitness is no more determined by the strength, but by the accuracy by which a prediction is made. This allows the XCS to obtain good solutions in cases the LCS can not [8].

3 Related Work

This section is devoted to describe the most relevant works using LCS and its derivatives in autonomous robot navigation.

Cliff and Ross [21] added memory mechanisms to the Zeroth-Level Classifier System (ZCS)[22] and employed it in a series of woods tasks. Good results were obtained, although optimal performance was rarely achieved. They found that the system has stability difficulties when dealing with long action chains. The authors suggested that this is mainly due to greedy classifier creation and conflicting overgeneral classifiers. It is mentioned that XCS and its accuracy-based fitness could be used to alleviate this problems. They also made an attempt on a close approximation to real robots. Again, long action chains appeared to be a determinant factor in reducing performance.

Stolzmann [11] proposed an alternative framework, called Anticipatory Classifier System (ACS), where each classifier has an anticipatory prediction of the consequence of its action on an environment. This new architecture allows the system to create an internal model of the environment in execution time. It was applied in robot learning tasks and it was demonstrated that the robotic agent could perform latent learning. Further work [23] combined action planning and latent learning into two robotic tasks, achieving near optimal performance in few trials.

Dorigo and Colombetti [24] applied the LCS, exploring the behaviour-based approach, to control a robot in simulated and real environments. The use of classifier systems was motivated by the fact that the LCS presents three main characteristics: parallelism, allowing a faster computation while adding flexibility to the design of the learning system; distributed architecture, allowing the division of the task into subtasks, simpler to be solved; and behaviour evolution, responsible for continuous adaptation in dynamic environments.

Katagami and Yamada [25] described an adaptive learning classifier system, based on XCS, that could speed up learning in a mobile robot, and applied it in a wall-following task and in a soccer game. It consisted of a human-robot interaction system, where the human agent was able to interact with the rules and directly teach the robot, improving the initial population set-up and adding new rules at any time.

Bonarini [26] adapted the LCS to incorporate fuzzy rule-based models, presenting an introductory investigation on coupling the fuzzy approach to the LCS

paradigm. In Bonarini [27], the author describes *ELF*, a fuzzy logic controller able to evolve a population of fuzzy rules. It is a quite complex system, with many similarities to some LCS algorithms, such as: the existence of a *strength* concept (related to the performance of a rule), influenced by a reinforcement program; the partition of rules into sub-populations, according to their matching with current environment situation; and competition to perform an action. The author applies the system to a set of experiments using a simulated animat and a real robot, producing good results. Recent work [28] presented FIXCS (Fuzzy Implementation of XCS), an extension to previous fuzzy-based LCS, inspired by XCS. The authors report a longer learning time, balanced by the increase of robustness.

Gerard and Sigaud [29] presented the YACS (Yet Another Classifier System), a system based on ACS, with the difference that in YACS the emphasis is on classifiers which anticipate well rather in classifiers which propose an optimal action. The system also combines dynamic programming algorithms and latent learning. The authors report good results in maze tasks but stressing the need for further investigations on generalization.

Hurst et al. [12] applied the ZCS classifier system [22] and introduced the TCS classifier system to control a real robot in a simple obstacle avoidance task, and also in a phototaxis task. Studley and Bull [3] coupled the TCS and the XCS approaches to create the X-TCS classifier system, which presented good results in a phototaxis task. Hurst and Bull [30] explores the use of constructivism-inspired mechanisms within a Neural Learning Classifier System (NCS). In the NCS, each rule is represented by an artificial neural network, and parameters are under self-adaptation. Simulated mazes were adopted to analyse performance, and experiments with a real robot were also performed. The TCS approach was combined with the NCS as an attempt to overcome the difficulties in dealing with real robots, such as time scale and ambiguity. The authors reported reasonable performance and showed the viability of learning classifier systems in real environment problems.

Webb et al. [31] implemented a XCS system to control a simulated Khepera in a non-Markov environment, using an internal memory mechanism described in [10]. Cazangi et al. [19] used a LCS to control a simulated and real robot in a task where the robot should learn to capture some targets while avoiding obstacle collisions. The results presented are very promising, evidencing the robustness and adaptability of the system.

Vargas et al. [32] presented a hibrid system, named Clarinet (Classifier Immune Network), which consists of an immuno-genetic network of classifiers. The novel hybrid approach demonstrated good performance and robustness in a simulated robotic autonomous navigation problem.

Based on agent-independent and agent-dependent characteristics of maze environments, Zatuchna [33] describes an extensive analysis of these environments by introducing new metrics for measuring complexity. Inspired by psychological principles, the author also proposes a new learning approach, named Associative Perception Learning (AgentP). This new model employs memory mechanisms and has a state-action-state rule structure similar to ACS. It also has distance

based reward distribution, which reflects the minimal number of steps to the goal state from the current state of the agent. In addition, the model does not attempt to learn generalizations of states. It is shown that this new model is able to successfully solve the majority of the maze problems proposed, including mazes with aliasing states. There are a number of other works employing ACS on robotics. For a further reference the reader should refer to [34].

Considering all the works published in the literature, since the first appearance of the LCS paradigm, one can notice that there are relatively few contributions to the field of robotics. One of the objectives of this work is to encourage the use of classifier systems in autonomous robot navigation by revisiting some experiments and also providing insights into rule evolution.

4 Description of the Platforms

This section is devoted to the description of the robot used and the simulator. The robot is a Khepera II mini-robot. It has a diameter of 70mm and is 30mm high, weighting around 80g. A robot with this reduced size allows the implementation of experiments in a small-size platform and with a low consumption of energy. The robot gets its energy by wire or by its internal batteries, which have an autonomy of 1 hour, approximately; it is sustained by two wheels, responsible for its motion. The wheels have independent electric motors and, by applying different speeds in the wheel, direction adjustments are obtained. The maximum speed of the robot is 1 m/s, and the minimum is 0.08 m/s.

The robot has in its basic structure 8 infrared sensors that incorporate emitters and receptors. The sensors measure the environment luminosity and the obstacle distance. The range of the sensors, related to obstacles, is 10cm, maximum. The time of data acquisition of each sensor is 2.5ms and, at each 20ms, a complete measure is done. The output of each measure is an analog value converted to a 10-bit number.

Some external factors, like the presence of incandescent lamps, can cause interference in the measurements of the distance sensors. This is due to the fact that the same sensor is used for both tasks (distance and light detection). For example, the distance sensor emits an infrared ray and calculates the distance to the obstacle based on the time this ray took to left the emitter and return to the receptor. However, if there is an incandescent lamp nearby, its luminosity will interfere the received rays, changing the sensory reading. As the light sensor uses the infrared range, we must apply lamps that are on the infrared zone. Therefore, it is important to take care when developing an experiment. This fact appears to have been noticed by the manufacturer, and the new Khepera series, Khepera III, have distance sensors based on ultrasound, being immune to light interference [35].

The simulations were carried out using a robot simulator developed by Perreta and Gallagher [36]. It consists of a program that reproduces in a very reasonable way the sensory behaviour of the real robot, which facilitates the migration from the virtual environment to the real environment. This simulator has open code and free license.

5 Experiments

This section presents experiments encompassing two distinct robot tasks. Both consists of consolidated experiment proposals, previously described in literature.

LCS and XCS were first conceived for binary coding. This was the codification adopted by the antecedent and the consequent of the classifiers here. Although the discretization leads to sensitivity loss, it was observed in [12] and [31] that it is possible to implement simulated and real experiments using this technique.

Therefore, for the antecedent coding, it was considered one left lateral sensor, one frontal sensor and one right lateral sensor, leading to a string of size 3. Each field is updated in the following way: if the sensory reading is superior to 50 (ranging from 0 to 1023), the field takes the value 1, and value 0 otherwise. Likewise, the antecedent related to the luminosity sensor was coded considering the value 1 if the reading was lower than 100 (ranging from 0 to 500) and 0 otherwise. The "#" symbol is also used. The robot actions were coded using 2 bits. The values "00" and "11" means straight ahead movement, "01" stands for turning 90° right, and "10" indicates turning 90° left. Examples of possible situations that the robot can face and its actions are depicted Table 2.

To avoid an action overload and iterations without an effective environment change, it was adopted a strategy suggested in Hurst et al. [12]. This strategy consists of letting the robot perform the same action until the environment has changed. As a result, it promotes a decrease in the number of irrelevant messages to the system and facilitates a faster evolution.

The simulation parameters were chosen based on previous studies in the literature: for the LCS [4], [16] and [17], and for the XCS [31], [20] and [37].

5.1 Experiment 1

The first experiment is carried out on a variation of the *woods* environment. This is one of the most used environments for robot learning tasks [8,21,38,39]. It consists of a bidimensional environment formed by cells, where each cell can contain an obstacle T (tree), an objective F (food), secondary objectives as energy level (E), and blank spaces (empty). The number and role of the secondary objectives are in charge of the designer, depending on the complexity and the behaviour to be examined. The robot is capable of detecting the eight neighbor cells. Usually, the robot task in this scenario is to learn the shortest path to the objective, avoiding collisions, and eventually accomplishing secondary objectives, such as recharge. A number of trials are performed, positioning the robot

Table 2. Possible Situations

Distance Antecedent	Light Antecedent	Interpretation
000	000	open space, no light
001	100	obstacle at right, light at left
011	010	obstacle at right front, light at front
010	000	obstacle at front, no light
100	001	obstacle at left, light at right
101	000	obstacle at right and left, no light

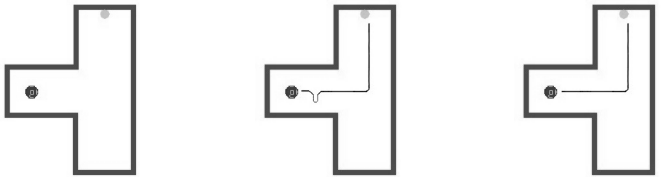


Fig. 2. Results for Simulated Experiments, including the arena (a), the LCS trajectory (b), and the XCS trajectory (c)

in an empty cell. If the robot reaches its objective, or if a number of iterations is achieved (thus avoiding a large looping), a new trial is initiated. It is important to highlight that this problem takes into consideration the obstacle avoidance problem.

In the experiment developed here, the environment has no discretization, or cells, thus the robot is in a continuous action space. See Figure 2(a). The light represents the objective to be reached, and the robot must learn to reach it in a minimum number of iterations.

LCS Results. The LCS was able to successfully solve the problem. The reward function used in this experiment was: 2 points, if the robot moves forward without collision; 10 points for every activated classifier, if it reaches the target; and 0 otherwise. The evolutionary process takes place either at each 40 iterations, or at each target arrival, or if a collision happens. According to Equation 1, the rule with highest bid is selected to act.

Table 3 illustrates the classifiers with more energy in the evolved population, and Table 4 presents the simulation parameters. Notice that the sequence of movements of the robot corresponds exactly to high energy classifiers. An interesting aspect to be emphasized is the difference between the energy of the classifiers. As the system does not have a long-term reward distribution mechanism, like the one present in XCS, multiple behaviour sequences guide the robot to reach the target, and not necessarily the shortest sequence is evolved. Thus, some classifiers activate more than others. For instance, notice the first classifier on Table 3, which points out light straight ahead. Every time this classifier activates, the robot arrives at the target and a larger reward is given.

The system converged after 12 generations, creating 45 classifiers and undergoing 3 collisions (mean values). Figure 2(b) illustrates the trajectory.

Table 3. LCS Results for Experiment 1

Distance Antecedent	Light Antecedent	Consequent	Energy
0##	010	11	245.87
01#	000	10	109.98
001	000	11	81.29
100	000	11	64.64

Table 4. Parameters for Experiment 1

LCS		XCS	
Parameter	Value	Parameter	Value
k_0	0.1	θ_{sub}	20
k_1	0.1	θ_{GA}	25
k_2	0.01	θ_{del}	25
SPow	3	θ_{mna}	3
σ	0.01	δ	0.1
n	200	N	400
Energy(0)	10	β	0.2
T_{Bid}	0.003	α	0.1
Mutation rate	0.05	μ	0.05
Crossover rate	0.8	χ	0.8
Probability of *dont't care*	0.1	$P_{\#}$	0.1
		ϵ_0	10

Table 5. XCS Results for Experiment 1

Distance Antecedent	Light Antecedent	Action	Prediction	Prediction Error	Fitness	Numerosiy	Experience
0#0	000	10	265.00	6.50	1.00	2	95
#0#	#00	00	410.00	239.81	0.62	5	272
000	010	11	1000.00	0.00	1.000	24	81
#00	#00	00	442.00	278.07	0.62	1	168

XCS Results. The XCS was able to successfully solve the problem. When reaching the light, a reward of 1000 was given, and 0 otherwise. A trial consisted of a maximum of 40 iterations, either if the robot reached the target, or if a collision happened. The first 10 iterations of a trial was done by exploration, and the next 10 iterations by exploitation. The genetic algorithm was applied only in exploration time, or at the end of a trial.

As its fitness is based on accuracy, and not directly on reward, like occurs in the LCS, the classifiers that have fitness near 1 are not necessarily the classifiers that predict the largest reward. Notice that the reward is distributed over population (fourth column in Table 5). This is intrinsic to the fitness updating algorithm of the XCS, in which the last classifier to act before the objective is accomplished has a higher reward prediction than the former classifiers, that guided the system to the final state. As the choice of which classifier to act depends on the prediction reward vector (PA), the system evolves to the minimum path to the target, avoiding situations as those present in the LCS, where it is not guaranteed that the system will always evolve to the shortest path. More complex environments guides here to the same problem observed in LCS, as described in [31]. It is important to notice, also, that the most adapted classifiers have its numerosity greater than the others.

Table 5 shows some of the evolved classifiers, and Table 4 presents the simulation parameters. The system converged in 17 trials, with 38 classifiers created, with 9 collisions occurring (mean values). Figure 2(c) illustrates the trajectory.

Real World Experiment. In order to validate the LCS and XCS performance in real world applications, the controllers already described were transfered from

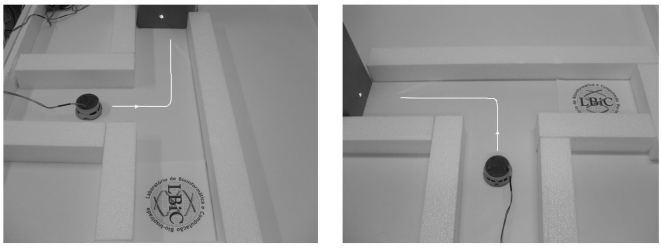


Fig. 3. Results for Real Experiments

simulation to a real environment. This is an important stage of the work once real robot implementations are considered fundamental for the success of a control technique.

As mentioned in section 4, the real robot has some differences compared to the simulated robot. The distance and light sensors readings have to be adapted, as well as the encoders that stands for wheel movement. Experiments are developed in a dark environment due to light interference problems. The light source had little spreading, aiming at not altering the sensors readings in every part of the scenario. Figure 3 shows the real arena and the trajectories of the robot when controlled by each system, LCS and XCS. The robot presented good performance for the real experiment when to compared to the simulated experiment.

5.2 Experiment 2

In autonomous robot navigation there is a request for developing controllers that are able to express non-reactive behaviour [40,41]. A non-reactive behaviour is defined if the robot does not rely only on its sensory readings while accomplishing a task, which is opposite to what happens when the behaviour is purely reactive. It was verified in [31] that environments with various ambiguity states can not be solved without the use of internal memory.

The experiment reported in this section is interesting because it shows the memory capacity of the agent, one of the requirements to express more complex behaviours. The experiment proposed is described in [40]. The task is illustrated in Figure 4. Jakobi [40] evolved artificial neural networks as controllers, and here the controllers will be implemented based on LCS and XCS.

It consists of a T-Maze environment. The robot begins to navigate at the first corridor, passes by a light source, and then must decide for a side to turn at the end of this first corridor. If it turns to the same side of the light source and gets to the end of the second corridor, the maximum reward is given. That is, the control system must remember, in some way, the side of the light source. This behaviour involves the capability of moving without collisions in the environment as well as deciding for one of the sides to turn at the end of the corridor. The robot must also implement a procedure to remember which side of the first corridor the light source was in order to perform the correct action at the end of the first corridor.

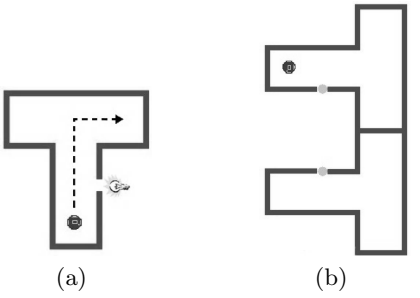


Fig. 4. Simulation Environment for Experiment 2. Adapted from [40].

In this experiment the robot interacts with the environment in a different way from the previous experiment. As the task now is much more complex, the causality of the whole system is crucial. By causality we mean that every time the robot is at the same position in the scenario, taking the same action different times would lead to the same external conditions (when one considers a static scenario, which is the case here).

Because of continuous action and due to the inherent noise and robot/environment inaccuracies the robot had to deal with a multi-objective task (obstacle avoidance and goal finding) together with this causality problem. When the task is relatively simple, the system can cope with that, but when dealing with more complex tasks, having this kind of uncertainties can lead to a malfunction of the system and also to a misinterpretation of the results. So the environment was divided into cells. When acting, the robot can go forward or turn 90°, to the right or to the left. This mechanism is specially useful in the XCS system, because of its difficulty in dealing with long sequences of actions.

LCS Results. The classifier system implemented does not use the long-term reward transmission called *bucket brigade*, described in [4]. Alternatively, the final bonus is equally distributed among the classifiers that were activated from the beginning to the end of the trial. Riolo [42] investigated this mechanism and concluded that it is able to perform similar to the *bucket brigade*.

The classifier antecedent structure is similar to the one described in Experiment 1. The reward scheme and parameters were the same.

The first results were not adequate. The system was unable to adapt to different positions of the light source. The controllers converged to a desired behaviour only for one of the options of light source position, that is, sometimes the robot always turned left, sometimes it always turned right. This can be explained by the fact that there is an ambiguity at the end of the corridor. After passing the first initial corridor and achieving the end of it, in front of the wall, the robot can not remember in which side the light was located, the message from the environment is simply *"wall straight ahead, no light"*. It became necessary to use some kind of internal memory.

Some works using LCS with internal memory were studied [31,38,42], and it was decided to use an extra bit in the antecedent, corresponding to the internal

Table 6. LCS Results for Experiment 2

Lamp at Left					
Distance Antecedent	Light Antecedent	Internal Memory	Consequent	Memory Consequent	Energy
1##	0##	0	11	0	974.67
101	100	0	11	1	1092.74
101	000	1	11	1	1093.43
0#0	0#0	1	10	0	1094.12
001	000	0	11	1	1094.81
Lamp at Right					
Distance Antecedent	Light Antecedent	Internal Memory	Consequent	Memory Consequent	Energy
1##	0##	0	11	0	974.67
1##	0##	0	11	0	974.67
1##	0##	0	11	0	974.67
010	000	0	01	0	998.61
100	000	1	11	0	999.30

memory of the robot, and an extra action, also internal, that can modify this bit value. Thus, the classifier is chosen based on the message from the environment and on the robot internal state. The robot can act over the environment and also change its internal memory.

With these modifications, the system was able to find a solution. Table 6 illustrates the sequence of classifiers evolved in the problem. Notice that the internal memory is widely used. The classifiers in Table 6 are presented in order of operation along navigation, allowing to find out the way the system solved the problem and represented the solution.

This experiment showed that the original LCS with an internal memory bit is capable of dealing with tasks with long-term rewards, which is not so common in the literature.

XCS Results. To cope with aliasing states, the XCS system was improved with an internal memory mechanism, described in [10]. It consists of an extra bit in the antecedent, which matches with an internal memory register, and a internal action bit, which is capable of modifying the memory register status. The internal action selection is always by exploitation. A trial consisted of a maximum of 20 iterations, either if the robot reached the target, or if a collision happened. At the beginning of each trial, the system randomly chooses if it will be an exploration or an exploitation trial. The genetic algorithm was applied only in exploration time, or at the end of a trial.

The XCS was able to solve the task successfully, though some considerations have to be pointed out. Regularly, XCS receives the maximum reward (1000) when reaching the goal, 0 otherwise. Following these assumptions, which are adopted in almost every XCS work, the system was very unstable in the sense that good solutions were hard to find. This can be explained by the long sequence of actions to be performed in this problem, associated with the use of memory. The robot must get to the goal to receive the reward, and then this reward is distributed to the population by means of rule interaction. The robot achieves the goal by the first time only by chance, there is no reward until that, so action-selection via exploitation is useless. When the robot finally reaches the

Table 7. XCS Results for Experiment 2

Lamp at Left									
Distance Antecedent	Light Antecedent	Internal Memory	Action	Internal Action	Prediction	Prediction Error	Fitness	Numerosiy	Experience
101	000	0	00	1	251.00	91.90	0.37	15.00	2151.00
101	1#0	1	00	0	180.00	4.18	1.00	17.00	524.00
101	000	0	00	1	230.00	94.16	0.32	15.00	2152.00
#10	0#0	1	10	0	506.00	4.67	1.00	20.00	465.00
0##	000	0	11	1	723.00	4.16	1.00	6.00	818.00
001	##0	1	11	0	996.00	4.00	1.00	11.00	485.00
Lamp at Right									
Distance Antecedent	Light Antecedent	Internal Memory	Action	Internal Action	Prediction	Prediction Error	Fitness	Numerosiy	Experience
101	000	0	00	1	259.00	105.18	0.38	15.00	2153.00
1#1	001	1	11	1	282.00	4.25	1.00	38.00	430.00
101	0#0	1	00	0	375.00	4.97	1.00	8.00	462.00
#1#	000	0	01	1	507.00	4.91	1.00	12.00	407.00
1#0	##0	1	11	0	721.00	4.56	1.00	5.00	293.00
100	#00	0	00	1	996.000	4,00	1.00	19.00	644.00

final state, it receives the reward, but only the last rule to activate receive this reward, all the other rules in the population have their reward prediction equal to zero. Only when the system gets closer to the goal again, the reward can be passed to the rule before the last rule is activated. This carries on until the reward map to that task is completely formed. Just to illustrate the complexity of this task, consider the work developed by Lanzi [38], who evolved an animat using similar modifications to the XCS as the ones made in this work. Both the environment and the actions to be performed were discrete. The task was not the same as the one studied here, but in essence they are similar. In that task, the optimum average performance of the animat was 2.9 steps to reach the goal and the average random walk to the goal was approximately 110 steps. In this work, the minimum number of steps to the goal is 6, therefore the random walk may become much longer.

The solution adopted was to reward the robot in 1000 when it gets the goal, and in 20 when it moves forward, as moving forward would be the most present action in an evolved system (the robot only needs to turn at the end of the first corridor), helping to reduce the exploration time. With this modification, the robot could finish the task efficiently, and the system now operates as a temporal difference learner. Table 7 gives an example of some rules of a successfully evolved population. The first point to stress is the use of the memory register. It is possible to see that the side of the light is determinant for the switching of internal state. The fitness is high and prediction error is low for almost every rule, except for the rules that are activated in more than one scenario location: the number of steps to the goal is different, however the sensory input is the same, leading to different reward predictions, which is responsible for the prediction error. Also, although the objective of turning correctly at the end of the corridor was accomplished, a compact population of maximum general rules was not obtained. This could be due to short simulation time, as more general rules did not have sufficient time to obtain high accuracy and fitness values.

6 Discussion and Future Work

The objective of this work was to establish an analysis of the performance of the strength-based LCS and the accuracy-based XCS when applied to two distinct robotic tasks. The first task is purely reactive, the second one is non-reactive. This work presented rule evolution analysis, giving examples of evolved populations and its peculiarities for each of the two types of LCS studied. Future works will include the same analysis of other LCS and XCS derivatives; in particular the ACS model which has a number of related works reported in the literature.

Both systems were capable of solving the tasks successfully. However, the strength-based LCS presented solutions in which the shortest sequence was not always present. Moreover, rules presented over-specialization and overgeneralization. This happens mainly because the main rules are reinforced by the most frequent situations. Therefore, these rules continue to act in contexts in which they are not adequate. In the robotics domain, these features are deleterious to the performance, the speed of evolution, and the robustness of the system. Nonetheless, as the reward comes immediately after an action, finding a good solution sometimes is faster using LCS than using XCS. Moreover, the LCS system, even presenting some limitations when compared to more recent proposals, can lead to insightful ideas for future work, as it is the original framework from which other classifier systems have been developed.

The autonomous controller for the mobile robot should deal with dynamic and noisy environments. The XCS system, by constructive characteristics, propagates rewards through the rules more recently used. To observe the evolution to a given behaviour, a set of rules must be exposed to the same situation for many times. Aiming at alleviating the computational burden, discretization of environment and actions may be adopted. Webb et al. [31] uses a discretization at the sensory level. This causes a reduction in the search space, helping the evolution of the system. However, this discretization or any other method that incorporates a priori knowledge reduces the level of autonomy of the system, for this knowledge should have been produced only by the robot and its learning system. An alternative may be the work of Studley and Bull [3], which incorporates the time to the reward function, creating the X-TCS, applying it in continuous environments, with continuous actions.

Another important topic associated with the experiments is the design of the reward function. The natural procedure would be to define different rewards for each behaviour. This attitude is criticised by Kovacs [37], which demonstrates that the reward function can compromise the generalization capability of the system. Thus, an immediate question is: how to define a reward for each action? And furthermore, how to define punishments for wrong actions in critic behaviours (for instance, avoiding collision against an obstacle)? The alternative proposed by Kovacs [37] is to change the frequency at which each of these situations should be encountered. In this sense, more relevant situations are encountered more frequently. Another option would be to separate the set of rules into subsets, leaving sets that represent more relevant situations with a larger number of rules. However, defining what is important or not is very challenging,

for the emergence of a complex behaviour can arise from a sequence of simple processes that would be suppressed by these approaches. Unfortunately, this problem has not been well explored in the literature, as yet.

Equally relevant is the analysis of the action selection methods. Some works in the literature have attributed high relevance to the way action selection is defined, supported by significant variation in the final performance of the system [3,37,43]. The most analyzed techniques are roulette-wheel, tournament and a mixture of exploration and exploitation. Once more, the literature lacks of contributions in this area when applied to autonomous robot navigation.

The XCS is currently the most studied and used LCS [3,43]. It has evolution and learning mechanisms that prevent overgeneral rules with high fitness. Another point that supports the use of XCS is its ability to distribute rewards in sequential tasks and not considering its immediate performance, as occurs in traditional LCS. This is implemented mainly by the credit assignment mechanism inspired by Q-Learning. The consequence is that XCS is capable to map, in a more balanced way, the action space of the problem. This aspect tends to promote a better performance.

As a final remark, it is pertinent to once more highlight that the primary objective of this work is not to compare the performance of two different learning classifier systems in robot autonomous navigation tasks, but to analyze their performance. This was fulfilled by presenting drawbacks and positive aspects of each one on each task, stressing the mechanisms of rule evolution (providing examples of evolved populations) and its numerous peculiarities. As a result of this thorough qualitative analysis, it is envisaged that the understanding improved of the general role that classifier systems may play in cognitive tasks, and more specifically in autonomous navigation of mobile robots.

Acknowledgments

Moioli and Von Zuben would like to thank CAPES and CNPq for their financial support. Patricia A. Vargas was particularly supported by the Spatially Embedded Complex Systems Engineering (SECSE) project, EPSRC grant no EP/C51632X/1. The authors also would like to thank Renato Cazangi for his valuable comments.

References

1. Pfeifer, R., Scheier, C.: Understanding Intelligence. MIT Press, Cambridge (1999)
2. Nolfi, S., Floreano, D.: Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines. Bradford Book (2004)
3. Studley, M., Bull, L.: X-TCS: Accuracy-based learning classifier system robotics. In: Congress on Evolutionary Computation, pp. 2099–2106. IEEE, Los Alamitos (2005)
4. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press (1975)

5. Holland, J.H.: Hidden Order. Addison-Wesley, Reading (1995)
6. Holland, J.H., Holyoak, K.J., Nisbett, R.E., Thagard, P.: Induction: Processes of Inference, Learning, and Discovery. MIT Press, Cambridge (1986)
7. Holmes, J.H., Lanzi, P.L., Stolzmann, W., Wilson, S.: Learning classifier systems: new models, successful applications (2000)
8. Wilson, S.W.: Classifier fitness based on accuracy. Evolutionary Computation 3, 149–175 (1995)
9. Wilson, S.W.: Classifiers that approximate functions. Natural Computing: an international journal 1(2-3), 211–234 (2002)
10. Lanzi, P.L., Wilson, S.W.: Toward optimal classifier system performance in non-markov environments. Evolutionary Computation 8(4), 393–418 (2000)
11. Stolzmann, W.: Learning classifier systems using the cognitive mechanism of anticipatory behavioural control. In: Proceedings of the First European Workshop on Cognitive Modelling, pp. 82–89 (1996)
12. Hurst, J., Bull, L., Melhuish, C.: TCS learning classifier system controller on a real robot. In: Proceedings of the 7th International Conference on Parallel Problem Solving from Nature (2002)
13. Bull, L.: Applications of Learning Classifier Systems. Springer, Heidelberg (2004)
14. Vargas, P.A., Lyra Filho, C., von Zuben, F.J.: Application of learning classifier systems to the on line reconfiguration of electric power distribution networks. Applications of Learning Classifier Systems 150, 260–275 (2004)
15. Armano, G.: NXCS experts for financial time series forecasting. In: Bull, L. (ed.) Applications of Learning Classifier Systems (2004)
16. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison Wesley, Inc., Reading (1989)
17. Vargas, P.A.: Classifier systems for loss reduction in electric power distribution networks (in portuguese). Master's thesis, School of Electrical and Computer Engineering, Unicamp, Brazil (2000)
18. Booker, L.B., Goldberg, D.E., Holland, J.H.: Classifier systems and genetic algorithms. Artificial Intelligence 40, 235–282 (1989)
19. Cazangi, R.R., Von Zuben, F.J., Figueiredo, M.: A classifier system in real applications for robot navigation. In: The IEEE Congress on Evolutionary Computation, Canberra, Australia, vol. 1, pp. 574–580 (2003)
20. Butz, M.V., Wilson, S.W.: An algorithmic description of XCS. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 2000. LNCS (LNAI), vol. 1996, pp. 253–272. Springer, Heidelberg (2001)
21. Cliff, D., Ross, S.: Adding temporary memory to ZCS. Adaptive Behavior (1995)
22. Wilson, S.W.: ZCS: A zeroth level classifier system. Evolutionary Computation 2(1), 1–18 (1994)
23. Stolzmann, W., Butz, M.V.: Latent learning and action planning in robots with anticipatory classifier systems. Learning Classifier Systems (1999)
24. Dorigo, M., Colombetti, M.: Robot Shaping: An Experiment in Behavior Engineering. MIT Press, Cambridge (1997)
25. Katagami, D., Yamada, S.: Interactive classifier system for real robot learning. In: Proceedings of the 2000 IEEE International Workshop on Robot and Human Interactive Communnication, Osaka, Japan (2000)
26. Bonarini, A.: An introduction to learning fuzzy classifier systems. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 1999. LNCS (LNAI), vol. 1813, pp. 83–106. Springer, Heidelberg (2000)
27. Bonarini, A.: Fuzzy modelling: Paradigms and practice. In: Pedrycz, W. (ed.) Fuzzy Modelling: Paradigms and Practice. Kluwer Academic Press, Norwell (1996)

28. Bonarini, A., Matteucc, M.: Fixcs: A fuzzy implementation of XCS. In: IEEE International Fuzzy Systems Conference, FUZZ-IEEE 2007 (2007)
29. Gerard, P., Sigaud, O.: YACS: Combining dynamic programming with generalization in classifier systems. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 2000. LNCS (LNAI), vol. 1996, pp. 259–266. Springer, Heidelberg (2001)
30. Hurst, J., Bull, L.: A neural learning classifier system with self-adaptive constructivism for mobile robot control. Artif. Life 12(3), 353–380 (2006)
31. Webb, E., Hart, E., Ross, P., Lawson, A.: Controlling a simulated khepera with an XCS classifier system with memory. In: Banzhaf, W., Ziegler, J., Christaller, T., Dittrich, P., Kim, J.T. (eds.) ECAL 2003. LNCS (LNAI), vol. 2801, pp. 885–892. Springer, Heidelberg (2003)
32. Vargas, P.A., de Castro, L.N., Michelan, R., Von Zuben, F.J.: An immune learning classifier network for autonomous navigation. In: Timmis, J., Bentley, P.J., Hart, E. (eds.) ICARIS 2003. LNCS, vol. 2787, pp. 69–80. Springer, Heidelberg (2003)
33. Zatuchna, Z.V.: AgentP: a Learning Classifier System with Associative Perception in Maze Environments. PhD thesis, School of Computing Sciences, University of East Anglia (2005)
34. Kovacs, T.: A learning classifier systems bibliography (2002), `http://www.cs.bris.ac.uk/~kovacs/lcs/search.html`
35. S. A. KTEAM (2007), `http://www.k-team.com`
36. Perreta, S.J., Gallagher, J.C.: The Java Khepera simulator from the wright state university, Ohio, USA (2004)
37. Kovacs, T.: Strength or Accuracy: Credit Assignment in Learning Classifier Systems. Springer, Heidelberg (2004)
38. Lanzi, P.L.: An analysis of the memory mechanism of XCSM. In: Proceedings of the Third Annual Conference on Genetic Programming (1998)
39. Lanzi, P.L., Wilson, S.W.: Optimal classifier system performance in non-markov environments. Technical report, Politecnico de Milano (1999)
40. Jakobi, N.: Minimal Simulations for Evolutionary Robotics. PhD thesis, University of Sussex (1998)
41. Husbands, P.: Evolving robot behaviours with diffusing gas networks. In: Evolutionary Robotics: First European Workshop, EvoRobot 1998 (1998)
42. Riolo, R.L.: The emergence of default hierarchies in learning classifier systems. In: Proceedings of the Third Congress on Genetic Algorithms (1989)
43. Butz, M.V., Goldberg, D.E., Tharakunnel, K.: Analysis and improvement of fitness exploitation in XCS: Bounding models, tournament selection, and bilateral accuracy. Evolutionary Computation 11(3), 239–277 (2003)

Author Index