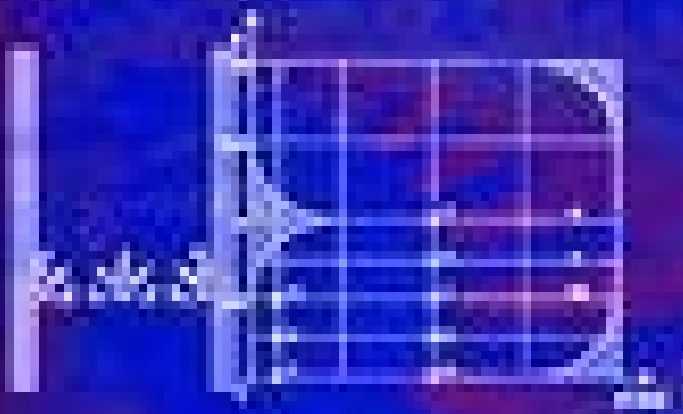
Lecture Notes in Artificial Intelligence

Pier Luca Lanzi
Wolfgang Stolzmann
Stewart W. Wilson (Eds.)

Learning Classifier Systems

From Foundations to Applications



Springer

Lecture Notes in Artificial Intelligence 1813

Subseries of Lecture Notes in Computer Science
Edited by J. G. Carbonell and J. Siekmann

Lecture Notes in Computer Science
Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Singapore
Tokyo

Pier Luca Lanzi Wolfgang Stolzmann
Stewart W. Wilson (Eds.)

Learning Classifier Systems

From Foundations to Applications

Springer

Series Editors

Jaime G. Carbonell, Carnegie Mellon University, Pittsburgh, PA, USA
Jörg Siekmann, University of Saarland, Saabrücken, Germany

Volume Editors

Pier Luca Lanzi
Politecnico di Milano, Dipartimento di Elettronica ed Informatzione
Piazza Leonardo da Vinci n. 32, 20133 Milano, Italy
E-mail: lanzi@morgana.elet.polimi.it

Wolfgang Stolzmann
Universität Würzburg, Institut für Psychologie III
Röntgenring 11, 97070 Würzburg, Germany
E-mail: stolzmann@psychologie.uni-wuerzburg.de

Stewart W. Wilson
Prediction Dynamics
Concord, MA 01742, USA
and
University of Illinois at Urbana-Champaign
Department of General Engineering
E-mail: wilson@prediction-dynamics.com

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Learning classifier systems : from foundations to applications /
Pier Luca Lanzi . . . (ed.). - Berlin ; Heidelberg ; New York ;
Barcelona ; Hong Kong ; London ; Milan ; Paris ; Singapore ;
Tokyo : Springer, 2000
(Lecture notes in computer science ; Vol. 1813 : Lecture notes
in artificial intelligence)
ISBN 3-540-67729-1

CR Subject Classification (1998): I.2, F.4.1, F.1.1

ISBN 3-540-67729-1 Springer-Verlag Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by PTP-Berlin, Stefan Sossna
Printed on acid-free paper SPIN: 10720288 06/3142 5 4 3 2 1 0

Preface

Learning classifier systems are a machine learning paradigm introduced by John Holland in 1976. They are rule-based systems in which learning is viewed as a process of ongoing adaptation to a partially unknown environment through genetic algorithms and temporal difference learning.

From the beginning, classifier systems have attracted the interest of researchers in many different areas, ranging from the design of gas pipelines to personal internet agents, and including cognitive science, data mining, economic trading agents, and autonomous robotics. In 1989 Stewart Wilson and David Goldberg presented a review of the first decade of classifier system research, discussing some of the milestones that characterized the field's early development. In 1992 the First International Workshop on Learning Classifier Systems (IWLCS92) was held in Houston, Texas.

The 1990s saw an increasing interest in the field: many successful applications to real-world problems were presented as well as new classifier system models. With seven years since the first workshop and more than 400 papers published, it was time in 1999 to examine again the state of the art of learning classifier system research. For this purpose the Second International Workshop on Learning Classifier Systems (IWLCS99) was organized and the idea of this volume conceived. The workshop, held in Orlando, Florida, July 13, 1999, attracted a vital community of researchers from many different areas who share a common interest in this machine learning paradigm. Some of the most interesting work presented at the workshop appears in this volume.

Our book provides an overview of the current state of the art of learning classifier systems and highlights some of the most promising research directions. The first paper of Part I asks the fundamental question: "What Is a Learning Classifier System?". Answers are given by John Holland, originator of classifier systems, and by other long-time researchers in the field: Lashon Booker, Marco Colombetti, Marco Dorigo, Stephanie Forrest, David Goldberg, Rick Riolo, Robert E. Smith, and Stewart Wilson. Three following papers summarize, from different perspectives, developments in the field since Wilson and Goldberg's 1989 review. Part II contains papers on advanced topics of current interest, including alternative representations, methods for evaluating rule utility, and extensions to existing classifier system models. Part III is dedicated to promising applications of classifier systems such as: data mining, medical data analysis, economic trading agents, aircraft maneuvering, and autonomous robotics. A classifier systems bibliography with more than 400 references completes the book.

We hope that this volume will be a key reference for researchers working with learning classifier systems over the next decade; the constantly increasing interest in this paradigm suggests that there will be many of them!

While we are already looking forward to editing the *sequel* of this book in 2010, we hope to see you in Paris on the 16^{th} September 2000, for the Third International Workshop on Learning Classifier Systems (IWLCS 2000).

March 2000

Pier Luca Lanzi
Wolfgang Stolzmann
Stewart W. Wilson

Organization

The idea of this book started during the organization of the Second International Workshop on Learning Classifier Systems (IWLCS99) which was held July 13, 1999 in Orlando (FL) during the Genetic and Evolutionary Computation Conference (GECCO99). Some of the best papers presented during that workshop are in this book.

Organizing Committee

Pier Luca Lanzi	Politecnico di Milano, Italy
Wolfgang Stolzmann	Universität Würzburg, Germany
Stewart W. Wilson	The University of Illinois at Urbana-Champaign, USA Prediction Dynamics, USA

Program Committee

Andrea Bonarini	Politecnico di Milano, Italy
Lashon B. Booker	The MITRE Corporation, USA
Marco Colombetti	Politecnico di Milano, Italy
Marco Dorigo	Université Libre de Bruxelles, Belgium
John H. Holmes	University of Pennsylvania, USA
Tim Kovacs	University of Birmingham, UK
Pier Luca Lanzi	Politecnico di Milano, Italy
Rick L. Riolo	University of Michigan, USA
Robert E. Smith	The University of The West of England, UK
Wolfgang Stolzmann	Universität Würzburg, Germany
Stewart W. Wilson	The University of Illinois at Urbana-Champaign, USA Prediction Dynamics, USA

Table of Contents

III Applications

IV The Bibliography

What Is a Learning Classifier System?

John H. Holland[1], Lashon B. Booker[2], Marco Colombetti[3], Marco Dorigo[4], David E. Goldberg[5], Stephanie Forrest[6], Rick L. Riolo[7], Robert E. Smith[8], Pier Luca Lanzi[3], Wolfgang Stolzmann[9], and Stewart W. Wilson[5,10]

[1] University of Michigan, USA
[2] The MITRE Corporation, USA
[3] Politecnico di Milano, Italy
[4] IRIDIA, Université Libre de Bruxelles, Belgium
[5] University of Illinois at Urbana-Champaign, USA
[6] Santa Fe Institute, USA
[7] Center for Study of Complex Systems
University of Michigan, USA
[8] The Intelligent Computer Systems Centre
The University of The West of England, UK
[9] University of Wuerzburg, Germany
[10] Prediction Dynamics, USA

Abstract. We asked "What is a Learning Classifier System" to some of the best-known researchers in the field. These are their answers.

1 John H. Holland

Classifier systems are intended as a framework that uses genetic algorithms to study learning in condition/action, rule-based systems. They simplify the "broadcast language" introduced in [26] by (i) eliminating the ability of rules to generate other rules, and (ii) by simplifying the specification of conditions and actions. They were introduced in [27] and were later revised to the current "standard" form in [28]. [31] gives a comprehensive description of this "standard" form, with examples. There are, however, many significant variants (e.g., Booker [9, this volume], Riolo [59], Smith [67, this volume], Wilson [83]).

In defining classifier systems I adopted the common view that the state of the environment is conveyed to the system via a set of detectors (e.g., rods and cones in a retina). The outputs of the detectors are treated as standardized packets of information, messages. Messages are used for internal processing as well, and some messages, by directing the system's effectors (e.g., its muscles), determine the system's actions upon its environment. There is a further environmental interaction that is critical to the learning process: the environment must, in certain situations, provide the system with some measure of its performance. Here, as earlier [26], I will use the term payoff as the general term for this measure.

P.L. Lanzi, W. Stolzmann, and S.W. Wilson (Eds.): LCS '99, LNAI 1813, pp. 3–32, 2000.

1.1 Basic Questions

Classifier systems address three basic problems in machine learning:

Parallelism and coordination. A system is unlikely to have enough single, monolithic rules to handle situations like "a red Saab by the side of the road with a flat tire", but such a situation is easily handled by simultaneously activating rules for the building blocks of the situation: "car", "roadside", "flat tire", and the like. In short, a rule-based system can handle a broad range of novel situations if it can act on them with combinations of "building block" rules. Then combinatorics works for the system instead of against it. Moreover, because appropriate building blocks appear frequently, in a wide range of situations, they are tested and confirmed at a high rate. The problem is to provide for the interaction and coordination of a large number of rules that are active simultaneously.

Credit assignment. Deciding which rules in a rule-based system are responsible for its successes, particularly when long sequences of "stage-setting" actions precede success, is an interesting and difficult problem. If the system is not continually monitored by a referee, solution of this problem is a sine qua non for learning. The problem is to credit an early action, which may look poor (as in the sacrifice of a piece in chess), for making possible a later positive action (as in setting the stage for the capture of a major piece).

In realistic situations, exhaustive exploration of all possible action paths (as in dynamic programming and Q-learning) is not feasible. So the credit assignment scheme must be "local". When many rules are active simultaneously the situation is exacerbated. Only a few of the early-acting rules may set the stage, while other rules active at the same time may be ineffective or, even, obstructive. Samuel's early work [65], using prediction over extended action sequences, points the way, but few have exploited his insights.

Rule discovery. Rule discovery (or its equivalent) is the most recondite problem in machine learning. It was not even well-handled in Samuel's remarkable work. We know that rules receiving little credit ("low strength") should be replaced, but random generation of new rules can only work for the smallest problems. The key to effectiveness is use of past experience to generate plausible hypotheses (rules) about situations as yet poorly understood – hypotheses not obviously contradicted by past experience.

Behind these three basic problems is a deeper question: How does a system improve its performance in a perpetually novel environment where overt ratings of performance are only rarely available? Obviously such improvement is not possible unless the environment has repeating (sub-)patterns. A learning task of this kind is more easily described if we think of the system as playing a game of strategy, like checkers or chess. After a long sequence of actions (moves), the system receives some notification of a "win" or a "loss" and, perhaps, some indication of the size of the win or loss. There is little information about specific

changes that would improve performance. Most learning situations for animals, including humans, have this characteristic – an extended sequence of actions is followed by some general indication of the level of performance.

1.2 How Classifier Systems Address These Questions

In classifier systems, parallelism and coordination are addressed by restricting rule action to the emission of messages. Tags in the messages allow flexible "addressing", providing coordination of activities. Detectors translate the current state of the environment into messages, and effectors translate selected messages into actions on that environment. The overall classifier system, then, can be viewed as a message processing system acting on the current list (set) of messages. Because messages only serve to activate rules, questions of "rule consistency" and "consistency maintenance" are avoided. More messages simply mean more active rules, and vice-versa.

Credit assignment is handled by setting up a market situation. In that market each rule acts as a go-between (broker, middleman) in chains leading from the current situation to (possible) favorable outcomes. At each instant, rules with satisfied conditions bid for the right to become active. If a rule becomes active, it pays its bid to the active predecessor(s) that sent messages satisfying its conditions (its "suppliers"). As an active rule, it then stands to profit from bids of subsequent bidders (its "consumers"). Rules active at the time of external payoff (reward, reinforcement) are apportioned the payoff as if it were income from "ultimate consumers".

Credit is accumulated by the rule as a strength (a kind of capital). Many of the variants of classifier systems offer alternative schemes for apportioning or accumulating credit. In general, whatever the apportionment scheme, stronger rules bid more, thereby being more likely to win the bidding process. That, in turn, makes those rules more likely to influence the system's behavior.

Rule discovery exploits the genetic algorithm's ability to discover and recombine building blocks. Classifier systems have "building blocks" at two levels: the parts (schemata) from which the condition and action parts of individual rules are constructed, and the rules themselves, as components of the overall system. The genetic algorithm works on this "ecology" at both levels. Because it is designed to work on populations (sets of rules), it is well suited to the task. Indeed classifier systems were designed with just this objective in mind.

Rule strength enters at both levels, being treated as a fitness (likelihood of being the parent of new rules) by the genetic algorithm. Variants offer different ways of translating strength into fitness, contingent on the particular variant used for credit assignment.

Each of the mechanisms used by the classifier system has been designed to enable the system to continue to adapt to its environment, while using its extant capabilities to respond instant-by-instant to that environment. In so doing the system is constantly trying to balance exploration (acquisition of new information and capabilities) with exploitation (the efficient use of information and capabilities already available).

1.3 Implementation

The computational basis for classifier systems is provided by a set of condition-action rules, called classifiers. A typical (single condition) rule has the form:

```
IF there is (a message from the detectors indicating)
            an object left of center in the field of vision,
THEN (by issuing a message to the effectors)
     cause the eyes to look left.
```

More generally, the condition part of the rule "looks for" certain kinds of messages; when the rule's conditions are satisfied, the action part specifies a message to be sent. Messages both pass information from the environment and provide communication between classifiers. Because many rules can be active simultaneously, many messages may be present at any given instant. From a computational point of view, it is convenient to think of the messages as collected in a list.

A computer-based definition of these rules requires a proper language for representing classifiers. It is convenient to think of the messages as bit strings (though representation via any alphabet or set of functions is equally possible). We can, in addition, think of each bit position as representing the value of a predicate (1 for true, 0 for false) though, again, this is only a convenience and not necessary for more complex representations. Classifier conditions then define equivalence classes (hyperplanes) over the message space using the *don't care* symbol '#'. More recently, it has proved useful to introduce a symmetry that allows messages to act over equivalence classes of conditions, using the 'fits all' symbol '?'. Other variants have been studied (e.g., [86, this volume], [1]).

If messages are bit strings of length m over the alphabet 1,0, then conditions are strings of length m over the alphabet 1,0,# and actions are strings of length m over the alphabet 1,0,?. The genetic algorithm mates these strings to produce new competing rules.

1.4 Future Research

In recent years there has been a focus on classifier systems as performance systems or evolutionary incarnations of reinforcement systems (e.g., Lanzi [47], Wilson [87, this volume]). This has been fruitful but, I think, falls short of the potential of classifier systems. Smith [67, this volume] makes the point that classifier systems, in their ability to innovate, offer broader vistas than reinforcement learning.

In my opinion the single most important area for investigation vis-a-vis classifier systems is the formation of default hierarchies, in both time ("bridging classifiers" that stay active over extended periods of time) and space (hierarchies of defaults and exceptions). Having been unable to convince either colleagues or students of the importance of such an investigation, I intend to pursue it myself next summer (year 2000).

2 Lashon B. Booker

Speculations about future directions for classifier system research can benefit from a look at Holland's original research goals in formulating the classifier system framework [27,31]. The early motivations for classifier system research were derived from the many difficulties that a learning system encounters in a complex environment. Realistic environments are both rich and continually varying. They force a learning system to deal with perpetually novel streams of information, and often impose continual requirements for action given sparse payoff or reinforcement. There are many examples of natural systems and processes that handle such challenges effectively: the central nervous system, co-adapted sets of genes, the immune system, economies and ecologies. Most artificial systems, on the other hand, tend to be brittle in the sense that substantial human intervention is often required in order to realign system responses to address changes in the environment. Holland identified a list of criteria for avoiding brittle behavior. These criteria summarize key aspects of what appears to lie behind the flexibility exhibited by natural systems: they implement processes that build and refine models of the environment. Moreover, these models typically involve massive numbers of elements adapting to the environment and to each other through many local nonlinear interactions. Classifier systems were proposed as an example of a rule-based system capable of learning and using multi-element models of this kind.

While initial research on classifier systems focused on many of the now-familiar computational mechanisms and algorithms, it is important to keep in mind that "The essence of classifier systems is a parallelism and standardization that permit both a 'building block' approach to the processing of information and the use of competition to resolve conflicts." [31, p. 611]. The "standard" computational procedures are significant because they illustrate how it is possible, in principle, to design and build a system that meets the criteria for avoiding brittleness. However, in my view, it is the design principles that provide the insights about what a classifier system "is". For that reason, I find it useful to think of the classifier system framework in the broadest sense as a general-purpose approach to learning and representation. This approach provides a way of looking at problems and structuring flexible systems to solve them, but it does not necessarily prescribe the detailed methods that are best suited for solving a particular problem class. Research in classifier systems typically focuses on computational methods, but too often advances in methodology are not leveraged to further our understanding of the broader issues that the classifier system framework was designed to address. I believe that the most important goal for classifier system research is to develop deeper insights about how our computational methods bring us closer to realizing the capabilities Holland envisioned in his original concept.

One way to achieve this goal is to develop more formal characterizations of classifier system methods and processes. There has been relatively little work along these lines since Holland's [30] early attempt to characterize what a regularity is in a perpetually novel environment, and what it means for a classifier

system to exploit these regularities. An agenda of theoretical issues that need to be pursued was proposed more than a decade ago [10] and it remains relevant today. Research like this is needed so that classifier system design decisions can be grounded in clear, preferably formal statements of the technical issues being addressed. We also need need to be able to characterize the capabilities and limitations of classifier systems more carefully, including some expectations regarding the quality of the solutions the system is likely to learn.

Another way to broaden our understanding of classifier system design principles is to study them from a wider variety of perspectives. The classifier system research to date has focused almost exclusively on the learning and representation issues that are prominent in reinforcement learning problems. Recently there have been systems and methods proposed that emphasize other perspectives on learning and representation. These perspectives have been derived from studies of the immune system [64], agent-based economic models [18], and ecologies [9]. New computational methods will emerge from this research, and similar excursions should be encouraged. Hopefully, this will lead to new insights about how to build systems that confirm Holland's original intuitions about the potential of the classifier system framework.

3 Marco Colombetti and Marco Dorigo

During most of the nineties, we have been using learning classifier systems (LCSs) to develop simulated animats and real robots able to learn their behavior. The types of behavior we focussed on were quite zoomorphic (exploring an environment, searching for and chasing a prey, fleeing from a predator, and so on). However, we were more interested in investigating ways of using reinforcement to *shape* the behavior of an agent, rather than in studying how a system can spontaneously adapt to a natural environment. We viewed our work mostly as a contribution to *behavior engineering*, as we call the discipline concerned with the development of flexible autonomous robots. Most of the results of this work were finally collected in a book [17].

Throughout our research, we regarded LCSs as a promising model for reinforcement learning (RL). For this reason, we have been surprised to see that LCSs are often contrasted with RL, as if they were different approaches. In our opinion, it is important to distinguish between RL as a class of problems, on one side, and the learning techniques typically studied by the RL community, on the other side. As is well known, the fundamental problem of RL is defined in the following setting (Fig. 1). An *agent* interacts with an *environment* through *sensors* and *effectors*. At times, the agent receives from the environment a scalar signal, called *reinforcement* (r). The problem is to exploit the agent's experience (i.e., the history of its interactions with the environment) to develop a behavior policy that maximizes some functional of reinforcement over time.

RL problems can be studied from two different viewpoints. They can be regarded as abstract problems, and dealt with in a purely mathematical way. The main point here is to define learning algorithms of low computational complexity

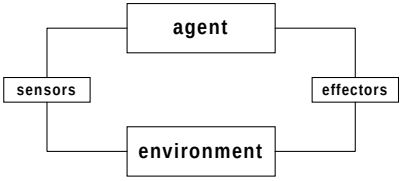


Fig. 1. The reinforcement learning setting.

that are guaranteed to converge to an optimal behavior policy. Alternatively, an RL researcher may take a more concrete standpoint, and focus on the agent as a software system. The main questions now become, H ow can we represent behavioral policies in a compact way? How can we efficiently implement RL algorithms on such representations? Should we trade policy optimality for higher computational efficiency, and how? Can we develop RL systems that scale up to problems of realistic size? These are the questions we had in mind when we started to apply RL techniques to robot shaping.

Since the beginning of our work, we felt that LCSs were fit for solving the RL problems we wanted to tackle. The features we found most appealing were the overall architecture of LCSs, and the use of a genetic algorithm (GA) in the discovery component. As far as architecture is concerned, we believe that the production rule paradigm, of which LCSs are an example, is particularly suitable for implementing adaptive agents. This paradigm allows for a very peculiar interaction of simple behavioral rules, in such a way that both cooperation and competition among several rules are exploited to generate opportunistic behavior. Production rules have been very successful within the classical approach of symbolic artificial intelligence, and we think that the reasons of such a success are still valid in a more low-level approach.

The main reason of our interest in LCSs, however, was due to its evolutionary component. As we have argued elsewhere [13], a major aspect of behavior engineering is the attempt to go beyond the limits of "rational design." By this we mean that the behavior of a real robot cannot be completely predefined. Learning, quite obviously, is a way of overcoming ignorance. However, ignorance is a problem not only for the learning agent, but also for the robot's designer. For an agent interacting with a physical environment, learning is going to be not only a matter of optimally tuning a set of parameters, but also of discovering structure. An optimal coupling of an agent with its environment is only possible if the dynamical structure of the environment is somehow mirrored by the structure of the agent's representation of its own behavior policy.

The evolutionary component of an LCS has the remarkable ability to produce a compact representation of a relevant subspace of all possible behavior rules. In principle, this gives LCSs the power to scale up to problems of realistic size. Unfortunately, in our work we found that the learning power of LCSs is still too weak to deal with the kinds of problems involved in concrete applications of

autonomous robots. We tackled this difficulty in the following way. First, we gave our learning agents as much input information as possible, by adopting a step-by-step reinforcement schedule. Second, we endowed our agents with some "innate" structure, by predesigning a hierarchical architecture for behavior control. To do so, we exploited ALECSYS [16,15], a transputer-based system that allows for the implementation of a network of interconnected LCSs. As a result, a substantial part of the final agent behavior is actually suggested by the designer, through the development of a suitable architecture and through a skillful definition of the reinforcement function. Notwithstanding this, we believe that the results are still of great interest for behavior engineering. In particular, the use of an RL approach allows the robot designer to concentrate on a high-level specification of the target behavior (through the definition of the reinforcement function), thus avoiding the risk of being caught up in a bundle of details.

Since we completed our work on robot shaping, much water has flown under the bridge of LCSs. The introduction of the XCS model by Wilson [83] appears to us to be a major achievement, for both theoretical and practical reasons. From a theoretical point of view, the main virtues of XCS are that it is a very neat model, and that it stresses the belonging of LCSs to the field of RL. We hope that, in the future, the RL community will be more inclined to regard LCSs as an interesting approach to the solution of RL problems. From the practical point of view, XCS has the advantage, at least to our eyes, that it gets rid of some aspects of the LCS model that, although very interesting in principle, have not proved to work effectively in practical applications. In particular, we want to mention the mechanism by which an LCS is expected to form rule chains that may be considered as simple behavior plans. In our experiments, we never happened to observe such a phenomenon. The current approach based on the use of memory registers [46,52] seems to us to be a more promising way of overcoming the limitations of simple reactive behavior.

4 Stephanie Forrest

Classifier systems were originally proposed as a model of inductive processes in human-like cognitive systems [25]. The basic classifier system architecture incorporates ideas from artificial intelligence, machine learning, cognitive psychology, economics, evolution, and computer design into one framework. The system has the following components: parallel forward-chaining rules, a reinforcement learning algorithm (the bucket brigade), a set of "genetic operators" for evolving the rule set, a simple interface with the external environment, and a theory of model construction based on homomorphic maps.

Classifier systems represented an advance in our thinking about cognitive modeling in several ways. Representations in classifier systems were (and are still) fine-grained—the execution of a single rule represented a much smaller reasoning step that those commonly found in symbolic production systems [56], and information was encoded as simple bit strings, rather than complex data structures, as in semantic networks or conceptual dependency theory. One hypothesis about

classifier systems was that symbolic-level reasoning would arise spontaneously from fortuitous interactions and combinations of individual rules. Such symbols would be discovered and manipulated naturally by the system, fluidly and implicitly, without the need for explicitly named locations in memory such as those found in the symbolic reasoning systems mentioned earlier. Many rules could be active simultaneously, thus allowing the system to consider multiple, and even conflicting, hypotheses up until the time that an output decision was made. But because the message list had finite size, there was also the possibility of competition among rules, allowing the system to focus. By contrast, most other reasoning systems of the day required that the system be maintained in a logically consistent state at all times. Classifier systems incorporated two important forms of learning—the bucket brigade to assign credit (reward) among among classifiers, and various discovery operators for generating variants and recombinations of existing successful rules. These learning mechanisms, combined with the 1,0,# vocabulary of rules, allow the system to discover and represent knowledge in terms of equivalence classes. This is important in environments with large numbers of states, where methods such as Q-learning [79] are problematic because of the huge number of possible state/action pairs. Finally, and in my view, most importantly, classifier systems were constructed as the realization of a formal theory about how intelligent systems construct internal models of their environment and use those models to enhance their existence. This theory, based on homomorphic maps and an extension, known as *quasi-morphisms*, remains as one of the central contributions of classifier systems.

With a few notable exceptions, for example [60], the past twenty years of research on classifier systems has focused on engineering the architecture so that it exhibits plausible computational behavior and can solve interesting real-world problems. Although great progress has been made on this front, there has been relatively little emphasis on the cognitive modeling issues that classifier systems were intended to address. Of these, perhaps the most central outstanding question is what symbols are and how they are discovered, maintained, and used effectively. It is commonly believed that higher animals, such as humans, rely heavily on the use of symbols to represent and manipulate models of the environment, for language, and other abstract cognitive tasks. However, we still have little understanding of how symbols are stored and processed in the brain. These and other questions related to classifier systems as a model of cognitive activity are an important direction for future research.

Ideas about situated intelligence, such as those described in [23,12], have changed our views about the nature of intelligent artifacts. Natural systems exhibiting intelligent behavior are now understood as having co-evolved with their environments, rather than as isolated designs. The interactions between these intelligences and their environments are central to their success, even in the case of brains where the majority of neural activity is internal to the system (e.g., when we are asleep and dreaming). Brains are structurally organized around important sources of sensory input, such as the visual system, and are continuously exposed to environmental signals, even during embryonic development, where

spontaneous retinal activity allows topographic mapping of the retina onto the developing visual cortex. Situated intelligent artifacts are perhaps more complex to think about, because they cannot be neatly separated from their environments, but they can in some cases use their environments in ways that simplify their computations.

Thus, a second important area of future research is to rethink our assumptions about classifier design from the perspective of situated intelligence. Classifier systems were originally intended as a generic architecture for cognitive systems, one that could be placed in a wide variety of different environments, and this has biased our ideas about how to deploy classifier systems. For example, most classifier systems have narrow-bandwidth interactions with their environments, and it might be appropriate to think about designing them from a more perceptual perspective, creating systems that are "awash" in environmental messages, as for example, in ref. [64] where a system resembling a classifier system is situated in a live local area network, exposed to a constant flow of TCP/IP packets.

One missing component in classifier systems is the ability to aggregate sets of rules into encapsulated components, analogous to subroutines in conventional programming languages. Classifier systems are "flat" in the sense that all rules have the same status, and groupings of rules into subroutines (corresponding to subassemblies in Hebbian models) are intended to occur automatically, without an explicit reinforcing mechanism. Although it may be technically possible to design rule sets that have this property, through the use of tags and bridging classifiers, it is highly unlikely that robust logically isolated components will be discovered and sustained through the learning operations of the classifier system. Rather than trying to improve our learning algorithms or to devise cleverer representations, I believe that an explicit mechanism is needed, analogous to Koza's automatic function definition for genetic programming [43]. Such a mechanism would recognize effective clusters of classifiers, aggregate them into a single unit such that the individual rules were not directly accessible from outside the cluster, define a limited interface to the rest of the system, and protect them from the ongoing pressures of mutation and crossover.

The original insights which inspired the design of classifier systems remain compelling, and they address important and unresolved issues in our understanding of intelligent systems. Few would argue that the exact mechanisms employed by classifier systems are those used in human brains, but the classifier system serves as an illustration of a set of design principles that are central in the design, and our understanding of the design, of many intelligent systems, including the brain. I believe that it is important for researchers to focus more on the basic principles exhibited by classifier systems and less on the specific implementation that was introduced nearly twenty years ago.

As an example, Steven Hofmeyr recently developed a model immune system which resembles the spirit of classifier systems, but implements few of the architectural details in the same way [24]. Detectors in this system represent generic immune cells, combining properties of T-cells, B-cells, and antibodies, and correspond to the condition parts of classifiers. However, instead of using a match

rule based on the $1, 0, \sharp$ alphabet traditionally used in classifier systems, Hofmeyr adopted one called *r-contiguous bits* [58] based only on bit strings. To extend the immune detector into a full-fledged condition/action classifier rule, a few bits can be concatenated to each detector to specify a response (analogous to different antibody *isotypes*). Instead of associating a strength with each rule, each detector in the model immune system has a life cycle, consisting of immature, naive, activated, memory, and death states. These states reflect how successful the detector is and how long it will live. The mechanisms that control how detectors move through their life cycle correspond to the role of the bucket brigade (credit assignment) in classifier systems. However, learning in the model immune system is simpler than classifier systems in the sense that credit is assigned directly from the environment to the detectors, and strength is not passed among immune cells. Internal feedbacks and self-regulation are modeled through a primitive form of a cytokine system (signalling molecules secreted by immune cells, which diffuse locally and either stimulate or downregulate other immune cells). In place of the message list, the system is intended to live in a computer network environment, constantly exposed to new signals. Bidding for messages in classifier systems is analogous to immune cells competing to bind to foreign datapaths. Although the mapping between this model immune system and classifier systems is not $1-1$, it captures many of the important properties of classifier systems and provides an example of how the spirit of classifier systems might be realized in new ways.

5 David Goldberg

Some Reflections on Learning Classifier Systems. I appreciate the editors' invitation to contribute to this important volume marking what must be called a *renaissance of learning classifier systems* (LCSs). Although I have kept my finger in the LCS pie through occasional papers on LCS subjects, the main body of my work shifted following my 1983 dissertation applying genetic algorithms (GAs) and LCSs to gas pipeline control; in recent years, I have largely focused my efforts to develop (1) an effective methodology for the design of GAs, (2) an integrated design theory for selectorecombinative GA design, (3) competent genetic algorithms—GAs that solve hard problems, quickly, reliably, and accurately, and (4) efficiency enhancement technologies (EETs) for faster solutions.

In this short essay, I'd like to give some personal reflections on why I shifted my work away from learning classifier systems, what I learned from those efforts, and why I am renewing a more active program in LCS research. I start by reflecting on my involvement in classifier systems back in the eighties.

Some Ancient LCS History. When I first proposed my dissertation topic in 1980, I was under the impression that these things called classifier systems were well understood and that my dissertation would be a relatively simple exercise in application. After all, as I sat through John Holland's lectures at the

University of Michigan, it all sounded so plausible: roving bands of rules fighting, mating, competing, dying, giving birth, and so forth. The first indication that there might be some work to do was when I went to look for some computer code to build on and found that there wasn't any available to me. The second indication of difficulty came when I actually built a classifer system on my little Apple II computer in Pascal, turned it on, and realized that productive system performance was completely drowned out by the action of myriad parasitic rules.

Thereafter, I realized that I couldn't shotgun my way to success. I knew that I needed to engineer the system well using bounding analyses, careful experiments, and intuition, and I proceeded to construct what I believe was the first classifier system to exhibit a default hierarchy. True, by today's standards, the accomplishment was modest, but I did succeed in creating a system with rules and messages, apportionment-of-credit, and genetic learning. The university granted me my PhD, and I moved on to the University of Alabama, wondering what I should do next.

A Quest for Competent GAs and 3 Lessons. Some pointed questioning in my dissertation defense helped me realize that EVERY aspect of classifier system design at the time was built on intellectual quicksand. Elsewhere, I have called classifier systems a *glorious quagmire*, and I believe this description at that time was apt. Genetic algorithms had bounding schema theory, but competent GAs had not yet been designed. The bucket brigade had the right idea, but the theorems of reinforcement learning were not yet available to us, and no one knew what rule syntax was necessary to solve the problems at hand. This situation seemed untenable to me. After all, in physical system design, engineers are used to having the laws of physics to guide system design. Here, the design environment was shifty, to say the least.

My immediate reaction to that environment was to look at the heart of genetics-based machine learning (GBML), the genetic algorithm, and see if I could tame GA design somewhat. My reasoning was that with effective GAs in hand, I would be able to return one day to classifier system design without the core rule induction method as a big question mark. This is not the forum for a detailed discussion of these explorations in competent GAs; various papers and a forthcoming monograph will better serve this function for the reader. Instead, I would like to briefly discuss three fairly high-level lessons of this journey, lessons that I believe carry over to the problem of competent LCS design.

Specifically, my explorations in competent GA design have taught me the importance of three things:

1. a proper methodology of invention and design,
2. the right kind of design theory, and
3. the right kind of test functions.

I briefly discuss each somewhat further.

Elsewhere I have borrowed from my colleague Gary Bradshaw and used the Wright brothers and their method of invention and design to help designers of complex *conceptual machines* such as genetic algorithms and classifier systems

to understand how to build conceptual machines that work. Too often in the domain of the conceptual, mathematical rigor is a siren song that wrecks the design voyage on the shoals of proof. It is important in designing any complex system to (1) decompose the design problem, (2) use effective, economic design theory to organize experimentation and guide design efforts, and (3) integrate the pieces of the design with dimensional reasoning. This method of invention is commonplace in the design of *material machines* such as airplanes, automobiles, and toasters, but it is not so frequently adopted in algorithmic circles. Design is design is design, and the rules of design don't change just because we are designing a genetic algorithm or a classifier system instead of an airplane or an automobile.

A key ingredient of this methodological prescription is economic design theory. By "economic" I mean that the models need to have *low costs in use*. In aerodynamics, designers use "little models" such as lift coefficients and drag coefficients to quickly and cheaply understand the magnitude of lift and drag forces an aircraft is likely to encounter and how those forces will scale as the aircraft is made larger and smaller. Genetic algorithmists and classifier system designers need to find and use appropriate "little" models to understand the various facets of GA and LCS design and how the "forces" acting on our systems scale as the systems change "size." Too much of our theory literature is devoted to elegant equations that tell us little about the design of our systems. The Wright brothers eschewed the Navier-Stokes equation in favor of lift and drag coefficients, much to the betterment of modern aviation. Perhaps we should do likewise.

One aspect of design theory that comes up in all contexts is the use of *boundedly difficult* test functions. In the domain of GAs, I have used the term *deception* to describe a key facet of GA problem difficulty. Some researchers are confused about the goals of such work, but the main idea is to imagine a system's problem from hell and use boundedly hellish problems to test the procedure. In general, we know the problem from hell is too difficult to solve quickly, but we should not give up on designing procedures that scale nicely on problems of lesser or bounded difficulty. Competent GAs being designed today have this nice scaling property and similar continuing concern for problem difficulty in LCS work should pay dividends.

A Return to LCSs. Recently, I have returned to a more active role in LCS research through collaborations with a number of LCS researchers, and I have been pleased (1) by the amount of fun that I'm having, (2) by the amount of progress that has been made in the field, (3) that my old LCS knowledge isn't completely useless, and (4) that the lessons of my competent GA journey appear to be quite applicable to the project of competent LCSs. I suppose I shouldn't be surprised by having fun with LCSs. The design of general-purpose learning devices such as LCSs is an engaging business, pure and simple. And I suppose that I shouldn't be surprised by the progress in the field. For a time it almost looked as if LCSs might die off, but stalwarts led largely by Stewart Wilson have kept the fire burning with important ties to the reinforcement learning literature and a number of novel approaches.

It does strike me that much of the progress has been made on the apportionment-of-credit/reinforcement-learning side of the ledger. The nature of the genetic algorithms in use appears not to have been much affected by the GA/ES/GP innovations of the last decade, and I hope that this an area where I might be able to make a contribution. At the very least, the competent GAs that have been developed over the last decade should be adapted to LCS usage and this should benefit the search for appropriate rules in difficult problems. Finally, it seems to me that the very complexity of the LCS design task deserves a systematic design methodology, a tractable design theory, and a systematic means of integrating and coordinating the function of different subsystems. My struggle with the design of competent GAs was greatly aided by discovering and using these things, and I believe that our continued struggle toward competent learning classifier systems will be similarly rewarded by adopting an analogous approach.

6 Rick L. Riolo

There are many answers to the question "What is a classifier system," all of them reflecting different views of the field and different uses for classifier systems. For me, the most interesting way to view a Holland/Burks (Michigan-style) classifier system [34] is as a way to *model* adaptive systems [33]. To that end, such a classifier system should have most or all of these general characteristics:

- A message board for communication.
- Rule-based representation of knowledge.
- A competition for rules to become active, biased by inputs, past performance, and predictions of expected outcomes.
- Parallel firing of rules, with consistency and coordination of activity arising endogenously, as an emergent property established and maintained by the dynamics of the bidding processing. Explicit conflict resolution is strictly enforced only at the effector interface.
- Credit allocation is done by temporal difference (TD) methods of some type, e.g., the traditional bucket-brigade algorithm, some kind of profit-sharing scheme, Q-learning algorithms, and so on. Note that there might be multile kinds of credit being allocated at the same time, e.g, traditional strength representing expected payoff based on past performance, some measure of rule payoff accuracy or consistency [83] or some measure of a rules ability to predict future state [59,69].
- Rule discovery is done by heuristics appropriate to the system being modeled. Examples include triggered coupling to capture asynchronic causal connections [29,63], surprise-triggered prediction [35], or traditional genetic algorithms with with mutation and recombination.

When it doesn't conflict with other modeling goals, a classifier system should also have one other feature which I think was in Holland's original view, namely it should use a *simple* rule syntax and semantics. One reason to use simple rules (and a simple architecture and mechanisms in general) is to retain the

possibility of taking advantage of parallel hardware [62,75]. But in addition, the use of simple rules (and mechanisms) makes it easier to build mathematical models which might be analytically tractable [30,80]. However, despite these arguments for simplicity, both for some modeling goals and for classifier systems being designed to do a particular task, it may be more productive to allow more complex rules, both in terms of matching capabilities and processing power.

Over the past ten years there has been much emphasis on using classifier systems as just that, i.e., systems to solve classification problems of one kind or another. In that context, much progress has been made in improving classifier system performance on a wide variety of problems [49]. However, when using classifier system for modeling purposes, the goal is not just to get the best performance for the least computational effort. Instead, a more important criteria is how well the classifier system exhibits the relevant behavior of the system being modeled, using mechanisms plausible in that context. Thus studies which most interest me are those that either: (A) explicitly have as a goal modeling some cognitive or other adaptive system, as in [36,8,35,19,38,2,70,14], or (B) explore the fundamental dynamical properties of classifier systems with particular architectures and mechanisms, with an an eye toward understanding what kinds of models could be built with such systems [81,22,61,69].

Finally, I must admit that the kinds of systems that are most interesting to me also are the most difficult to study and understand. There is no question that in the past five to ten years much has been learned by studying simpler systems like ZCS [82], and that what we have learned will serve as a good base for future research. However, for classifier systems to reach the full potential originally outlined for them by Holland, I think we must move toward research on more complex systems. In particular, these include classifier systems which allow multiple rules to fire and post messages in parallel, which have the potential to link rule activation over time by way of tagging mechanisms, and which are set in environments that only give "payoff" occassionally and are rich enough to require extensive generalization. These are, then, environments which will require classifier systems to construct and use general, multi-step models if they are to perform well. Thus the recent work on multi-step environments and non-markov environments [51], and anticipatory cfsystems [71] are, to me, signs that the field is advancing very well.

Of course as we again move toward running more complex classifier systems, we should expect difficult issues and problems to arise which are not generally seen in simpler systems. In particular, I expect the biggest challenges to be a result of the fact that such classifier systems will have a rich "ecology" of rules, consisting of intricate interactions and complicated dependencies between rules which are involved in both competitive and cooperative relations with each other. For instance, we shall again be faced with the emergence of various kinds of parasites and free riders, which are ubiquitous in natural ecologies and other similarly complex adaptive systems. Since natural ecologies don't have an externally imposed task or performance metric, parasites are just another part of the glory of nature. On the other hand, complex adaptive systems that do have an

external performance metric (e.g., individual metazoans must survive to reproduce) have developed a host of complex mechanisms to manage problems that always arise in multi-level organizations of competing entities which also must cooperate to perform best [11,7,55]. Both for those who are taking an engineering approach to classifier systems, i.e. who want to design and use them for particular exogenously determined tasks, and for those who want to use classifier systems to model complex adaptive systems, these and other unanticipated side effects will provide great challenges [39].

7 Robert E. Smith

To begin to consider learning classifier systems (LCSs), one must first consider what an LCS is.

This is not as clear as one might imagine. A typical description of a LCS will include rules, usually taken from the common {1,0,#} syntax, that are acting as population members in a genetic algorithm. The typical description will also include some form of match-and-act, conflict resolution, and credit assignment mechanisms, that facilitate the rules interacting to influence some "environment". There are typically two variations of LCS, the "Michigan approach", where each rule is a separate individual in the GA population, and the "Pitt approach", where each GA population member is a complete set of rules for the given application environment. Comments in this section will begin with an assumption of the "Michigan approach" LCS, but it is important to note bridges between these two approaches are an important area for further investigation. Although the elements discussed above are typical to LCS descriptions, they may not define the LCS approach. In particular, specific implementation details of LCS syntax, and the system components that facilitate rule interactions with the environment, may obscure the essence of the LCS approach. Fundamentally, the LCS is defined by a population of entities that

- act individually, responding to, and taking actions on, an external environment,
- while evolving as population members, under the action of evolutionary computation.

This definition is independent of syntax or implementation details. Given this definition, the most significant complexity in the LCS approach is the need for co-evolution of entities.

Co-evolution as an approach to solving complex problems is the key thrust of the LCS approach. Co-evolution is at the cutting-edge of evolutionary computation research. In general, it involves a complex balancing act between the competitive pressures of evolution, and the cooperative interactions needed to positively effect the environment. Co-evolution brings in some of the most complex issues of evolutionary systems, including emergence the maintenance of steady-state diversity, emergence of species, symbiosis, parasitism, and others.

Because these issues involve evaluating the utility of functional interdependent entities, the issue of credit assignment (and related issues of conflict resolution) must be carefully considered. Fortunately, recent advances in machine learning (specifically in reinforcement learning) have substantially clarified these issues. While the LCS credit assignment and conflict resolution schemes of the past relied on tenuous analogies as their bases, well established reinforcement learning schemes now provide a firmer foundation upon which to build LCS advances.

Within the field of reinforcement learning, there are substantial questions to which the co-evolutionary approach of the LCS may provide answers. Chief amongst these is the need for schemes that automatically form generalizations. Another issue of importance is the formation of internal memory processing to cope with non-Markovian environments. Both of these issues depend on complex interactions of entities (like rules and associated messages), for which the co-evolutionary approach of the LCS may be well suited.

Although reinforcement learning provides a framework in which LCSs are likely to advance, it is important that this framework not become a limitation. In particular, reinforcement learning usually focuses on well-defined concepts of Bellman optimality. While these formalisms provide a well-defined basis for evaluating and developing LCS technology, they may not encompass all the ways in which LCSs may be used.

So-called "artificial intelligence" and "machine learning" technologies have often responded to an antiquated vision of computation, where computers provide faster, more accurate solutions than humans. In this vision, concepts of well-defined optimality are the goals of AI. However, these goals are not those we would usually assign to humans. Humans are usually asked simply find solutions that are "better" than those found in the past, in some sense that has no clear mathematical definition. Moreover, there is often a real world premium on human solutions that are simply innovative or creative. Such concepts are unlikely to receive mathematical definition.

Given the ubiquity of networked computers, more modern visions of computation are emerging, where computers may be expected to provide innovation and creativity, like their human counterparts. When one observes that evolution is directed at ongoing adaptation and new ways of exploiting available resources, the co-evolutionary approach of the LCS may be of particular value within this new vision of computation.

In summary, important areas for immediate, formal investigation of LCSs include:

- Formal consideration of co-evolution,
- Integration of LCSs within the framework of reinforcement learning (including generalization and application to non-Markovian tasks).

As these investigations advance, application of LCSs should continue to be explored, both within realms of formally defined problems, and in systems that required automated innovation, novelty, and creativity.

8 The Editors

8.1 Pier Luca Lanzi

Since spring 1997, I have been teaching learning classifier systems as a part of a course on Knowledge Engineering and Expert Systems for Master students at the Politecnico di Milano. Teaching requires *neat definitions* to improve the students' understanding of new concepts and *clear motivations* to ground abstract models to real world problems. Thus teaching learning classifier systems necessitates an answer to two basic, though important, questions: *what* is a learning classifier system? And *why* do we use learning classifier systems?

As this chapter witnesses, there are many ways of answering the former question. Among the possible ones, my favourite answer is:

> Learning classifier systems are a Machine Learning *paradigm* introduced by John Holland in 1978. In learning classifier systems an agent learns to perform a certain task by *interacting* with a partially unknown environment from which the agent receives feedback in the form of numerical *reward*. The incoming reward is exploited to guide the *evolution* of the agent's *behavior* which, in learning classifier systems, is represented by a set of rules, the *classifiers*. In particular, *temporal difference learning* is used to estimate the goodness of classifiers in terms of future reward; *genetic algorithms* are used to favour the reproduction and recombination of better classifiers.

I like the term "*paradigm*" because it stresses the fact that classifier systems *do not* specify an algorithm but they characterize an *entire class* of algorithms. I use "*interacting*" and "*reward*" to point out that learning classifier systems tackle *also* reinforcement learning problems and therefore should be considered a reinforcement learning technique. The terms "*evolution*," "*genetic algorithms*," and "*behavior*" highlight the fact that, in classifier systems, learning is viewed as a process of ongoing *adaptation* to a partially unknown environment, *not* as an optimization problem as in most reinforcement learning. Finally, the term "*temporal difference learning*" states that the methods used to distribute the incoming reward to classifiers are analogous to (and sometimes the same as) those techniques employed in "traditional" reinforcement learning.

Given that there is a general agreement on what is a classifier system yet a question remains: *why do we use learning classifier systems?*

This question can be answered according to two different perspectives. First, we can look at learning classifier systems as reinforcement learning techniques. In such a case to answer the question above we should present a set of applications in which learning classifier systems prove to be better than "traditional" reinforcement learning techniques. Alternatively, we can observe that learning classifier systems are more general than those traditional reinforcement learning techniques that are inspired by methods of Dynamic Programming (e.g., Watkins' Q-learning [79]). Those techniques in fact usually make a number of assumptions on the environment (e.g., the environment must be a Markov Decision

Process) and on the agent's goal (e.g., the agent's goal must be formally defined as a maximization problem) that learning classifier systems do not require. For instance, the goal of a learning classifier system might be very general (e.g., to *survive*) and not expressible in terms of an optimization problem. From this viewpoint, to answer the question "Why do we use learning classifier systems?" we should present a number of problems that classifier systems can solve but other reinforcement learning techniques cannot.

Whether we look at classifier systems as a reinforcement learning technique or as a more general framework it is interesting to note that in the literature only a few people have presented experimental results to support the use of learning classifier systems in place of other techniques.

Consider for example the area of reinforcement learning applications. It is general recognized that a sure advantage of learning classifier systems is their *generalization* capability. By means of don't care symbols (#) classifier systems can develop a compact representation of the concepts learned, whereas the complexity of traditional reinforcement learning technique grows exponentially in the problem size. At this point, the reader may argue that since generalization supports the use of learning classifier systems, then there are many papers that discuss the generalization capabilities of classifier systems. But, if we look at the works published in these twenty years, we note that most of the papers concerning generalization focus on the representational capabilities of classifier systems rather than on the degree of generalization that classifier systems can develop. Some authors (e.g., Riolo [61]) showed that classifier systems can develop interesting generalizations. But until 1996 no author provided extensive results to support the hypothesis that classifier systems could tackle reinforcement learning problems better than tabular techniques *because of their generalization capabilities*.

In 1996 Wilson [84] (see also [85]) presented a set of initial results showing that the solutions developed by his XCS classifier systems can be *significantly* more compact than that required by tabular techniques. Wilson's work was later extended by Kovacs [40] who hypothesized that Wilson's XCS develops the most general and accurate representation of the concept learned by the agent; Kovacs supported his *optimality* hypothesis experimentally.

I think that when looking at Wilson's results most people focused more on the novel classifier system architecture that Wilson proposed (XCS) rather than on the raw results that his XCS was able to achieve in terms of generalization capabilities. For instance, in this volume two papers, Booker [9] and Kovacs [41], discuss what is an adequate definition of classifiers fitness, while generalization is just partially discussed by Kovacs [41] and by Shaun and Barry [66].

On the other hand if we give a "crude" look at Wilson's results [85] we find that XCS was able to evolve a solution made of nearly 30 classifiers for a reinforcement learning problem which would require a Q-table equivalent to more or less 550 classifiers. Although these results are limited to a set of grid environments, they represent the first *direct* and *crude* comparison between classifier systems and tabular reinforcement learning. For this reason, I believe that these

results, as well as those in [40], should be consider fundamental to classifier system research since they contribute (with respect to the reinforcement learning framework) to give an answer to a very important question, i.e., why do we use learning classifier systems?

For this reason, I think that the generalization capabilities of classifier systems should be extensively investigated in the next years so to provide a more solid basis and stronger motivations to the research in this area.

Reinforcement learning represents only *one* possible way of looking at classifier systems. As I previously observed, reinforcement learning make strong assumptions on the environment and, *most important*, on the goal of the learning process. In contrast, learning classifier systems do not make particular assumptions on the environment and on the agent's goal which is generally defined in terms of *adaptation to the environment*. But even of we change our viewpoint on classifier systems the question "Why do we use learning classifier systems?" is still important.

One possible answer to this question is provided by the many applications of classifier systems to the problem of modeling the emergence of complex behaviors in real systems. For instance, learning classifier systems can be used to model adaptive agents in artificial stock markets (e.g., Brian Arthur et al. [2], Lebaron et al. [53], Vriend [76,77,78], Marimon *et al.* [54]). On the other hand, it is not completely clear at the moment whether classifier systems are the *only* approach to tackle these types of applications. For instance a number of researchers are currently applying other techniques (e.g., reinforcement learning [57]) to model artificial stock markets.

Recently, Smith et al. [68] (see also [67, this volume]) presented an application of classifier systems which *undoubtedly* cannot be easily modeled with other learning paradigm and thus strongly supports classifier systems. In particular Smith et al. [68] developed a classifier systems that can *discover innovation* in terms of novel fighting aircraft maneuvering strategies. Their results, in my opinion, are fundamental to classifier system research because they prove that learning classifier systems can successfully tackle problems that are beyond the *plain* reinforcement learning framework; for this reason they also provide motivation to the use of classifier systems in very diverse and "*previously unthinkable*" application areas.

Since the early days of classifier systems, more than 400 papers on classifier systems have been published. Looking at the bibliography at the end of this book we note that there was a time in the mid 1990s when there was only a little research on classifier systems. I remember that in 1997 when I presented my first work on XCS [45] at ICGA97 [3], there were *only* three papers on learning classifier systems: mine, Holmes' [37], and Federman's [20]. Basically, we were not enough even to fill one session; at that time I thought that the area I was working in was actually quite small. In 1998, during GP98 [44], there were two sessions on learning classifier systems and more than ten papers. In 1999, at GECCO [4] there were four sessions on learning classifier systems and

one workshop: more or less thirty papers were presented. But why are classifier systems experiencing this renaissance?

There are surely different interpretations of what happened in the last five years, of this reborn interest in learning classifier systems. My belief is that much of this is due to the effort of a number of people who traced novel research directions, suggesting new *motivations* so to provide answers to some important questions about classifier systems. These people renewed part of this area without giving up original Holland's principles and their unique flavor. And certainly, part of the current renaissance is also due to the people who followed and currently follow those directions.

8.2 Wolfgang Stolzmann

The field of learning classifier systems (LCS) is young. However, two approaches have been developed: the Michigan Approach and the Pitt Approach. I restrict myself to the Michigan approach because it is the classical approach. The foundations for LCS were laid by Holland [26] within the framework of his theoretical studies of genetic algorithms. The first LCS, called CS-1, was introduced by Holland and Reitman [36]. Since then many different types of LCS have been described in the literature. Thus, it is difficult to give a unified definition for an LCS. Therefore I would like to focus on the points that I consider most important.

A learning classifier system (LCS) is a machine learning system that learns a collection of simple production rules, called classifiers. Its knowledge is represented in a classifier list. An LCS can be regarded as a learning agent that acts in an arbitrary environment. In order to interact with the environment it has an input interface with detectors for sensory information from the environment and an output interface with effectors for motor actions. Each detector of the input interface contains information about one attribute of the environmental state and delivers values 0 and 1. Thus, all an LCS knows about the environmental state is represented in a bit-string called message. A message is the internal representation of an environmental state. Each component contains the information of one detector. Formally messages belong to the set {0,1}k (k is the number of detectors). A classifier is a condition/action-rule. If the condition-part of a classifier matches the current message, then the classifier can become active. The action-part of an active classifier interacts with the effectors and causes a motor action of the LCS. *Conditions* belong to the set $\{0,1,\#\}^k$. A condition is matched by any message that has 0's and 1's in exactly the same positions as the 0's and 1's in the condition. A '#' in a condition is called a "don't care"-symbol. It matches both a 0 and a 1. For example the condition 1## is matched by any message that starts with a 1; #00 is matched by 100 and 000 and the condition 010 is only matched by the message 010. A learning classifier system derives its name from its ability to learn to classify messages from the environment into general sets like {m — m starts with a 1}, {100 , 000} and {010}.

The basic execution cycle of an LCS consists in an iteration of the following steps:

1. A message m is perceived by the input interface.
2. m is compared with the condition parts of all classifiers of the current classifier list. The matching classifiers form a match set.
3. One classifier out of the match set is selected by roulette-wheel selection [21] or another kind of competition.
4. The selected classifier becomes active. Its action part interacts with the output interface and causes a motor action of the LCS.
5. Reinforcement learning is applied (see below).
6. Rule discovery is applied to produce new classifiers while replacing other, low- strength classifiers (see below).

As mentioned above I focus on the points that I consider most important. I ignored the fact that classifiers can produce new messages so that the LCS has to deal with a message list, I ignored that a classifier can contain more than one condition and I ignored the possibility that more than one classifier can become active during one execution cycle. If there is more than one active classifier, then the LCS has to deal with inconsistent information for the output interface. There are two levels of learning in an LCS: A first level of learning called credit assignment, consists of reinforcement learning on the classifiers. A classifier is reinforced (i.e. its rule strength is modified) in dependence on environmental reward called payoff (Bucket Brigade Algorithm, Profit Sharing Plan, Q-Learning, ...). A second level that is usually independent of the first one consists of rule discovery in order to generate new classifiers. For that an LCS typically uses genetic algorithms. Up to now Stewart W. Wilson's XCS is the most important kind of LCS (cf. "State of XCS Classifier System Research", this volume). Tim Kovacs's LCS bibliography [42], contains 33 entries out of 479 that deal with XCS. XCS were applied in various environments like the (n+2^n)-Multiplexer (e.g. [83]) Woods-environments (e.g. [83,48]), and Finite-State-Worlds (e.g. [5,6]. The results show that an XCS is capable of forming complete reward-maps, i.e. $X \times A \Rightarrow P$ maps where X is the set of states, A is the set of actions and P is the set of expected payoffs. "It does not learn what input sensation will follow a given action. That is, it does not learn an $X \times A \Rightarrow Y$ map, where Y is the following sensation" (Wilson [83, p. 173]). Or in other words it does not learn an internal world model. Holland [32], Riolo [59], and Stolzmann [72, this volume] show how internal world models can be learned in LCS. Very simple internal world models are used in reinforcement learning especially in Dyna-Q (e.g. Sutton [73]; Sutton & Barto [74, p. 233]) Future work will have to show how LCS can be used to learn internal world models with a minimum number of classifiers and how these internal world models can be used in reinforcement learning. Furthermore, in my opinion future LCS research should include:

- real world applications,
- a definition of standard test environments like the (n+2n)-Multiplexer, Woods-environments and Finite-State-Worlds in order to facilitate the comparison of different approaches,

– applications in non-deterministic environments, especially in non-Markov environments (cf. Lanzi & Wilson [52, in press]),
– and LCS work should include comparisons to relevant reinforcement learning work.

8.3 Stewart W. Wilson

A classifier system is a learning system based on Darwinian principles. The system consists of two parts: a collection or "population" of condition-action rules called classifiers; and an algorithm for utilizing, evaluating, and improving the rules. Holland's great insight was to see that a learning system might be based on a Darwinian process applied to a rule base. To me, his idea is intriguing and inspiring because it represents a plausible computational picture of our mental world. Moreover, it is a proposal that seems rich enough to encompass, eventually, very difficult questions like multi-level autonomous adaptation.

Besides animals' more-or-less well-learned and reflexive condition-action responses, we are constantly making predictions, often relative to rewards, about the consequences of behavior, and choosing accordingly. But there is a "fog of uncertainty": we are not quite sure what will result in the present situation, not even sure *if* the present situation is what we think it is. But we go with what we have, try things, and attempt to register the outcomes. In short, everything we know is hypothetical, whether perceptual or cognitive. We hope to find better hypotheses, ones that more accurately define the antecedent condition and therefore predict better, and also cover the largest domain and reduce our mental effort. New hypotheses come largely from pieces of what works already—where else?—but the process is mostly unclear and mysterious. It is often invisible. A new hypothesis arrives suddenly; it is not consciously cranked out. There may be mental work, but then the key steps just happen.

To all this would seem to correspond, in Holland's scheme, the population of classifiers viewed as a set of competing hypotheses, each in some state of relative confirmation, subject to modification and replacement by a better hypothesis born suddenly from chance recombination and mutation of existing above-average material. Some classifiers generalize over sensory inputs, providing a rudimentary imitation of early-stage perception and an example of symbol grounding—the classifiers' messages being the symbols. Other classifiers respond to internal messages by posting further messages, affording computational completeness but more importantly, room to erect any so-called cognitive structure. Still other classifiers cause actions and thus complete the loop with the environment, whose response keeps the system viable and fuels the Darwinian calculation. The principles of variation and selection being ubiquitous elsewhere in life, it is natural to look seriously for them inside the head, and for this Holland has provided a splendid computational framework that merits every effort to understand and develop its implications.

The framework is many-faceted, and it's not surprising that besides successes, research over the past twenty years has identified certain tensions among the framework's elements. Two seem most important. One tension arises because

the classifiers are in Darwinian competition, but a certain amount of cooperation among classifiers is necessary. For example, to reach external payoff the system may depend on a chain of classifiers acting sequentially through time: the chain members cooperate in the sense that each member depends on the activation of its predecessors and on the ability of its successors to push on to payoff. But the competition may disturb this. Selection of one member of the chain reduces the survival chances of the other members, so that the chain may break causing all members to lose out. This competitive/cooperative tension occurs in several guises, and the consequence is often instability leading to reduced performance.

A key part of the solution was the early proposal—made mainly for other reasons—to restrict the action of the genetic algorithm to the match sets, i.e., it was proposed to replace the panmictic (population-wide) GA with a GA operating just in the niches defined by each match set. This lessens the competition between sequential classifiers, since sequential classifiers occur in distinct match sets. The cost in discovery capability seems to be minor, since classifiers in separate match sets are basically solving different problems and crosses between them are not particularly fruitful. However, the niche GA introduces a new hazard: overgeneralization. A more general classifier will tend to show up in more match sets. With a niche GA, it will have more reproductive opportunities so that generality becomes an advantage in itself, and increasingly general—eventually overgeneral—classifiers will tend to multiply in the population.

The second major tension is between performance and generality. Consider an individual classifier. Its current prediction ("strength" in the older terminology) is an average of payoffs received in the situations (niches) in which it matches and its action is taken. Some situations may have large payoffs, others quite small ones: all are averaged in the prediction, which masks any underlying variance. However, if the variance is great, there can be a negative effect on system performance. In low-payoff situations one classifier's action may be chosen over a better action if the first classifier's prediction is misleadingly high due to averaging in of payoffs received in higher-payoff niches. The result will be degradation of performance in some degree. Moreover, the offending classifier, and the degradation, will persist since selection under the GA is based on the prediction. Thus the framework's potential for generalizing classifier conditions—essential for formation of percepts and concepts—is at odds with performance.

The first step toward solution was to ask: What would happen if fitness for the GA were based not on the prediction, but on some measure of the prediction's accuracy? Then the offending high-variance classifier above would not receive a high fitness, and would be eliminated from the system. On the other hand, a classifier with low prediction variance would survive and multiply; desirably, such a classifier would *not* cause the kind of performance error cited earlier. However, this solution is not complete since basing fitness on prediction accuracy does not in itself address the framework's need to find classifiers that are *both* general and accurate. Fitness based on accuracy would tend to favor very specific classifiers, since greater accuracy usually goes with greater specificity.

I would argue that these two tensions, competition vs. cooperation and performance vs. generality, were at the root of much of the field's earlier difficulties. Interestingly, the full solutions to each of them are complementary. I said that the first step in solving competition/cooperation was the niche GA. The second step turns out to be fitness based on accuracy, because that stops the niche GA's tendency toward overgeneralization. I said the first step in solving performance/generality was fitness based on accuracy. The second step turns out to be a niche GA, since its generalization pressure is just what is needed to push toward classifiers that are maximally general as well as accurate. Thus fitness based on accuracy and a niche GA *in combination* appear to overcome the principal problems of classifier systems' first twenty years, opening the way to a strong advance in the period just beginning.

There are many promising directions for classifier system research. Some are suggested at the end of my "State" paper in this volume. But an important one not listed there would take a new look at an old subject—the full Holland framework itself. In recent years there has been something of a retreat from research on the full system with its message list and other aspects. Progress has come from paring the system down and putting a simpler beast under the microscope to see more clearly the function of various mechanisms. The principal result, in my opinion, has been understanding the two tensions discussed above, and their solution. Now it is time to go back to the full framework and see what happens if in it fitness is indeed based on accuracy and the GA occurs in the niches. As far as I know, that has not been done. The tests occurred and were successful in the simpler systems. It seems quite possible, and worth the effort required, to try the changes in the full system, testing on a problem in which the message list's capacity to represent internal state is essential for high performance.

References

1. Manu Ahluwalia and Larry Bull. A Genetic Programming-based Classifier System. In Banzhaf et al. [4], pages 11–18.
2. W. Brian Arthur, John H. Holland, Blake LeBaron, Richard Palmer, and Paul Talyer. Asset Pricing Under Endogenous Expectations in an Artificial Stock Market. Technical report, Santa Fe Institute, 1996. This is the original version of LeBaron1999a.
3. Thomas Bäck, editor. *Proceedings of the 7th International Conference on Genetic Algorithms (ICGA97)*. Morgan Kaufmann: San Francisco CA, 1997.
4. Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*. Morgan Kaufmann: San Francisco, CA, 1999.
5. Alwyn Barry. Aliasing in XCS and the Consecutive State Problem: 1 – Effects. In Banzhaf et al. [4], pages 19–26.
6. Alwyn Barry. Aliasing in XCS and the Consecutive State Problem: 2 – Solutions. In Banzhaf et al. [4], pages 27–34.

7. John Tyler Bonner. *The Evolution of Complexity*. Princeton University Press, Princeton, New Jersey, 1988.
8. Lashon B. Booker. *Intelligent Behavior as an Adaptation to the Task Environment*. PhD thesis, The University of Michigan, 1982.
9. Lashon B. Booker. Do We Really Need to Estimate Rule Utilities in Classifier Systems? In Lanzi et al. [50], pages 125–142. (this volume).
10. Lashon B. Booker, David E. Goldberg, and John H. Holland. Classifier Systems and Genetic Algorithms. *Artificial Intelligence*, 40:235–282, 1989.
11. Leo W. Buss. *The Evolution of Individuality*. Princeton University Press, Princeton, New Jersey, 1987.
12. H. J. Chiel and R. D. Beer. The brain has a body: Adaptive behavior emerges from interactions of nervous system, body and environment. *Trends in Neurosciences*, 20:553–557, 1997.
13. Marco Colombetti and Marco Dorigo. Evolutionary Computation in Behavior Engineering. In *Evolutionary Computation: Theory and Applications*, chapter 2, pages 37–80. World Scientific Publishing Co.: Singapore, 1999. Also Tech. Report. TR/IRIDIA/1996-1, IRIDIA, Université Libre de Bruxelles.
14. Michael Sean Davis. *A Computational Model of Affect Theory: Simulations of Reducer/Augmenter and Learned Helplessness Phenomena*. PhD thesis, Department of Psychology, University of Michigan, 2000.
15. Marco Dorigo. Alecsys and the AutonoMouse: Learning to Control a Real Robot by Distributed Classifier Systems. *Machine Learning*, 19:209–240, 1995.
16. Marco Dorigo and Marco Colombetti. Robot shaping: Developing autonomous agents through learning. *Artificial Intelligence*, 2:321–370, 1994. ftp://iridia.ulb.ac.be/pub/dorigo/journals/IJ.05-AIJ94.ps.gz.
17. Marco Dorigo and Marco Colombetti. *Robot Shaping: An Experiment in Behavior Engineering*. MIT Press/Bradford Books, 1998.
18. E.B. Baum. Toward a model of intelligence as an economy of agents. *Machine Learning*, 35:155–185, 1999.
19. J. Doyne Farmer, N. H. Packard, and A. S. Perelson. The Immune System, Adaptation & Learning. *Physica D*, 22:187–204, 1986.
20. Francine Federman and Susan Fife Dorchak. Information Theory and NEXTPITCH: A Learning Classifier System. In Bäck [3], pages 442–449.
21. David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Mass., 1989.
22. David E. Goldberg. Probability Matching, the Magnitude of Reinforcement, and Classifier System Bidding. *Machine Learning*, 5:407–425, 1990. (Also TCGA tech report 88002, U. of Alabama).
23. H. Hendriks-Jansen. *Catching Ourselves in the Act*. MIT Press, Cambridge, MA, 1996.
24. S. Hofmeyr and S. Forrest. Immunity by design: An artificial immune system. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 1289–1296, San Francisco, CA, 1999. Morgan-Kaufmann.
25. J. H. Holland, K. J. Holyoak, R. E. Nisbett, and P. Thagard. *Induction: Processes of Inference, Learning, and Discovery*. MIT Press, 1986.
26. John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975. Republished by the MIT press, 1992.
27. John H. Holland. Adaptation. In R. Rosen and F. M. Snell, editors, *Progress in theoretical biology*. New York: Plenum, 1976.

28. John H. Holland. Adaptive algorithms for discovering and using general patterns in growing knowledge bases. *International Journal of Policy Analysis and Information Systems*, 4(3):245–268, 1980.
29. John H. Holland. Escaping brittleness. In *Proceedings Second International Workshop on Machine Learning*, pages 92–95, 1983.
30. John H. Holland. A Mathematical Framework for Studying Learning in Classifier Systems. *Physica D*, 22:307–317, 1986.
31. John H. Holland. Escaping Brittleness: The possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems. In Mitchell, Michalski, and Carbonell, editors, *Machine learning, an artificial intelligence approach. Volume II*, chapter 20, pages 593–623. Morgan Kaufmann, 1986.
32. John H. Holland. Concerning the Emergence of Tag-Mediated Lookahead in Classifier Systems. *Special issue of Physica D (Vol. 42)*, 42:188–201, 1989.
33. John H. Holland. *Hidden Order: How Adaptation Builds Complexity.* Addison-Wesley, Reading, MA, 1995.
34. John H. Holland and Arthur W. Burks. Adaptive Computing System Capable of Learning and Discovery. Patent 4697242 United States 29 Sept., 1987.
35. John H. Holland, Keith J. Holyoak, Richard E. Nisbett, and P. R. Thagard. *Induction: Processes of Inference, Learning, and Discovery.* MIT Press, Cambridge, 1986.
36. John H. Holland and J. S. Reitman. Cognitive systems based on adaptive algorithms. In D. A. Waterman and F. Hayes-Roth, editors, *Pattern-directed inference systems.* New York: Academic Press, 1978. Reprinted in: Evolutionary Computation. The Fossil Record. David B. Fogel (Ed.) IEEE Press, 1998. ISBN: 0-7803-3481-7.
37. John H. Holmes. Discovering Risk of Disease with a Learning Classifier System. In Bäck [3]. http://cceb.med.upenn.edu/holmes/icga97.ps.gz.
38. Keith J. Holyoak, K. Koh, and Richard E. Nisbett. A Theory of Conditioning: Inductive Learning within Rule-Based Default Hierarchies. *Psych. Review*, 96:315–340, 1990.
39. Kevin Kelly. *Out of Control.* Addison-Wesley, Reading, MA, 1994.
40. Tim Kovacs. Evolving Optimal Populations with XCS Classifier Systems. Master's thesis, School of Computer Science, University of Birmingham, Birmingham, U.K., 1996. Also tech. report CSR-96-17 and CSRP-96-17 ftp://ftp.cs.bham.ac.uk/pub/tech-reports/1996/CSRP-96-17.ps.gz.
41. Tim Kovacs. Strength or Accuracy? Fitness calculation in learning classifier systems. In Lanzi et al. [50], pages 143–160. (this volume).
42. Tim Kovacs and Pier Luca Lanzi. A Learning Classifier Systems Bibliography. In Lanzi et al. [50], pages 323–350. (this volume).
43. John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs.* The MIT Press, Cambridge, MA, 1994.
44. John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors. *Genetic Programming 1998: Proceedings of the Third Annual Conference.* Morgan Kaufmann: San Francisco, CA, 1998.
45. Pier Luca Lanzi. A Study of the Generalization Capabilities of XCS. In Bäck [3], pages 418–425. http://ftp.elet.polimi.it/people/lanzi/icga97.ps.gz.
46. Pier Luca Lanzi. Adding Memory to XCS. In *Proceedings of the IEEE Conference on Evolutionary Computation (ICEC98).* IEEE Press, 1998. http://ftp.elet.polimi.it/people/lanzi/icec98.ps.gz.

47. Pier Luca Lanzi. *Reinforcement Learning by Learning Classifier Systems*. PhD thesis, Politecnico di Milano, 1998.
48. Pier Luca Lanzi. An Analysis of Generalization in the XCS Classifier System. *Evolutionary Computation*, 7(2):125–149, 1999.
49. Pier Luca Lanzi and Rick L. Riolo. A Roadmap to the Last Decade of Learning Classifier System Research (from 1989 to 1999). In Lanzi et al. [50], pages 33–62. (this volume).
50. Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors. *Learning Classifier Systems: An Introduction to Contemporary Research*, volume 1813 of *LNAI*. Springer-Verlag, Berlin, 2000.
51. Pier Luca Lanzi and Stewart W. Wilson. Optimal classifier system performance in non-Markov environments. Technical Report 99.36, Dipartimento di Elettronica e Informazione - Politecnico di Milano, 1999. Also IlliGAL tech. report 99022, University of Illinois.
52. P.L. Lanzi and S. W. Wilson. Toward optimal classifier system performance in non-Markov environments. *Evolutionary Computation*, 2000. to appear.
53. Blake Lebaron, W. Brian Arthur, and R. Palmer. The Time Series Properties of an Artificial Stock Market. *Journal of Economic Dynamics and Control*, 1999.
54. Ramon Marimon, Ellen McGrattan, and Thomas J. Sargent. Money as a Medium of Exchange in an Economy with Artificially Intelligent Agents. *Journal of Economic Dynamics and Control*, 14:329–373, 1990. Also Tech. Report 89-004, Santa Fe Institute, 1989.
55. Richard E. Michod. *Darwinian Dynamics: Evolutionary Transitions in Fitness and Individuality*. Princeton University Press, Princeton, New Jersey, 1999.
56. Alan Newell and Herbert Simon. *Human Problem Solving*. Prentice Hall, Engelwood Cliffs, NJ.
57. E. Oliveira, J.M. Fonseca, and N. Jennings. Learning to be competitive in the Market. 1999. Proceedings of the AAAI Workshop on Negotiation, Orlando (FL).
58. J. K. Percus, O. Percus, and A. S. Perelson. Predicting the size of the antibody combining region from consideration of efficient self/non-self discrimination. *Proceedings of the National Academy of Science*, 90:1691–1695, 1993.
59. Rick L. Riolo. Lookahead Planning and Latent Learning in a Classifier System. pages 316–326. A Bradford Book. MIT Press, 1990.
60. Rick L. Riolo. Lookahead planning and latent learning in a classifier system. Ann Arbor, MI, 1991. In the Proceedings of the Simulation of Adaptive Behavior Conference, MIT Press, 1991.
61. Rick L. Riolo. Modeling Simple Human Category Learning with a Classifier System. pages 324–333. Morgan Kaufmann: San Francisco CA, July 1991.
62. George G. Robertson. Parallel Implementation of Genetic Algorithms in a Classifier System. In John J. Grefenstette, editor, *Proceedings of the 2nd International Conference on Genetic Algorithms (ICGA87)*, pages 140–147, Cambridge, MA, July 1987. Lawrence Erlbaum Associates. Also Tech. Report TR-159 RL87-5 Thinking Machines Corporation.
63. George G. Robertson and Rick L. Riolo. A Tale of Two Classifier Systems. *Machine Learning*, 3:139–159, 1988.
64. S. A Hofmeyr and S. Forrest. Architecture for an Artificial Immune System. Submitted to Evolutionary Computation. Available at `http://www.cs.unm.edu/ steveah/ecs.ps`, 1999.
65. Samuel, A. L. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journ. R & D*, 3:211–229, 1959. Reprinted in Feigenbaum, E., and Feldman, J. (eds.), *Computer and Thoughts*, pp. 71–105, New York: McGraw-Hill, 1963.

66. Shaun Saxon and Alwyn Barry. XCS and the Monk's Problems. In Lanzi et al. [50], pages 223–242. (this volume).
67. R. E. Smith, B. A. Dike, B. Ravichandran, A. El-Fallah, and R. K. Mehra. The Fighter Aircraft LCS: A Case of Different LCS Goals and Techniques. In Lanzi et al. [50], pages 285–302. (this volume).
68. Robert E. Smith, B. A. Dike, R. K. Mehra, B. Ravichandran, and A. El-Fallah. Classifier Systems in Combat: Two-sided Learning of Maneuvers for Advanced Fighter Aircraft. In *Computer Methods in Applied Mechanics and Engineering*. Elsevier, 1999.
69. Wolfgang Stolzmann. Learning Classifier Systems using the Cognitive Mechanism of Anticipatory Behavioral Control, detailed version. In *Proceedings of the First European Workshop on Cognitive Modelling*, pages 82–89. Berlin: TU, 1996. http://www.psychologie.uni-wuerzburg.de/stolzmann/.
70. Wolfgang Stolzmann. Two Applications of Anticipatory Classifier Systems (ACSs). In *Proceedings of the 2nd European Conference on Cognitive Science*, pages 68–73. Manchester, U.K., 1997. http://www.psychologie.uni-wuerzburg.de/stolzmann/.
71. Wolfgang Stolzmann. Anticipatory classifier systems. In *Proceedings of the Third Annual Genetic Programming Conference*, pages 658–664, San Francisco, CA, 1998. Morgan Kaufmann. http://www.psychologie.uni-wuerzburg.de/stolzmann/gp-98.ps.gz.
72. Wolfgang Stolzmann. An Introduction to Anticipatory Classifier Systems. In Lanzi et al. [50], pages 175–194. (this volume).
73. Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224, Austin, TX, 1990. Morgan Kaufmann.
74. Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning – An Introduction*. MIT Press, 1998.
75. Kirk Twardowski. Implementation of a Genetic Algorithm based Associative Classifier System (ACS). In *Proceedings International Conference on Tools for Artificial Intelligence*, 1990.
76. Nicolaas J. Vriend. On Two Types of GA-Learning. In S.H. Chen, editor, *Evolutionary Computation in Economics and Finance*. Springer, 1999. in press.
77. Nicolaas J. Vriend. The Difference Between Individual and Population Genetic Algorithms. In Banzhaf et al. [4], pages 812–812.
78. Nicolaas J. Vriend. An Illustration of the Essential Difference between Individual and Social Learning, and its Consequences for Computational Analyses. *Journal of Economic Dynamics and Control*, 24:1–19, 2000.
79. C.J.C.H. Watkins. Learning from delayed reward. PhD Thesis, Cambridge University, Cambridge, England, 1989.
80. Thomas H. Westerdale. An Approach to Credit Assignment in Classifier Systems. *Complexity*, 4(2), 1999.
81. Stewart W. Wilson. Adaptive "cortical" pattern recognition. pages 188–196. Lawrence Erlbaum Associates: Pittsburgh, PA, July 1985.
82. Stewart W. Wilson. ZCS: A zeroth level classifier system. *Evolutionary Computation*, 2(1):1–18, 1994. http://prediction-dynamics.com/.
83. Stewart W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995. http://prediction-dynamics.com/.
84. Stewart W. Wilson. Generalization in XCS. Unpublished contribution to the ICML '96 Workshop on Evolutionary Computing and Machine Learning. http://prediction-dynamics.com/, 1996.

85. Stewart W. Wilson. Generalization in the XCS classifier system. In Koza et al. [44], pages 665–674. http://prediction-dynamics.com/.
86. Stewart W. Wilson. Get Real! XCS with Continuous-Valued Inputs. In Lanzi et al. [50], pages 209–220. (this volume).
87. Stewart W. Wilson. State of XCS Classifier System Research. In Lanzi et al. [50], pages 63–82. (this volume).

A Roadmap to the Last Decade of Learning Classifier System Research
(From 1989 to 1999)

Pier Luca Lanzi[1] and Rick L. Riolo[2]

[1] Artificial Intelligence & Robotics Project
Dipartimento di Elettronica e Informazione
Politecnico di Milano
lanzi@elet.polimi.it
[2] Center for Study of Complex Systems
University of Michigan
Ann Arbor, MI 48109-1120
rlriolo@umich.edu

Abstract. In 1989 Wilson and Goldberg presented a critical review of the first ten years of learning classifier system research. With this paper we review the subsequent ten years of learning classifier systems research, discussing the main achievements and the major research directions pursued in those years.

1 Introduction

Learning classifier systems are a machine learning paradigm introduced by John H. Holland. They made their first appearance in 1978 in the paper "Cognitive Systems Based on Adaptive Algorithms" by Holland and Reitmann [64]. However, classifier systems were foreshadowed in 1971 with the paper "Processing and Processors for Schemata" by Holland [56], and in the *broadcast language* introduced in [57].

In learning classifier systems an *agent* learns to perform a certain task by *interacting* with a partially unknown *environment*, using *rewards* and other feedback to guide an internal *evolutionary* process which modifies its *rule-based model* of the world. The agent senses the environment through its *detectors*; based on its current sensations and its past experience, the agent selects an action which is sent to the *effectors* in order to be performed in the environment. Depending on the effects of the agent's action, the agent occasionally receives a reward. The agent's general goal is to obtain as much reward as possible from the environment.

In his pioneer work, Holland brought together two ideas which later became key topics of the research in machine learning. The first idea is that Darwinian theory of the *survival of the fittest* can be used to trigger the adaptation of an artificial system to an unknown environment. This idea later became the basis of important research areas like Evolutionary Computation, Adaptive Behavior,

P.L. Lanzi, W. Stolzmann, and S.W. Wilson (Eds.): LCS '99, LNAI 1813, pp. 33–61, 2000.

and Artificial Life. The second idea is that an agent can learn to perform a task just by trying to maximize the rewards it receives from an unknown environment. This model of learning through "*trial and error*" interactions has been formalized and developed in the area of Reinforcement Learning, which is now a major branch of machine learning research.

Learning classifier systems have been around more than twenty years. In these two decades they have received increasing attention by many researchers belonging to many areas.

Ten years ago Wilson and Goldberg [165] presented a critical review on the first decade of learning classifier systems research discussing the results reached until 1989 and the "future" research directions. Since then a lot of work has been presented in the literature which casts new light on the properties of this learning paradigm, and which shows that learning classifier systems can tackle a large variety of applications, e.g., autonomous robotics [36], medical data analysis [8, 65], and personal software agents [166].

Today with this paper we try to provide a *roadmap* to the second decade of learning classifier system research covering the years from 1989 to 1999. This is *not* a *critical review* of the latest results and the possible future research directions, as was done by Wilson and Goldberg in 1989. In fact we believe that the papers in this book already provide such a review from different perspectives. Rather this paper is a guide to the many papers concerning learning classifier systems that have been published in the last ten years. In particular we will focus solely on Michigan-style classifier systems; for a recent discussion concerning Pittsburgh-style classifier systems we refer the reader to [102]. In this paper we will assume that the reader is familiar with the basic structure of learning classifier systems; if this is not the case we refer the interested reader to Goldberg's book "*Genetic Algorithms in Search, Optimization, and Machine Learning*" [49] for an introduction to Holland's approach to learning classifier systems. Another description of a more complex learning classifier system is Booker's GOFER-1 [10]. For an overview of more recent approaches to classifier systems, see Wilson's "ZCS: A zeroth level classifier system" [158] for an introduction to the ZCS classifier system and to "Classifier Fitness Based on Accuracy" [159] for an introduction to the most recent XCS classifier system. In this volume other models are described as well [14,131].

The remainder of the paper is organized as follows. We begin in Section 2 giving a general overview of what has happened in learning classifier systems research since 1989. Then in Section 3 and Section 4 we consider the most active research topics in this area and for each one of them we briefly summarize what has been explored and achieved in the past ten years or so. Finally, in Section 5 we consider those areas where learning classifier systems have been applied and report the main results obtained.

2 Ten Years with a Bird's Eye View

In the past ten years two main factors contributed to the research in classifier systems. The first "indirect" contribution came from the important developments in the field of Reinforcement Learning (see [76] and [140] for a survey). The second "direct" contribution came from the development of new learning classifier systems in which Holland's essential ideas are kept, but within different architectures.

2.1 Reinforcement Learning

Reinforcement Learning is usually defined as a mechanism by which an agent can learn to solve a *problem* or to perform a certain task through *trial and error* interactions [140] with an environment which provides only minimal feedback, in the form of occasional *rewards*. In particular, for both learning classifier systems and for other systems using reinforcement learning algorithms, the agent knows nothing about the task it has to learn, and it is only interested in maximizing the reward it receives. From this perspective classifier systems share many similarities with reinforcement learning methods and therefore shall be considered a reinforcement learning method. On the other hand, these two methods are also quite different. Reinforcement learning typically adopts an economic metaphor: learning is equated to the search for an optimal decision policy, and appropriate hypotheses often guarantee that such a policy exists and that the method will eventually converge to it (see for instance Watkins' Q-learning [152]). Learning classifier systems are based on a combination of an economic metaphor of local exchange, for short term learning, with a biological metaphor of evolution, for long term learning. That is, learning by reinforcement, via the local exchanges of the bucket brigade or similar algorithms, is viewed as only one part of a process of ongoing adaptation of an agent to an initially unknown environment; the other key mechanism is based on the metaphor of *rule discovery by evolution*. Because of this evolutionary component, typically convergence to satisfactory behavioral policies cannot be proved formally (except perhaps in the unrealistic limit of infinite available time), and thus has to be studied experimentally.

The last decade has seen the flourishing of research in reinforcement learning. The ongoing mathematical formalization of the reinforcement learning framework and the large number of successful applications to real-world problems has made this area one of the most promising research topics in machine learning. In the early '90s learning classifier systems indeed received benefits from the growing interest of the research community in reinforcement learning. Since classifier systems can *also* tackle reinforcement learning problems, many researchers applied them to typical reinforcement learning applications, e.g., autonomous robotics (see Section 5.1). This is the case of Dorigo and Colombetti [38,36] who reported a large number of successful applications of classifier systems to autonomous robotics. At the same time other authors applied classifier systems to general machine learning problems (e.g. supervised classification [8]), but at the beginning of this decade there was the perception that autonomous robotics

could be *the* promising area for applying classifier systems. Today things are quite changed and robotics is just *one* of the areas where classifiers systems are applied. In fact, in addition to robotics, interesting results have been reported in many other application domains, including air combat maneuvering [133], control of traffic signals [40], medical data analysis [73,65,67,68], sequence prediction [41], and supervised classification [124].

2.2 Types of Learning Classifier Systems

For many years the research on learning classifier systems[1] was done solely on the original architecture proposed by Holland [64]. Many researchers developed their own version of Holland's complex architectures in which Holland's classifier system was the main building component (e.g., Dorigo's ALECSYS [33]). All these implementations shared more or less the same features which can summarized as follows: (i) some form of a *bucket brigade* algorithm was used to distribute the rewards received to the classifiers; (ii) evolution was triggered by the *strength* parameter of classifiers; (iii) the internal message list was used to keep track of past inputs. Two "standard implementations" were made available at that time, one by Goldberg [49] and one by Riolo [113].

Holland's learning classifier system had a number of known problems which in some cases prevented the system from achieving satisfactory performance (see [165] for a discussion). To improve performance some small modifications to the original framework were proposed but interesting performance was rarely reported. Thus the need for more radical changes to the original architecture already was perceived in the early years of research. In 1987 Wilson [155] introduced a new type of "one-step" classifier system, BOOLE, particularly devised for the problem of learning Boolean functions. Wilson showed that BOOLE could learn multiple disjunctive concepts faster than neural networks. BOOLE was extended into NEWBOOLE by Bonelli et al. [9] in 1990; the performance of NEWBOOLE was compared with other two learning algorithms, the rule inducer CN2 [18] and neural networks trained by Back Propagation, on three medical domains. The results reported showed that NEWBOOLE performed significantly better than the other machine learning methods.

Separately, Booker [11] introduced GOFER-1, a new type of classifier system which in many ways anticipates more recent classifier models. In GOFER-1 the classifiers fitness is a function of both payoff and non-payoff information, and the genetic algorithm works in environmental niches instead of in the whole population. Booker applied GOFER-1 to boolean multiplexer problems obtaining near optimal performance and to the state space search problem [52] with interesting results.

Wilson [158] observed that the architecture of learning classifier systems is too complex to permit careful, revealing studies of the learning capabilities of these systems. Accordingly he simplified the original framework by introducing

[1] Again, we are restricting this paper to consideration of Michigan-style classifier systems; for a summary of work with Pittsburgh-style systems, see [102].

ZCS, a *zeroth level* classifier system [158]. ZCS differs from the original framework mainly in that (1) it has no internal message list and (2) it distributes the reward to classifiers by a technique, QBB [158], very similar to Watkins' Q-learning [152]. In this sense ZCS represents one of the first attempts to bridge the then existing gap between the credit assignment algorithms used in learning classifier systems (i.e., the bucket brigade) and those algorithms used in other reinforcement learning systems.[2] Furthermore ZCS significantly differs from previous systems by omitting several heuristics and operators that were developed to aid those systems in reaching acceptable performance (see for example Wilson [154] and Dorigo [32]). Interestingly, this basic ZCS system performed similarly to more complex implementations previously presented in the literature. These results suggested that Holland's ideas could work even in a very simple framework, although ZCS performance was not optimal yet.

Optimal performance in different applications was finally reported in 1995 when Wilson [159] introduced the XCS classifier system. While XCS maintains Holland's essential ideas about classifier systems, it differs much from all the previous architectures. First in XCS Q-learning is used to distribute the reward to classifiers, instead of a bucket brigade algorithm.[3] Second, in XCS the genetic algorithm acts in environmental niches instead of on the whole population, as in the work of Booker on GOFER-1 [11]. Most important, in XCS the fitness of classifiers is based on the accuracy of classifier predictions instead of the prediction itself, a solution partially anticipated in the works of Frey and Slate [46] and Booker [11]. Wilson [159,161] showed that by using classifier accuracy as the fitness of the genetic algorithm, XCS is able to evolve classifiers that are (i) *accurate*, i.e., they give an accurate prediction of the expected reward, and (ii) *maximally general*, i.e., they match as many situations as possible without being overgeneral. For an analysis of the advantages of accuracy-based fitness in XCS, we refer the interested reader to Kovacs' paper in this volume [80]. For an overview of the main results reported in the literature since 1995, see Wilson's review of the state of XCS research in this book [164].

3 Basics

We now focus on the fundamental topics in classifier system research: the structure of classifiers, the credit allocation procedure, the discovery component, and the internal message list.

[2] Dorigo and Bersini [34] compared these two paradigm conceptually; but ZCS is one of the first learning classifier system implementations that used a Q-learning-like technique to assign the reward to classifiers. Another early use of a Q-Learning like technique was in CFSC2 [117], in which non-bucket brigade temporal difference methods were used to update the three kinds of strength used by the system to do its look-ahead planning. Other early papers discussing the relationships between temporal difference methods, Q-Learning and the bucket brigade include [138,151, 95].

[3] ZCS distributes the reward with an algorithm which is a mix of bucket brigade and Q-learning (QBB) while XCS uses an adaptation of standard Q-learning.

3.1 The Classifier Syntax

In most learning classifier systems the agent's detectors and effectors are represented by fixed length binary strings. Accordingly, classifier conditions are strings in the ternary alphabet {0,1,#} while classifier actions are binary strings.[4] This simple representation was initially introduced by Holland as a result of balancing the tradeoff between the descriptive power of the classifier language and computational efficiency [61].

Despite their elementary representation, Holland's classifier systems have been successfully used to tackle many problems as demonstrated by the many successful applications of classifier systems we review in Section 5, and those reported in the bibliography at the end of this book. On the other hand many authors have raised some concerns about the representational capabilities of the binary encoding.

Schuurmans and Schaeffer [126] showed that binary encoding may lead to unsatisfactory performance. In particular they observed that binary encoding (i) *unavoidably* introduces a bias in the generalization capabilities of the system making some concepts easy to generalize and others almost impossible to generalize; (ii) it has a limited capability to represent disjunctions and relations among sensors; finally (iii) it has a positional semantics which may not be able to capture certain relations among sensors that are positionally independent. Later Shu and Shaeffer [130] proposed an extension to the ternary representation of conditions in which named variables were used to describe relations among inputs bits.

Booker [13] analyzed the issues raised in [126] and suggested that these limitations are mainly to due to the encoding scheme (i.e., the way the sensors are encoded into bitstrings) rather than to the binary encoding itself. Booker [13] showed that *feature manifold* encodings, employed in pattern classification [55], can improve the representational capabilities of classifiers *without* modifying the classifier syntax. In particular Booker reports a number of examples in which the use feature manifolds avoid most of the problems discussed by Schuurmans and Schaeffer [126]. However, to our knowledge no one has yet implemented a classifier system which uses a feature manifold approach.

Sen [128] focused on the problem of selecting the most adequate classifier syntax to get the minimal sized (i.e., most general) representation of the target concept. Sen introduced a *value* representation in which the classifier condition are conjunctions of elementary terms. Each term is of the form "$x = c$" where x represents the (non-binary) input of a certain sensor, c represents an admissible value for the sensor x. Value representation is compared with binary encoding and some guidelines for selecting the most appropriate classifier syntax for a given problem are presented.

Collard and Escazut [21] focused on the problem of representing equality/inequality relations between bits in the sensory inputs. They introduced the

[4] In Holland's original descriptions of classifier systems [62,60], and in CFSC1, Riolo's implementation [113,121], the action parts of rules also include the "#" character, acting as a "pass through" operator.

idea of *relational schemata* (R-Schemata) which can represent relations among different sensor bits. They showed that relational schemata are more powerful than standard schemata (P-Schemata) although they still cannot represent any the relations among bits within the same condition. Accordingly they extended R-Schemata into RP-Schemata and compared their approach with default hierachies [116]. Finally they applied an implementation of RP-Schemata to the problem of learning two Boolean functions showing that it performs better than standard representation, although not optimally.

Separately, Wilson [157] suggested another way to increase the representational capabilities of classifier systems would be to replace the traditional trinary-based matching semantics with predicates in the real domain, for example: "$(0.0 < x < 0.1) \wedge (0.8 < y < 0.9)$." In this initial work classifier conditions represent these intervals by storing the center of a receptive field and its spread, both binary encoded. In addition, the classifier actions are scalar weights, and the system output is defined to be a linear combination of the output of classifiers that match the current sensory inputs. Wilson validates his proposal experimentally on two small problems regarding the learning of simple real functions.

Wilson [163, this book] has recently extended his original ideas [157] by adding an interval based representation to XCS [159]. As in his original proposal, classifier conditions are predicates over continuous-valued input, but in this case intervals are identified by their lower/upper bounds (i.e., making it easier to change the intervals asymmetrically) and are represented by real numbers instead of being binary encoded. Furthermore, classifier actions are discrete valued and *are not* combined [157]. In a system called XCSR, Wilson [162] applied XCS with continuous-valued inputs to the problem of learning a real version of the boolean multiplexer reporting very interesting results. (For further discussions, see Wilson's paper on XCSR in this volume [163].)

Lanzi [89,90] introduced two other representations for classifier conditions and implemented them on the XCS classifier systems. The first is the so-called "*messy*" representation [89], which was inspired by the work of Goldberg *etal.* on messy genetic algorithms [48], and which also bears some resemblance to Sen's variable representation [128]. The second representation is inspired by Koza's work on Genetic Programming [82] and uses a subset of Lisp S-expressions to represent conditions over Boolean and symbolic sensory inputs. Lanzi applied both representations to a set of known test problems showing that these two versions of XCS, namely XCSm and XCSL, reach optimal performance.

Ahluwalia and Bull [1] coupled binary conditions with symbolic actions represented by general S-expressions. They applied the new system, GP-CS, to extract features from data before these are classified with the K-nearest-neighbor method.

Finally, there has been considerable work on the *fuzzy* classifier systems, which blend the representational approach of fuzzy logic with rule-based classifier systems using evolutionary algorithms to evolve the parameters of various fuzzy matching and output functions. For a review, see [7] in this volume.

3.2 The Credit Assignment and Discovery Components

The performance of a specific learning classifier system depends on two main factors: (i) how the environmental reward is distributed to classifiers, i.e., the credit assignment procedure; and (ii) what values bias the evolution of classifiers in the population, i.e., the fitness of the genetic algorithm.[5] In this section we will focus on work involving those two factors. Note, however, that most classifier systems also use learning heuristics besides the genetic algorithm. These heuristics range from versions of the common cover detector message operator (which produces rules that match initially unmatched input messages) to special heuristics designed for a particular domain to which the classifier system is being applied.

In Holland's classifier systems the *bucket brigade* algorithm is used to update the *strength* parameter of classifiers which is also used as the fitness of the genetic algorithm [60]. It is well known that Holland's model exhibits a number of problems which sometimes prevent the system from achieving satisfactory performance [165]. In the past ten years many improvements to Holland's framework were suggested which, in some cases, consisted in a modified credit assignment procedure (e.g., [95]) or, in other cases, in a modified discovery component (e.g., [32]), or a modification of both (e.g., [161]).

It has long been recognized that when discussing the behavior of a learning classifier system it is difficult to separate the allocation of credit from the discovery component. For instance Riolo [115] showed that to perform sequential behavior a classifier system should allocate *strength* properly while preventing the proliferation of high rewarded classifiers. Furthermore Riolo's results suggested that the discovery algorithms should focus the search on the part of the problem that are not managed instead of focusing on the highest strength rules. While in some papers published over the past ten years the emphasis may appear to be on either the allocation of credit or the evolutionary components of a classifier system, in fact it is the need to coordinate these two components that has led to most of the recent experimentation with alternative classifier system architectures.

Booker's GOFER-1 [11] uses a niched genetic algorithms in which the classifier fitness depends on non-payoff as well as payoff information. In GOFER-1 a classifier's utility is a combination of three factors: (i) *impact*, an estimate of the classifier local payoff; (ii) *consistency*, an estimate of the error affecting the classifier prediction; and (iii) *match score*, an estimate of classifier specificity. Booker applied GOFER-1 both to one-step and multi-step problems [12], reporting very good results.

Huang [72] analyzed the performance of bucket brigade in a small toy problem. His results showed the approaches based on bucket brigade can correctly estimate the utility of classifiers that are applied in one context while they can fail when classifiers are applied in different contexts. Huang introduced a version

[5] Besides the question of what classifier attribute should be used as fitness by the evolutionary rule discovery algorithm, there is the additional issue of when the genetic algorithm should be applied, but we will largely ignore that issue in this chapter.

of bucket brigade sensitive to the activation context and tested it on the same toy problem. Liepins *etal.* [96] analyzed the possible alternative credit assignment procedures; then Liepins and Wang [95] introduced a hybrid bucket brigade-backward averaging (BB-BA) algorithm with the aim of combining the two credit allocation schemes proposed by Holland [64,59]. Twardoski [147] considered four of possible credit assignment strategies proposed until 1993 (backward averaging [64], bucket brigade [59], hybrid BB-BA [95], and Q-learning [152]) and presented an empirical evaluation of them on the pole-balancing problem. The results reported showed that Liepins and Wong's hybrid solution learned twice as fast as the other methods. In NEWBOOLE Bonelli et al. [9] introduced a "symmetrical payoff-penalty" procedure to distribute the reward to classifiers; though NEWBOOLE itself is limited to one-step problems, the credit allocation mechanism may be more generally applicable. Wilson [158] compared bucket brigade with a mixed technique of bucket brigade and Q-learning (namely QBB) on a grid problem with ZCS. The two methods performed almost identically and none of them achieved optimal performance. Wilson noted that the unsatisfactory performance of ZCS, both with bucket brigade and QBB, was mainly due to the proliferation of overgeneral classifiers. Later Cliff & Ross [20] confirmed Wilson's results with experiments on more complex problems.

Separately from the work on allocation of credit procedures, other researchers have studied the behavior of the discovery component of classifier systems. Horn, Goldberg, and Deb [71] observed that classifier systems usually face an implicitly multiobjective task. Accordingly, they decomposed a simple classifier system and applied previous experience on multiobjective optimization to analyze the behavior of the discovery component. In particular, they focused on niching in classifier systems, highlighting the correspondence between implicit niching, which in classifier systems is induced by the sharing of resources among classifiers, and explicit fitness sharing in some genetic algorithms [50]. Horn, Goldberg, and Deb [71] showed that classifier systems are able to maintain high quality and diverse niches virtually indefinitely. Later, Horn and Goldberg [70] extended that analysis, developing an analytical model to study niching in classifier systems. The model was used to calculate approximated upper and lower bounds on the niche existence time. Dorigo [32], with a more "hands-on" approach, defined a genetic operator *mutespec* to get rid of overgeneral classifiers. He also introduced the concept of *energy* of a classifier system (defined as the sum of classifiers strength) and showed that *energy* can be an effective way to trigger the activation of the genetic algorithm.

Until 1995 the analysis classifier system performance was focused mainly on one specific component, i.e., either the credit assignment *or* the genetic algorithm. However, this separation did not reflect a logical separation within the classifier system architecture. In fact, in all the works discussed so far, the strength parameter was updated through the credit assignment algorithm, and it also was the fitness used by the genetic algorithm.

In 1995 Wilson [159] introduced another approach to classifier systems, XCS, in which the allocation of credit for achieving *reward* is logically separated from

the discovery component. In XCS the fitness of classifiers *does not* depend on the prediction of reward given by the classifier (i.e., the *strength* parameter in Holland's framework) but instead fitness depends on the *accuracy* of classifier predictions. In addition, Q-Learning is used in XCS to distribute the reward among classifiers.

Wilson's choice of an accuracy-based fitness was partly anticipated by the early works of Holland [58], in which it was suggested that classifier fitness might be based on the *consistency* of classifier prediction, and in the experiments of Frey and Slate [46] who successfully applied an accuracy-based classifier system to the letter recognition problem. Wilson's intuition was that prediction should estimate how much reward might result from a certain action, but that the evolution learning should be focused on most reliable classifiers, i.e., classifiers that give a more precise (accurate) prediction. This is in contrast to the traditional approach, in which evolutionary learning was focused on those rules that predicted the most reward, with the implicit idea that the predicted reward would be an *expectation*, i.e., it would include both magnitude (value) and probability (accuracy) of reward. Wilson showed that with accuracy-based fitness XCS can reach *optimal* performance while evolving a minimal population of maximally general and accurate classifiers. For an overview of the results of XCS research we refer the interested reader to Wilson's paper in this volume [164].

Recently, Smith et al. [133] used a classifier system for acquiring novel fighter aircraft maneuvers. In their system, credit is assigned only once after the end of an engagement to all the classifiers which were active during the air combat engagement. This is similar to the profit-sharing approach first introduced by [64] and later used by [52], among others. Noticeably, the procedure *does not* update classifier parameters but instead replaces them with new values; the genetic algorithm is applied at fixed intervals and in some cases can replace the whole population. Smith et al. [133] results demonstrate that the applicability of classifier systems is not limited to reinforcement learning and thus they should be tried in more diverse fields, e.g., the discovery of novel behaviors (see [131, this book] for further details). Booker has developed a new learning classifier system architecture in which there is not an explicit measure of classifiers utilities, but instead *endogenous fitness* is used (see [14, this book] for further details). Finally, Westerdale has done much work on the formal analysis of various standard and non-standard credit allocation schemes; see [153] for recent results and references to earlier work.

3.3 Internal Memory

The learning capabilities of an adaptive agent relies on and is constrained by the way the agent perceives the environment. On the one hand, there are cases in which the agent's sensory inputs provide sufficient information to let it decide in every situation what is the best action to take. However, in many applications the agent's sensory inputs provide only partial information about the actual state of the environment, in which case the agent cannot decide what is the best action solely looking at its current sensory inputs. In the former case we say that

the environment is *Markovian* with respect to the agent's sensors; in the latter case we say that the environment is *non-Markovian*.

The distinction between Markov and non-Markov environments is fundamental in reinforcement learning. If the environment is Markovian the agent can completely rely on its sensors, while if the environment is non-Markovian the agent needs some sort of "memory" of previous experience to cope with the partial information coming from its sensors. This distinction also draws a separation between classifier systems and other learning systems which use traditional reinforcement learning algorithms.

Holland's classifier systems represent information about the structure of the world in the form of rules (which represent transitions between states) and messages on an *internal message list*. The system can use the message list to store information about (a) the current state of the world (in the form of detector messages), and (b) about previous states (in the form of messages from classifiers which were active at previous time steps). Because classifier system rules can represent knowledge of the world in the form of *default hierarchies*, homomorphisms, *quasi-homomorphisms* and other abstract structures [62], classifier systems are able to represent true *models* of the world (vs. simple isomorphic maps of the world). This is in contrast with the architectures for representing knowledge found in many studies of reinforcement learning algorithms such as Q-learning. For example, the systems used in [152] and TD(λ) [138] are memoryless. Thus classifier systems are in principle more powerful than many simple architectures used with Q-learning and TD(λ), since classifier systems can learn optimal behaviors in non-Markovian environments while simple map structures using Q-learning and TD(λ) can learn optimal behaviors only if the environment is Markovian.[6]

The internal message list is one of the most intriguing characteristics of learning classifier systems and it has been widely studied in the literature. Riolo and Robertson [121] reported limited success in applying a version of Holland classifier system to solve a non-Markov letter sequence prediction task. Smith [132] reported unsatisfactory performance in very basic non-Markov environments with Holland's classifier system. He analyzed the behavior of the system in those environments and identified a number of problems which prevented it from achieving optimal performance. Colombetti and Dorigo [22] successfully used one bit of memory (the *internal state*) to coordinate two classifier systems to solve a non-Markov task. In this case the effect of internal memory on the agent's behavior was decided at design time (*hardwired*) and not learned as in the previous approaches mentioned above.

Wilson's ZCS [158] has no internal memory, therefore its learning capabilities are limited to Markov environments. However a *small* bit register can be added to ZCS in order to tackle non-Markov problems [158]. Cliff and Ross [20] applied ZCS with internal memory, ZCSM, to simple non-Markov environments repor-

[6] Of course Q-Learning, back-propagation and other temporal difference methods are also applied to neural network representations of the world, which are provably equal to classifier systems in their representational and computational power.

ting promising (although non-optimal) performance. In a similar way, Lanzi [86] added internal memory to Wilson's XCS classifier system [159] which, likewise ZCS, does not have internal memory. Lanzi applied XCS with internal memory, XCSM, to simple non-Markov environments reporting optimal performance in most cases [86,87]. Recently Lanzi and Wilson [92] extended these results by applying XCSM to more difficult non-Markov problems. Their experiments show that XCSM can learn optimal solutions in complex non-Markov environments suggesting that the approach scales up. Barry [5,6] took a subset of non-Markov problems in which the same action, for the same sensory input, receives different rewards. He called this phenomenon "*Consecutive State Problem*" and introduced a solution for this particular problem that is simpler than internal memory. The solution consists of updating the classifier prediction parameter if the action has caused a change in the sensory inputs. This solution is validated experimentally.

A different approach to tackling non-Markov problems consists of using *corporations of classifiers* instead of internal memory [165]. This approach can be viewed as mixing the Holland and Pitt approaches to the organization of classifier systems, with respect to credit allocation, activation and rule discovery, in order to solve a number of problems that have plagued classifier systems. A corporations is a cluster of classifiers within the population that are treated as an indivisible entity during the discovery phase and/or the performance phase. Classifiers within a corporation are *explicitly* linked to each other to form a *chain*. When a classifier of a certain chain (i.e., corporation) is activated it forces the activation of classifiers that follows it in the chain. Tomlison and Bull reported some success in applying first a corporate version of ZCS [144,143] and later a corporate version of XCS [145] on a set non-Markov problems. For further discussion on classifier corporations we refer the interested reader to Tomlison and Bull paper in this volume [146].

The need to bias activity so that classifiers in a chain fire in sequence with high probability was also shown to be important in systems that *implicitly* linked classifiers, through the posting of "tagged" messages on the internal message list [121,115]. These tagged messages may be thought of as a memory of previous states, which can then be used later to disambiguate a non-Markovian state. This view is also seen in Smith [132], which showed with an indirect experiment that the sequential activations of classifiers creates a remembered context for later classifiers in the chain. This context helps these classifiers to cope with the partial information coming from the sensory inputs.

4 Advanced Topics

We now focus on some of the most important advanced topics of learning classifier system research: generalization, long action chains, planning, and scalability.

4.1 Generalization

An intriguing and important characteristic of learning classifier systems is their ability to *generalize* over their experiences so as to produce a compact representation of a certain learning task, i.e., to produce a true *model* of the environment. Generalization also draws a distinction between learning classifier systems and and many systems which use reinforcement learning algorithms (e.g., TD(λ) and Q-learning) to update values attached to states and transitions stored in a "pure," tabular format, i.e., isomorphic "models" which do not involve generalization.

Of course, generalization is a matter of representation: the more powerful the language used to express classifier conditions, the more varied and often better the possible generalizations which can be expressed in it. Accordingly, in the early '90s most of the research on generalization focused on the representation issue: how to define a language capable of representing adequate generalizations in a given domain; see Section 3.1 for an overview of some of this work.

On the other hand, correct generalization is also a matter of "evolution", i.e., what kind of generalizations are *likely* to emerge, given the interaction of a classifier system's particular credit allocation and rule discovery algorithms. In the early years it was shown that, under adequate assumptions, classifier systems can evolve hierarchies of concepts called *default hierarchies* [62]. Default hierarchies represent a target concept by a multilevel structure in which top-level classifiers represent general concepts while bottom-level classifiers represent more specific concepts. One nice characteristic of default hierarchies is that they can build a *quasi-homomorphic* model of the problem being learned which may be more compact that a strict *homomorphic* model [62,63].

Riolo [112] showed that classifier systems can maintain default hierarchies once they are formed. Riolo extended this analysis in [114] and [118], where he studied how default hierarchies can emerge from scratch. While the results described in those papers showed that default hierarchies did initially emerge and dominate the rule population, after additional evolution the default hierarchies were in effect filled in with more specific rules which were able to perform better when in conflict with the generalist rules in the original default hierarchies. As a result the classifier system would eventually build a homomorphic model of the environment, confirming early hypotheses made in [112,156].

In any event, default hierachies have been shown to be very useful ways to represent knowledge in classifier systems. For example, Holyoak [69] showed that classifier systems with default hierarchies could replicate a wide variety of phenomena observed in classical conditioning. Later, Riolo [117] applied a classifier system with a cooperative bucket brigade to the problem of modeling human performance on a simple medical diagnosis task. His results show that a small default hierarchy can replicate human behavior better than a set of "*flat*", i.e., non hierarchical, set of classifiers. In addition, Riolo's system demonstrates interesting generalization capability by predicting correct results not predicted with previous approaches. Smith and Goldberg [135] (see also [134]) analyze the impact of specificity-biased credit allocation and conflict resolution schemes on

the evolution of default hierarchies. Their analysis suggested that these schemes cannot be expected to perform adequately in an arbitrary classifier system. They suggested associating two factors with each classifier (a payoff estimate and a priority factor) in place of the original strength, coupled with an alternate credit allocation/conflict resolution scheme. Their results showed that the new scheme allows the exploitation of a broader class of default hierarchies.

Apart from the results concerning default hierarchies, most of the results reported in the literature for generalization with classifier systems (e.g., [115] or [158]) showed a proliferation of overgeneral classifiers. With XCS Wilson [159] reported a first set of results showing that XCS is able to produce correct generalizations over a certain problem. Later [160] (published also in [161]) Wilson augmented the generalization capabilities of XCS by (i) moving the genetic algorithm from the match set to the action set and (ii) introducing a *subsumption deletion* operator. The results reported by Wilson and extended by Kovacs [77, 78] suggested Kovacs's *Optimality Hypothesis* which states that XCS may be able to evolve the *minimal* representation of a target concept. Kovacs validated his hypothesis experimentally on very hard learning problems. Recently, Kovacs [79] studied the impact of different deletion policies with respect to the generalization issue, testing some modifications to Wilson's original XCS deletion policy.

Note that Wilson's XCS tends to build a complete mapping from sensation-action pairs to payoff predictions [159], i.e., a homomorphic model of the solution, whereas earlier classifier systems have the tendency to develop quasi-homomorphic models [62,112,117]. However, in some cases the more homomorphic models of XCS also tend to be more accurate, though less compact, than equivalent quasi-homomorphic models.

The discussion concerning the type of generalizations that classifier systems should built appears to be open and lively still, as suggested by Holland's and Wilson's contributions to the paper "What is a learning classifier system?" in this volume.

4.2 Long Action Chains and Planning

In order to learn complex behaviors an agent must be able to develop long sequences of actions, i.e., long *action chains*. In addition, one criteria for good classifier system performance is that the agent should be able to learn such action chains receiving only a little information from the environment, i.e., being rewarded in only few situations, leading to long delays in rewards between some *stage setting* actions and the ultimate reward those actions lead to. Unfortunately, learning classifier systems tend to perform poorly when the information available from the environment is scarce, i.e., when there are only a few rewarded situations. Wilson and Goldberg [165] observed that up to 1989 only three cases were reported in which classifier systems developed long sequences of actions: [155,121] and [52, the Pittsburg Approach].

In the mid 90's Dorigo and Colombetti reported some interesting results showing that learning classifier systems can be used to train physical autonomous agents to perform relatively complex sequential tasks (see [22,36]). However,

these results were mainly achieved with hierarchies of interconnected, but distinct classifier systems (see Section 4.3). Moreover in most of these applications a lot of information about the task was given, i.e., the agent was rewarded for each action taken, so the central problem of learning from only delayed rewards was largely sidestepped.

Taking inspiration from animal behavior, Booker [12] modified his GOFER-1 classifier system [11] so that actions persist until no other classifiers advocate it or the action is rewarded. Booker reported interesting results applying this variant of GOFER-1 to a relatively complex navigation task [139]. Riolo [115] studied the emergence of coupled sequence of classifiers, highlighting a number of features that improve the emergence of action chains (as described above, in the context of corporate classifier systems).

Wilson's ZCS can develop short action chains but, as showed by Cliff and Ross [20], it cannot learn long sequences of actions because of the proliferation of overgeneral classifiers. To improve these results corporations [165] can be added to ZCS [143,145] but the benefits are limited.

The problems that afflicted ZCS appear to be solved in XCS. In [159] Wilson showed that XCS can perform optimally in relatively complex sequential problems. Lanzi [88] extended these results showing that XCS can reach optimal performance in more complex sequential problems in which ZCS failed.

To improve the learning of complex behaviors an *internal* model of the environment can be developed [139]. This is the approach outlined by Holland in many earlier works, e.g., in [62] and [60]. The classifier system's model can be used for learning with little additional experience from the *external* environment, and it can be used for planning. Sutton [139] took this approach in Dyna-Q, a system which used a Q-learning algorithm on a isomorphic (non-generalized) representation of the world; his results showed that the use of the internal model leads to impressive improvements in the performance over other Q-learning approaches to learning sequential tasks.

Riolo [117] was the first to apply this approach to learning classifier systems. His CFSC2 classifier system learns a model of the environment by means of state transitions that are represented by classifiers. For this purpose, some loci of standard trinary classifier conditions and actions are used to represent special "tags." These tags are used to distinguish the classifiers (and the messages they post) which represent (i) states in the world and (ii) transitions from states to states, as a result of the systems actions. In addition, the tags are used to distinguish between messages that are intended to cause the system to actually take actions in the world, from other messages that are intended to represent planning, in effect "imagining" what would happen if certain actions were taken. In addition, each classifier has three measures of utility associated with it: (i) a long term strength analogous to the usual *strength* in Holland's system; (ii) a short term strength which is updated *every time step*, even when the system is not executing actions in the environment (i.e., it is executing rules that implement its planning activity); and (iii) an estimate of the predictive accuracy of the transition that the classifier represents. The credit assignment procedure also was modified to

cope with these three parameters, using a Q-learning type algorithm. There is no genetic algorithm, and no generalization: the classifiers in effect implemented an isomorphic model of the environment. Riolo tested CFSC2 on a relatively complex sequential problem also employed by Sutton [139] and Booker [12]. His results confirm early results from systems using Q-learning, showing that a classifier system which uses an internal model of the environment to do *lookahead planning* can speed up learning considerably, as well as make it possible for the systems to do *latent learning*, a kind of learning not possible in systems, artificial or biological, without internal models which predict what *state* follows from a given state-action pair. Furthermore, Riolo suggested that because of their generalization capabilities, classifiers can be used to build models of the environment that are more compact than those built by Dyna-Q, which assumed a tabular representations of its model.

Roberts [120] extended his version of Wilson's *Animat* [119] adding to each classifier a *followset*, which contains a record of situations and rewards that followed the activation of the classifier. In particular, each followset has an original time stamp along with a set of events, each of which consists of either (i) a time stamp and a sensory configuration, or (ii) a time stamp and a reward value. Roberts compared his basic implementation of Wilson's Animat to one enriched with followsets on two simple environments; his results showed that there was no significant difference in the performance the two systems.

Recently, Stolzmann has introduced the idea of *Anticipatory Classifier Systems*, ACS, in which classifiers are enriched with an *expectation* of the state that will follow the activation of the classifier, similar to (but richer than) the predictions made in Riolo's CFSC2. For a complete discussion of the characteristics of Anticipatory Classifier Systems and the main results achieved so far, we refer the interested reader to Stolzmann's introductory paper [136] and to Butz and Stolzmann paper [137] for the robotics applications of anticipatory classifier systems, both in this volume.

4.3 Scalability

One of the main issues when considering a learning paradigm is that of *scalability*, i.e., how rapidly does the learning time or system size grow as the problem complexity grows?

Unfortunately it is very difficult to answer this question for learning classifier systems since their evolutionary component does not allow any evaluation of the computational complexity of this approach. In fact, as far as we know, only a few people have discussed this issue. In particular, Wilson [161] has applied XCS to a set of multiplexer problems of increasing difficulty (the 6-, 11-, an 20-multiplexer). His results suggest that in XCS the complexity of the learning process grows as a *low order polynomial* of the complexity of the problem, *not* with the complexity of the learning space as in other learning techniques (e.g. neural networks and nearest-neighbor approaches). Wilson's results were recently extended to the case when internal memory is added to XCS. Lanzi and Wilson [93] applied XCS with internal memory, XCSM, to a set of non-Markov

problems of increasing complexity showing that, as hypothesized by Wilson [161], the population size and the learning time did not grow as a function of the search space, but instead roughly as a function of the complexity of the problem.

Even though it might be possible to prove that in classifier systems the learning time grows with the problem difficulty and not with the size of the search space, a question yet remains: is it feasible to design a *unique* learning classifier systems which solves an *arbitrarily difficult* problem? For instance, is it feasible to train an arbitrarily complex robotic agent to perform a very difficult task with one learning classifier system?

Note that this is an important question which has implications not only for classifier systems but also for systems using reinforcement learning in general. One preliminary encouraging answer came out in the early days of modern learning classifier systems research by researchers who were mainly working with physical autonomous agents: Dorigo and Colombetti [36] and Donnart and Meyer [28]. In both these studies complex robotics problems are tackled by hierarchical architectures: Dorigo and Colombetti's ALECSYS uses a hierarchy of Holland's style classifier systems in which hierarchy is defined at design time while single modules are evolved; Donnart and Meyer's *MonaLysa* [28] has a hierarchical architecture in which a learning classifier system, implementing the reactive behavior layer, is coupled with other non-evolutionary components (e.g., the planning module).[7]

Interestingly, these works on hierarchical learning classifier systems in part anticipated the recent developments reported in reinforcement learning [27,141]. For instance, the *formal* decomposition method, MAXQ, proposed by Dietterich has some similarities with the *Behavior Analysis and Training* (BAT) methodology proposed by Dorigo and Colombetti [24].

5 Applications

In the following we present an overview of some interesting applications of learning classifier systems. For the sake of brevity we cannot discuss *all* the applications that have been presented in the literature since 1989. For the list of all papers that discuss classifier systems applications presented so far we refer the reader to Kovacs and Lanzi's learning classifier system bibliography at the end of this volume [81].

5.1 Autonomous Robotics

Autonomous robotics has always considered an important testbed for reinforcement learning algorithms. In effect robotics problems are usually difficult to describe using more traditional machine learning methods (e.g., supervised learning) while they are easily modeled as reinforcement learning problems.

Accordingly, autonomous robotics is probably the single research area where learning classifier systems have been applied most during the last ten years. The

[7] See Section 5.1 for more detail on the robotics applications.

most extensive research in the application of learning classifier systems to autonomous robotics was carried out by Dorigo and Colombetti [36] with other colleagues. In the years from 1990 until 1998 they applied different versions of their parallel learning classifier system, ALECSYS, to a different number of robotics applications. Borrowing the idea "*shaping*" from experimental psychology they introduced the concept of *Robot Shaping* defined as the incremental training of an autonomous agent. To apply this idea to real world problems they defined a behavior engineering methodology named BAT: Behavior Analysis and Training. An overview of their work can be found in Dorigo and Colombetti's book "Robot Shaping: An experiment in Behavior Engineering"; the technical details of their parallel classifier systems, ALECSYS, are discussed in [38,31,37,33]; discussions about Robot Shaping and BAT methodology can be found in [35,22,24, 23]; some applications are discussed in [104,25]. Donnart and Meyer developed a hierarchical architecture *MonaLysa* for controlling autonomous agents [28,30, 29]. MonaLysa integrates a reactive module implemented by a learning classifier system with other deliberative modules (e.g., the planning module). Many other applications to autonomous robotics have been carried out with *Fuzzy Learning Classifier Systems*. For a review for those works we refer the reader to Bonarini's review in this volume [7].

5.2 Supervised Classification

Another important research area where learning classifier systems have been applied is that of supervised classification. In this type of application, a series of examples which represent a target concept (phenomenon) is presented to the classifier system. The goal of the learning process is to develop a general and compact *model* of the target phenomenon that the examples represent. This model can be used to *analyze* the target phenomenon previously observed or to classify *unseen* examples. The applications of learning classifier systems in this area take two main approaches: that of cognitive modeling (e.g., Holland and Reitman [64], Riolo [117], and, recently, Hartley [54]); and that of data analysis and data mining (e.g., Bonelli et al. [8], Holmes [65,66,67,68], Guiu et al. [73, 74], Saxon and Barry [124, this volume]).

In the former case, learning classifier systems are used as a way of *modeling* human categorization process. Riolo [117] used a classifier system to model human performance on a simple deterministic categorization task; his classifier system is able to replicate data on human performance and also show some interesting generalization capabilities on unseen cases. Qualitatively similar results are presented by Sen [129] who also studied classifier systems performance on filtering and condensation tasks [85]. Hartley [53,54] compares NEWBOOLE [9] and XCS [159] on a categorization task. His results show that the different methods of fitness evaluation of these models (i.e., strength-based in NEWBOOLE, accuracy-based in XCS) have a significant impact on the knowledge the systems evolve, in particular that accuracy-based fitness produces a more complete model of the problem space (in Wilson's words [159] a *complete mapping*).

In data mining applications, learning classifier systems are used to extract *compact* descriptions of *interesting* phenomenon usually described by multidimensional data. Bonelli and Parodi [8] compared NEWBOOLE with other two learning algorithms, the rule inducer CN2 [18] and neural networks, on three medical domains showing that NEWBOOLE performed significantly better than the other methods. Sen [127] modified the conflict resolution strategy of NEWBOOLE to improve its performance on known machine learning datasets. The reported results show that this variant of NEWBOOLE outperforms many machine learning methods. Holmes [65,66,67,75] applied his version of NEWBOOLE, EpiCS, to classify epidemiologic surveillance data. The results of his research are discussed in Holmes' paper "Learning Classifier Systems Applied to Knowledge Discovery in Clinical Research Databases" in this volume [68]. Saxon and Barry studied the performance of XCS in a set of known machine learning problems, namely the MONK problems, demonstrating that XCS is able to produce a classification performance and rule set which exceeds the performance of most current Machine Learning techniques [142]. Their results are extensively discussed in their paper "XCS and the Monk's Problems" in this volume.

5.3 Other Applications

Apart the applications in autonomous robotics and data mining and analysis, there are many other specific problem domains where learning classifier systems have been applied. In this section we mention just a few of them. Note that these uses of classifier systems fall into two basic categories: (i) "engineering" applications, i.e., where the goal is to get a classifier system which can do a task as well as possible, with no concern about exactly how it does it, and (ii) "modeling" applications, where the goal is to construct a plausible model of some real system, in which the mechanisms used in the classifier system should plausibly capture some essential features of the system being modeled, and in which the classifier system generates behavior which should match that of the real system being modeled, independent of whether that behavior can be described as good or bad performance on some given task.

In *Computational Economics*, learning classifier systems have been used to model adaptive agents in artificial stock markets (see for example the works of Brian Arthur et al. [2] and Lebaron et al. [94]). Vriend [148,149,150] compared the performances of Michigan style and Pittsburg style classifier systems on the same economic model. Marimon *etal.* [98] have modeled the emergence of money using populations of classifier systems, each individual modeling the behavior of one economic agent. Bull [16] applied a variant of Wilson's ZCS in a simulated "Continuous Double-Auction" (CDA) market. Other papers discussing the use of classifier systems as a tool for modeling agents in economic systems include those by Miller and Holland [100] and by Mitlöhner [101].

Recent use of classifier systems to model various aspects of human cognitive systems include (i) the work of Satterfield [123] in which a classifier system is used to model humans learning languages in bilingual environments, enabling her to test alternative conceptual models of this process, (ii) the work of Druhan

and Mathews [39], in which a classifier system was used as a model of how humans learn (artificial) languages, and (iii) the work of Davis [26], who build a computation model of Affect Theory, which was able to generate behavior which matched known behaviors in humans, and which generated behavior which can be seen as predictions of novel phenomena which can be experimentally verified.

Other uses of classifier systems include the work of Federman *etal.* [41,42, 43,45], who have applied learning classifier systems to predict the next note in different type of music. Richards *etal.* [106,107,108,109,110,111] applied classifier systems to 2-D and 3-D shape optimization. In another shape related application, Nagasaka and Taura [103] generate shape features using classifier systems. Smith et al. [133] (see also [131, this volume]) applied learning classifier systems to the problem of discovering novel fighter maneuvering strategies. Sanza et al. [122] used an agent with a classifier system to support users who navigate in a virtual environment. Frey and Slate [46] applied a classifier system to a letter recognition problem; Cao et al. [17], Escazut and Fogarty [40] to traffic controllers; Gilbert *etal* [47] to the profit optimization of a batch chemical reaction.

6 Summary

The research in learning classifier systems is not limited to a specific research field as the review we presented here demonstrates. In fact learning classifier systems have been studied and applied in many different areas. With this paper we tried to give a map to the readers for seeing their way not only through the many papers which have been published since 1989, but also through the many inter-related research problems and issues which have been addressed in those (and preceding) years.

References

1. Manu Ahluwalia and Larry Bull. A Genetic Programming-based Classifier System. In Banzhaf et al. [4], pages 11–18.
2. W. Brian Arthur, John H. Holland, Blake LeBaron, Richard Palmer, and Paul Talyer. Asset Pricing Under Endogenous Expectations in an Artificial Stock Market. Technical report, Santa Fe Institute, 1996. This is the original version of LeBaron1999a.
3. Thomas Bäck, editor. *Proceedings of the 7th International Conference on Genetic Algorithms (ICGA97)*. Morgan Kaufmann: San Francisco CA, 1997.
4. Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*. Morgan Kaufmann: San Francisco, CA, 1999.
5. Alwyn Barry. Aliasing in XCS and the Consecutive State Problem: 1 – Effects. In Banzhaf et al. [4], pages 19–26.
6. Alwyn Barry. Aliasing in XCS and the Consecutive State Problem: 2 – Solutions. In Banzhaf et al. [4], pages 27–34.

7. Andrea Bonarini. An Introduction to Learning Fuzzy Classifier Systems. In Lanzi et al. [91], pages 83–104. (this volume).
8. Pierre Bonelli and Alexandre Parodi. An Efficient Classifier System and its Experimental Comparison with two Representative learning methods on three medical domains. In Booker and Belew [15], pages 288–295.
9. Pierre Bonelli, Alexandre Parodi, Sandip Sen, and Stewart W. Wilson. NEWBOOLE: A Fast GBML System. In *International Conference on Machine Learning*, pages 153–159, San Mateo, California, 1990. Morgan Kaufmann.
10. Lashon B. Booker. Classifier Systems that Learn Internal World Models. *Machine Learning*, 3:161–192, 1988.
11. Lashon B. Booker. Triggered rule discovery in classifier systems. In Schaffer [125], pages 265–274.
12. Lashon B. Booker. Instinct as an Inductive Bias for Learning Behavioral Sequences. In Meyer and Wilson [99], pages 230–237.
13. Lashon B. Booker. Representing Attribute-Based Concepts in a Classifier System. In Rawlins [105], pages 115–127.
14. Lashon B. Booker. Do We Really Need to Estimate Rule Utilities in Classifier Systems? In Lanzi et al. [91], pages 125–142. (this volume).
15. Lashon B. Booker and Richard K. Belew, editors. *Proceedings of the 4th International Conference on Genetic Algorithms (ICGA91)*. Morgan Kaufmann: San Francisco CA, July 1991.
16. Larry Bull. On using ZCS in a Simulated Continuous Double-Auction Market. In Banzhaf et al. [4], pages 83–90.
17. Y. J. Cao, N. Ireson, Larry Bull, and R. Miles. Design of a Traffic Junction Controller using a Classifier System and Fuzzy Logic. In *Proceedings of the Sixth International Conference on Computational Intelligence, Theory, and Applications*. Springer-Verlag, 1999.
18. P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3(4):261–283, 1989.
19. Dave Cliff, Philip Husbands, Jean-Arcady Meyer, and Stewart W. Wilson, editors. *From Animals to Animats 3. Proceedings of the Third International Conference on Simulation of Adaptive Behavior (SAB94)*. A Bradford Book. MIT Press, 1994.
20. Dave Cliff and Susi Ross. Adding Temporary Memory to ZCS. *Adaptive Behavior*, 3(2):101–150, 1995. Also technical report: ftp://ftp.cogs.susx.ac.uk/pub/reports/csrp/csrp347.ps.Z.
21. Philippe Collard and Cathy Escazut. Relational Schemata: A Way to Improve the Expressiveness of Classifiers. pages 397–404. Morgan kaufmann Publishers: San Francisco CA, 1995.
22. Marco Colombetti and Marco Dorigo. Training agents to perform sequential behavior. *Adaptive Behavior*, 2(3):247–275, 1994. ftp://iridia.ulb.ac.be/pub/dorigo/journals/IJ.06-ADAP94.ps.gz.
23. Marco Colombetti and Marco Dorigo. Evolutionary Computation in Behavior Engineering. In *Evolutionary Computation: Theory and Applications*, chapter 2, pages 37–80. World Scientific Publishing Co.: Singapore, 1999. Also Tech. Report. TR/IRIDIA/1996-1, IRIDIA, Université Libre de Bruxelles.
24. Marco Colombetti, Marco Dorigo, and G. Borghi. Behavior Analysis and Training: A Methodology for Behavior Engineering. *IEEE Transactions on Systems, Man and Cybernetics*, 26(6):365–380, 1996.

25. Marco Colombetti, Marco Dorigo, and G. Borghi. Robot shaping: The HAMSTER Experiment. In M. Jamshidi et al., editor, *Proceedings of ISRAM'96, Sixth International Symposium on Robotics and Manufacturing, May 28–30, Montpellier, France*, 1996.
26. Michael Sean Davis. *A Computational Model of Affect Theory: Simulations of Reducer/Augmenter and Learned Helplessness Phenomena*. PhD thesis, Department of Psychology, University of Michigan, 2000.
27. T.G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. Submitted for journal publication. Available at .
28. Jean-Yves Donnart and Jean-Arcady Meyer. A hierarchical classifier system implementing a motivationally autonomous animat. In Cliff et al. [19], pages 144–153.
29. Jean-Yves Donnart and Jean-Arcady Meyer. Hierarchical-map Building and Self-positioning with MonaLysa. *Adaptive Behavior*, 5(1):29–74, 1996.
30. Jean-Yves Donnart and Jean-Arcady Meyer. Spatial Exploration, Map Learning, and Self-Positioning with MonaLysa. In Maes et al. [97], pages 204–213.
31. Marco Dorigo. Using Transputers to Increase Speed and Flexibility of Genetic-based Machine Learning Systems. *Microprocessing and Microprogramming*, 34:147–152, 1991.
32. Marco Dorigo. Genetic and Non-Genetic Operators in ALECSYS. *Evolutionary Computation*, 1(2):151–164, 1993. Also Tech. Report TR-92-075 International Computer Science Institute.
33. Marco Dorigo. Alecsys and the AutonoMouse: Learning to Control a Real Robot by Distributed Classifier Systems. *Machine Learning*, 19:209–240, 1995.
34. Marco Dorigo and Hugues Bersini. A Comparison of Q-Learning and Classifier Systems. In Cliff et al. [19], pages 248–255.
35. Marco Dorigo and Marco Colombetti. Robot shaping: Developing autonomous agents through learning. *Artificial Intelligence*, 2:321–370, 1994. ftp://iridia.ulb.ac.be/pub/dorigo/journals/IJ.05-AIJ94.ps.gz.
36. Marco Dorigo and Marco Colombetti. *Robot Shaping: An Experiment in Behavior Engineering*. MIT Press/Bradford Books, 1998.
37. Marco Dorigo and U. Schnepf. Genetics-based Machine Learning and Behaviour Based Robotics: A New Synthesis. *IEEE Transactions on Systems, Man and Cybernetics*, 23(1):141–154, 1993.
38. Marco Dorigo and Enrico Sirtori. Alecsys: A Parallel Laboratory for Learning Classifier Systems. In Booker and Belew [15], pages 296–302.
39. Barry B. Druhan and Robert C. Mathews. THIYOS: A Classifier System Model of Implicit Knowledge in Artificial Grammars. In *Proc. Ann. Cog. Sci. Soc.*, 1989.
40. Cathy Escazut and Terence C. Fogarty. Coevolving Classifier Systems to Control Traffic Signals. In John R. Koza, editor, *Late Breaking Papers at the 1997 Genetic Programming Conference*, Stanford University, CA, USA, July 1997. Stanford Bookstore.
41. Francine Federman and Susan Fife Dorchak. Information Theory and NEXTPITCH: A Learning Classifier System. In Bäck [3], pages 442–449.
42. Francine Federman and Susan Fife Dorchak. Representation of Music in a Learning Classifier System. In Rad and Skowron, editors, *Foundations of Intelligent Systems: Proceedings 10th International Symposium (ISMIS'97)*. Springer-Verlag: Heidelberg, 1997.
43. Francine Federman and Susan Fife Dorchak. A Study of Classifier Length and Population Size. In Koza et al. [83], pages 629–634.

44. Stephanie Forrest, editor. *Proceedings of the 5th International Conference on Genetic Algorithms (ICGA93)*. Morgan Kaufmann, 1993.
45. Francine Federman and Gayle Sparkman and Stephanie Watt. Representation of Music in a Learning Classifier System Utilizing Bach Chorales. In Banzhaf et al. [4], page 785. One page poster paper.
46. Peter W. Frey and David J. Slate. Letter Recognition Using Holland-Style Adaptive Classifiers. *Machine Learning*, 6:161–182, 1991.
47. A. H. Gilbert, Frances Bell, and Christine L. Valenzuela. Adaptive Learning of Process Control and Profit Optimisation using a Classifier System. *Evolutionary Computation*, 3(2):177–198, 1995.
48. D. E. Goldberg. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3:493–530, 1989.
49. David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Mass., 1989.
50. D.E. Goldberg and J. Richardson. Genetic Algorithms with Sharing for Multimodal Function Optimization. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49, 1987.
51. John J. Grefenstette, editor. *Proceedings of the 1st International Conference on Genetic Algorithms and their Applications (ICGA85)*. Lawrence Erlbaum Associates: Pittsburgh, PA, July 1985.
52. John J. Grefenstette. Credit Assignment in Rule Discovery Systems Based on Genetic Algorithms. *Machine Learning*, 3:225–245, 1988.
53. Adrian Hartley. Genetics Based Machine Learning as a Model of Perceptual Category Learning in Humans. Master's thesis, University of Birmingham, 1998. ftp://ftp.cs.bham.ac.uk/pub/authors/T.Kovacs/index.html.
54. Adrian Hartley. Accuracy-based fitness allows similar performance to humans in static and dynamic classification environments. In Banzhaf et al. [4], pages 266–273.
55. R. Hayes-Roth. Patterns of induction and associated knowledge acquisition algorithms. In *Pattern Recognition and artificial intelligence*. New York: Academic Press, 1976.
56. John H. Holland. Processing and processors for schemata. In E. L. Jacks, editor, *Associative information processing*, pages 127–146. New York: American Elsevier, 1971.
57. John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975. Republished by the MIT press, 1992.
58. John H. Holland. Adaptation. In R. Rosen and F. M. Snell, editors, *Progress in theoretical biology*. New York: Plenum, 1976.
59. John H. Holland. Properties of the bucket brigade. In Grefenstette [51], pages 1–7.
60. John H. Holland. Escaping Brittleness: The possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems. In Mitchell, Michalski, and Carbonell, editors, *Machine learning, an artificial intelligence approach. Volume II*, chapter 20, pages 593–623. Morgan Kaufmann, 1986.
61. John H. Holland. *Hidden Order: How Adaptation Builds Complexity*. 1996.
62. John H. Holland, Keith J. Holyoak, Richard E. Nisbett, and P. R. Thagard. *Induction: Processes of Inference, Learning, and Discovery*. MIT Press, Cambridge, 1986.
63. John H. Holland, Keith J. Holyoak, Richard E. Nisbett, and Paul R. Thagard. Classifier Systems, Q-Morphisms, and Induction. Research Notes in Artificial Intelligence, pages 116–128. Pitman Publishing: London, 1989.

64. John H. Holland and J. S. Reitman. Cognitive systems based on adaptive algorithms. In D. A. Waterman and F. Hayes-Roth, editors, *Pattern-directed inference systems*. New York: Academic Press, 1978. Reprinted in: Evolutionary Computation. The Fossil Record. David B. Fogel (Ed.) IEEE Press, 1998. ISBN: 0-7803-3481-7.
65. John H. Holmes. *Evolution-Assisted Discovery of Sentinel Features in Epidemiologic Surveillance*. PhD thesis, Drexel Univesity, 1996. http://cceb.med.upenn.edu/holmes/disstxt.ps.gz.
66. John H. Holmes. Discovering Risk of Disease with a Learning Classifier System. In Bäck [3]. http://cceb.med.upenn.edu/holmes/icga97.ps.gz.
67. John H. Holmes. Differential negative reinforcement improves classifier system learning rate in two-class problems with unequal base rates. In Koza et al. [83], pages 635–642. http://cceb.med.upenn.edu/holmes/gp98.ps.gz.
68. John H. Holmes. Learning Classifier Systems Applied to Knowledge Discovery in Clinical Research Databases. In Lanzi et al. [91], pages 243–264. (this volume).
69. Keith J. Holyoak, K. Koh, and Richard E. Nisbett. A Theory of Conditioning: Inductive Learning within Rule-Based Default Hierarchies. *Psych. Review*, 96:315–340, 1990.
70. Jeffrey Horn and David E. Goldberg. Natural Niching for Cooperative Learning in Classifier Systems. In Koza et al. [84], pages 553–564.
71. Jeffrey Horn, David E. Goldberg, and Kalyanmoy Deb. Implicit Niching in a Learning Classifier System: Nature's Way. *Evolutionary Computation*, 2(1):37–66, 1994. Also IlliGAL Report No 94001, 1994.
72. Dijia Huang. The Context-Array Bucket-Brigade Algorithm: An Enhanced Approach to Credit-Apportionment in Classifier Systems. In Schaffer [125], pages 311–316.
73. Josep Maria Garrell i Guiu, Elisabet Golobardes i Ribé, Ester Bernadó i Mansilla, and Francesc Xavier Llorà i Fàbrega. Automatic Classification of mammary biopsy images with machine learning techniques. In E. Alpaydin, editor, *Proceedings of Engineering of Intelligent Systems (EIS'98)*, volume 3, pages 411–418. ICSC Academic Press, 1998. http://www.salleurl.edu/~xevil/Work/index.html.
74. Josep Maria Garrell i Guiu, Elisabet Golobardes i Ribé, Ester Bernadó i Mansilla, and Francesc Xavier Llorà i Fàbrega. Automatic Diagnosis with Genetic Algorithms and Case-Based Reasoning. *To appear in AIENG Journal*, 1999. (This is an expanded version of Guiu98a).
75. John H. Holmes. Evaluating Learning Classifier System Performance In Two-Choice Decision Tasks: An LCS Metric Toolkit. In Banzhaf et al. [4], page 789. One page poster paper.
76. Leslie Pack Kaelbling, Michael L. Littman, and Andew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4, 1996.
77. Tim Kovacs. Evolving Optimal Populations with XCS Classifier Systems. Master's thesis, School of Computer Science, University of Birmingham, Birmingham, U.K., 1996. Also tech. report CSR-96-17 and CSRP-96-17 ftp://ftp.cs.bham.ac.uk/pub/tech-reports/1996/CSRP-96-17.ps.gz.
78. Tim Kovacs. XCS Classifier System Reliably Evolves Accurate, Complete, and Minimal Representations for Boolean Functions. In Roy, Chawdhry, and Pant, editors, *Soft Computing in Engineering Design and Manufacturing*, pages 59–68. Springer-Verlag, London, 1997. ftp://ftp.cs.bham.ac.uk/pub/authors/T.Kovacs/index.html.

79. Tim Kovacs. Deletion schemes for classifier systems. In Banzhaf et al. [4], pages 329–336. Also technical report CSRP-99-08, School of Computer Science, University of Birmingham. http://www.cs.bham.ac.uk/~tyk.
80. Tim Kovacs. Strength or Accuracy? Fitness calculation in learning classifier systems. In Lanzi et al. [91], pages 143–160. (this volume).
81. Tim Kovacs and Pier Luca Lanzi. A Learning Classifier Systems Bibliography. In Lanzi et al. [91], pages 323–350. (this volume).
82. John Koza. *Genetic Programming*. MIT Press, 1992.
83. John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors. *Genetic Programming 1998: Proceedings of the Third Annual Conference*. Morgan Kaufmann: San Francisco, CA, 1998.
84. John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors. *Genetic Programming 1996: Proceedings of the First Annual Conference*, Stanford University, CA, USA, 1996. MIT Press.
85. J. Kruschke. ALCOVE: An exemplar-based connectionist model of category learning. *Psychology Review*, 99:22–44, 1992.
86. Pier Luca Lanzi. Adding Memory to XCS. In *Proceedings of the IEEE Conference on Evolutionary Computation (ICEC98)*. IEEE Press, 1998. http://ftp.elet.polimi.it/people/lanzi/icec98.ps.gz.
87. Pier Luca Lanzi. An analysis of the memory mechanism of XCSM. In Koza et al. [83], pages 643–651. http://ftp.elet.polimi.it/people/lanzi/gp98.ps.gz.
88. Pier Luca Lanzi. An Analysis of Generalization in the XCS Classifier System. *Evolutionary Computation*, 7(2):125–149, 1999.
89. Pier Luca Lanzi. Extending the Representation of Classifier Conditions Part I: From Binary to Messy Coding. In Banzhaf et al. [4], pages 337–344.
90. Pier Luca Lanzi. Extending the Representation of Classifier Conditions Part II: From Messy Coding to S-Expressions. In Banzhaf et al. [4], pages 345–352.
91. Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors. *Learning Classifier Systems: An Introduction to Contemporary Research*, volume 1813 of *LNAI*. Springer-Verlag, Berlin, 2000.
92. Pier Luca Lanzi and Stewart W. Wilson. Optimal classifier system performance in non-Markov environments. Technical Report 99.36, Dipartimento di Elettronica e Informazione - Politecnico di Milano, 1999. Also IlliGAL tech. report 99022, University of Illinois.
93. P.L. Lanzi and S. W. Wilson. Toward optimal classifier system performance in non-Markov environments. *Evolutionary Computation*, 2000. to appear.
94. Blake Lebaron, W. Brian Arthur, and R. Palmer. The Time Series Properties of an Artificial Stock Market. *Journal of Economic Dynamics and Control*, 1999.
95. Gunar E. Liepins, M. R. Hillard, M. Palmer, and G. Rangarajan. Credit Assignment and Discovery in Classifier Systems. *International Journal of Intelligent Systems*, 6:55–69, 1991.
96. Gunar E. Liepins, Michael R. Hilliard, Mark Palmer, and Gita Rangarajan. Alternatives for Classifier System Credit Assignment. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, pages 756–761, 1989.
97. Pattie Maes, Maja J. Mataric, Jean-Arcady Meyer, Jordan Pollack, and Stewart W. Wilson, editors. *From Animals to Animats 4. Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior (SAB96)*. A Bradford Book. MIT Press, 1996.

98. Ramon Marimon, Ellen McGrattan, and Thomas J. Sargent. Money as a Medium of Exchange in an Economy with Artificially Intelligent Agents. *Journal of Economic Dynamics and Control*, 14:329–373, 1990. Also Tech. Report 89-004, Santa Fe Institute, 1989.
99. J. A. Meyer and S. W. Wilson, editors. *From Animals to Animats 1. Proceedings of the First International Conference on Simulation of Adaptive Behavior (SAB90).* A Bradford Book. MIT Press, 1990.
100. John H. Miller and John H. Holland. Artificial adaptive agents in economic theory. *American Economic Review*, 81(2):365–370, 1991.
101. Johann Mitlöhner. Classifier systems and economic modelling. In *APL '96. Proceedings of the APL 96 conference on Designing the future*, volume 26 (4), pages 77–86, 1996.
102. D. E. Moriarty, Alan C. Schultz, and John J. Grefenstette. Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, 11:199–229, 1999. http://www.ib3.gmu.edu/gref/papers/moriarty-jair99.html.
103. Ichiro Nagasaka and Toshiharu Taura. 3D Geometic Representation for Shape Generation using Classifier System. pages 515–520. Morgan Kaufmann: San Francisco, CA, 1997.
104. Mukesh J. Patel and Marco Dorigo. Adaptive Learning of a Robot Arm. In Terence C. Fogarty, editor, *Evolutionary Computing, AISB Workshop Selected Papers*, number 865 in Lecture Notes in Computer Science, pages 180–194. Springer-Verlag, 1994.
105. Gregory J. E. Rawlins, editor. *Proceedings of the First Workshop on Foundations of Genetic Algorithms (FOGA91).* Morgan Kaufmann: San Mateo, 1991.
106. Robert A. Richards. *Zeroth-Order Shape Optimization Utilizing a Learning Classifier System.* PhD thesis, Stanford University, 1995. Online version available at: http://www-leland.stanford.edu/~buc/SPHINcsX/book.html.
107. Robert A. Richards and Sheri D. Sheppard. Classifier System Based Structural Component Shape Improvement Utilizing I-DEAS. In *Iccon User's Conference Proceeding.* Iccon, 1992.
108. Robert A. Richards and Sheri D. Sheppard. Learning Classifier Systems in Design Optimization. In *Design Theory and Methodology '92.* The American Society of Mechanical Engineers, 1992.
109. Robert A. Richards and Sheri D. Sheppard. Two-dimensional Component Shape Improvement via Classifier System. In *Artificial Intelligence in Design '92.* Kluwer Academic Publishers, 1992.
110. Robert A. Richards and Sheri D. Sheppard. A Learning Classifier System for Three-dimensional Shape Optimization. In H. M. Voigt, W. Ebeling, I. Rechenberg, and H. P. Schwefel, editors, *Parallel Problem Solving from Nature – PPSN IV*, volume 1141 of *LNCS*, pages 1032–1042. Springer-Verlag, 1996.
111. Robert A. Richards and Sheri D. Sheppard. Three-Dimensional Shape Optimization Utilizing a Learning Classifier System. In Koza et al. [84], pages 539–546.
112. Rick L. Riolo. Bucket Brigade Performance: II. Default Hierarchies. In John J. Grefenstette, editor, *Proceedings of the 2nd International Conference on Genetic Algorithms (ICGA87)*, pages 196–201, Cambridge, MA, July 1987. Lawrence Erlbaum Associates.
113. Rick L. Riolo. CFS-C: A Package of Domain-Independent Subroutines for Implementing Classifier Systems in Arbitrary User-Defined Environments. Technical report, University of Michigan, 1988.
114. Rick L. Riolo. *Empirical Studies of Default Hierarchies and Sequences of Rules in Learning Classifier Systems.* PhD thesis, University of Michigan, 1988.

115. Rick L. Riolo. The Emergence of Coupled Sequences of Classifiers. In Schaffer [125], pages 256–264.
116. Rick L. Riolo. The Emergence of Default Hierarchies in Learning Classifier Systems. In Schaffer [125], pages 322–327.
117. Rick L. Riolo. Lookahead Planning and Latent Learning in a Classifier System. In Meyer and Wilson [99], pages 316–326.
118. Rick L. Riolo. Modeling Simple Human Category Learning with a Classifier System. In Booker and Belew [15], pages 324–333.
119. Gary Roberts. A Rational Reconstruction of Wilson's Animat and Holland's CS-1. In Schaffer [125].
120. Gary Roberts. Dynamic Planning for Classifier Systems. In Forrest [44], pages 231–237.
121. George G. Robertson and Rick L. Riolo. A Tale of Two Classifier Systems. *Machine Learning*, 3:139–159, 1988.
122. Cédric Sanza, Christophe Destruel, and Yves Duthen. A learning method for adaptation and evolution in virtual environments. In *3rd International Conference on Computer Graphics and Artificial Intelligence, April 1998, Limoges, France*, 1998.
123. Teresa Satterfield. *Bilingual Selection of Syntactic Knowledge: Extending the Principles and Parameters Approach*. Kluwer, Amsterdam, 1999. Uses a learning classifier system to model human learning in biligual situations.
124. Shaun Saxon and Alwyn Barry. XCS and the Monk's Problems. In Lanzi et al. [91], pages 223–242. (this volume).
125. J. David Schaffer, editor. *Proceedings of the 3rd International Conference on Genetic Algorithms (ICGA89)*, George Mason University, June 1989. Morgan Kaufmann.
126. Dale Schuurmans and Jonathan Schaeffer. Representational Difficulties with Classifier Systems. In Schaffer [125], pages 328–333. http://www.cs.ualberta.ca/~jonathan/Papers/classifier.ps.gz.
127. Sandip Sen. Improving classification accuracy through performance history. In Forrest [44], pages 652–652.
128. Sandip Sen. A Tale of two representations. In *Proc. 7th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pages 245–254, 1994.
129. Sandip Sen. Modeling human categorization by a simple classifier system. WSC1: 1st Online Workshop on Soft Computing. Aug 19-30, 1996. http://www.bioele.nuee.nagoya-u.ac.jp/wsc1/papers/p020.html, 1996.
130. Lingyan Shu and Jonathan Schaeffer. VCS: Variable Classifier System. In Schaffer [125], pages 334–339. http://www.cs.ualberta.ca/~jonathan/Papers/vcs.ps.gz.
131. R. E. Smith, B. A. Dike, B. Ravichandran, A. El-Fallah, and R. K. Mehra. The Fighter Aircraft LCS: A Case of Different LCS Goals and Techniques. In Lanzi et al. [91], pages 285–302. (this volume).
132. Robert E. Smith. Memory Exploitation in Learning Classifier Systems. *Evolutionary Computation*, 2(3):199–220, 1994.
133. Robert E. Smith, B. A. Dike, R. K. Mehra, B. Ravichandran, and A. El-Fallah. Classifier Systems in Combat: Two-sided Learning of Maneuvers for Advanced Fighter Aircraft. In *Computer Methods in Applied Mechanics and Engineering*. Elsevier, 1999.
134. Robert E. Smith and David E. Goldberg. Reinforcement Learning with Classifier Systems: Adaptive Default Hierarchy Formation. Technical Report 90002, TCGA, University of Alabama, 1990.

135. Robert E. Smith and David E. Goldberg. Variable Default Hierarchy Separation in a Classifier System. In Rawlins [105], pages 148–170.
136. Wolfgang Stolzmann. An Introduction to Anticipatory Classifier Systems. In Lanzi et al. [91], pages 175–194. (this volume).
137. Wolfgang Stolzmann and Martin Butz. Latent Learning and Action-Planning in Robots with Anticipatory Classifier Systems. In Lanzi et al. [91], pages 303–320. (this volume).
138. R. S. Sutton. Learning to predict by the methods of temporal differences. In *Machine Learning 3*, pages 9–44. Boston: Kluwer, 1988.
139. Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224, Austin, TX, 1990. Morgan Kaufmann.
140. Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning – An Introduction*. MIT Press, 1998.
141. R.S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence*, 112:181–211.
142. S. B. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. De Jong, S. Džeroski, S. E. Fahlman, D. Fisher, R. Hamann, K. Kaufman, S. Keller, I. Kononenko, J. Kreuziger, R. S. Michalski, T. Mitchell, P. Pachowicz, Y. Reich, H. Vafaie, W. Van de Welde, W. Wenzel, J. Wnek, and J. Zhang. The MONK's problems: A performance comparison of different learning algorithms. Technical Report CS-91-197, Carnegie Mellon University, Pittsburgh, PA, 1991.
143. Andy Tomlinson and Larry Bull. A Corporate Classifier System. In A. E. Eiben, T. Bäck, M. Shoenauer, and H.-P Schwefel, editors, *Proceedings of the Fifth International Conference on Parallel Problem Solving From Nature – PPSN V*, number 1498 in LNCS, pages 550–559. Springer Verlag, 1998.
144. Andy Tomlinson and Larry Bull. On Corporate Classifier Systems: Increasing the Benefits of Rule Linkage. In Banzhaf et al. [4], pages 649–656.
145. Andy Tomlinson and Larry Bull. A zeroth level corporate classifier system. pages 306–313, 1999.
146. Andy Tomlinson and Larry Bull. A Corporate XCS. In Lanzi et al. [91], pages 194–208. (this volume).
147. Kirk Twardowski. Credit Assignment for Pole Balancing with Learning Classifier Systems. In Forrest [44], pages 238–245.
148. Nicolaas J. Vriend. On Two Types of GA-Learning. In S.H. Chen, editor, *Evolutionary Computation in Economics and Finance*. Springer, 1999. in press.
149. Nicolaas J. Vriend. The Difference Between Individual and Population Genetic Algorithms. In Banzhaf et al. [4], pages 812–812.
150. Nicolaas J. Vriend. An Illustration of the Essential Difference between Individual and Social Learning, and its Consequences for Computational Analyses. *Journal of Economic Dynamics and Control*, 24:1–19, 2000.
151. Christopher J.C.H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, 1989.
152. C.J.C.H. Watkins. Learning from delayed reward. PhD Thesis, Cambridge University, Cambridge, England, 1989.
153. Thomas H. Westerdale. An Approach to Credit Assignment in Classifier Systems. *Complexity*, 4(2), 1999.
154. Stewart W. Wilson. Knowledge Growth in an Artificial Animal. In Grefenstette [51], pages 16–23. Also appeared in Proceedings of the 4th Yale.

155. Stewart W. Wilson. Classifier Systems and the Animat Problem. *Machine Learning*, 2:199–228, 1987. Also Research Memo RIS-36r, the Rowland Institute for Science, Cambridge, MA, 1986.
156. Stewart W. Wilson. Bid Competition and Specificity Reconsidered. *Complex Systems*, 2(6):705–723, 1988.
157. Stewart W. Wilson. Classifier System mapping of real vectors. In *Collected Abstracts for the First International Workshop on Learning Classifier System (IWLCS-92)*, 1992. October 6–8, NASA Johnson Space Center, Houston, Texas.
158. Stewart W. Wilson. ZCS: A zeroth level classifier system. *Evolutionary Computation*, 2(1):1–18, 1994. http://prediction-dynamics.com/.
159. Stewart W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995. http://prediction-dynamics.com/.
160. Stewart W. Wilson. Explore/exploit strategies in autonomy. In Maes et al. [97], pages 325–332.
161. Stewart W. Wilson. Generalization in the XCS classifier system. In Koza et al. [83], pages 665–674. http://prediction-dynamics.com/.
162. Stewart W. Wilson. Get real! XCS with continuous-valued inputs. In L. Booker, Stephanie Forrest, M. Mitchell, and Rick L. Riolo, editors, *Festschrift in Honor of John H. Holland*, pages 111–121. Center for the Study of Complex Systems, 1999. http://prediction-dynamics.com/.
163. Stewart W. Wilson. Get Real! XCS with Continuous-Valued Inputs. In Lanzi et al. [91], pages 209–220. (this volume).
164. Stewart W. Wilson. State of XCS Classifier System Research. In Lanzi et al. [91], pages 63–82. (this volume).
165. Stewart W. Wilson and David E. Goldberg. A Critical Review of Classifier Systems. In Schaffer [125], pages 244–255. http://prediction-dynamics.com/.
166. Zhaohua Zhang, Stan Franklin, and Dipankar Dasgupta. Metacognition in Software Agents Using Classifier Systems. In *AAAI-98. Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 83–88, Madison (WI), 1998. AAAI-Press and MIT Press.

State of XCS Classifier System Research

Stewart W. Wilson

The University of Illinois, Urbana-Champaign IL 61801, USA
Prediction Dynamics, Concord MA 01742, USA
wilson@prediction-dynamics.com

Abstract. XCS is a new kind of learning classifier system that differs from the traditional kind primarily in its definition of classifier fitness and its relation to contemporary reinforcement learning. Advantages of XCS include improved performance and an ability to form accurate maximal generalizations. This paper reviews recent research on XCS with respect to representation, internal state, predictive modeling, noise, and underlying theory and technique. A notation for environmental regularities is introduced.

1 Introduction

A classifier system is a learning system that seeks to gain reinforcement from its environment based on an evolving set of condition-action rules called classifiers. Via a Darwinian process, classifiers useful in gaining reinforcement are selected and propagate over those less useful, leading to increasing system performance. The classifier system idea is due to Holland [4], who laid out a framework that included generalizability of classifier conditions, internal message-passing and reinforcement, and computational completeness. However, despite considerable research, the performance of the traditional system has been mixed, and there have been few advances on the initial theory.

Recently, Wilson [30] introduced XCS, a classifier system that retains essential aspects of Holland's model, while improving on it in some respects. XCS's definition of classifier fitness is different from the traditional one, and the system's organization includes ideas from the contemporary field of reinforcement learning [25]. In particular, due to the fitness change—which is theoretically based—XCS tends to evolve classifiers that are both accurate in their prediction of payoff and maximally general, resulting in higher performance as well as compactness of representation. The traditional model provides no theory as to how generalization would occur accurately, and in fact it does not. XCS's performance gets further theoretical transparency from the reinforcement learning (RL) connection, particularly the use of predictions and discounting. Theoretical understanding of performance in the traditional model is limited.

XCS has been investigated on problems including learning the Boolean multiplexer functions, and to find goals in grid-like "woods" and maze environments. XCS reaches optimal performance in these problems, which are generally more

P.L. Lanzi, W. Stolzmann, and S.W. Wilson (Eds.): LCS '99, LNAI 1813, pp. 63–81, 2000.

difficult than problems set for the traditional system, where performance is rarely optimal.

Because XCS appears both theoretically and practically to be a clear advance, and also because these developments have come rather quickly, it is desirable to undertake a review of the current state of XCS research. The following discussion assumes a familiarity with classifier systems, as well as acquaintance with the basics of XCS [30]. The material is organized under five topics: representation, internal state, predictive modeling, noise and uncertainty, and XCS theory and technique. First, though, it will be useful to introduce some simple notation to describe classifier system environments.

2 Environment Notation

A classifier system acts to gain payoff in its environment. Suppose that x represents a particular environmental state as detected by the system's sensors at time t and that a represents an action the system is capable of taking in that state. Let p represent an amount of payoff. Then the notation $(x, a) \rightarrow p$ says that if the system takes action a in state x, payoff p will be received. The expression states what we may term a *predictive regularity* of the environment.

With some further notation, we can represent the totality of predictive regularities. Let X stand for the set of all environmental states (as read from its sensors) that the system will encounter, let A represent its set of available actions, and let P represent the set of possible payoffs. Then there is a *mapping* $\mathrm{X} \times \mathrm{A} \Rightarrow \mathrm{P}$ from the product set of states and actions to the set of payoffs, that expresses the totality of predictive regularities in the system's environment. The mapping can also be regarded as representing the system's *payoff* landscape, since it associates a value from P with every point in the space of sensor variables and actions defined by the $\mathrm{X} \times \mathrm{A}$ product set. (Not all points in the space may be realizable by the system).

Suppose that, for a given action a and payoff p, more than one state x_i satisfies $(x_i, a) \rightarrow p$. Let us write $(\{x\}, a) \rightarrow p$ to express all such states compactly, where we have dropped the subscript. We term the set $\{x\}$ a *categorical regularity*. It denotes a maximal set of states which are equivalent in the sense that they all satisfy $(x, a) \rightarrow p$ for a given value of a and a given value of p.

We can now look at the payoff landscape $\mathrm{X} \times \mathrm{A} \Rightarrow \mathrm{P}$ from a categorical perspective. In general, the landscape will not be indefinitely "irregular". Instead, it will contain regions $(\{x\}, a)$ over which the value of p is constant, or nearly so. Each such region is a categorical regularity. The points within the region are equivalent with respect to payoff.

To summarize, a predictive regularity says that action a in state x leads reliably to payoff p. A categorical regularity says that, given a, there may be many states x which result in the same p.

Note, as an important aside, that often the payoff from the environment is *zero* for most pairs (x, a). This is the case in environments where the system must perform a sequence of actions in order to arrive in a state where payoff can be

obtained. Thus, much of the time, the system must "find its way" without benefit of payoff directly from the environment, i.e., without so-called *external payoff* or reward. To deal with this situation, Holland proposed the *bucket brigade* algorithm, in which external payoff is in effect passed back from each activated classifier to its predecessor. The bucket brigade was in fact an early proposal for reinforcement learning in sequential environments.

The technique used in XCS is similar in some respects, but is closer conceptually and algorithmically to *Q-learning* [27], a theoretically grounded and widely used technique in contemporary reinforcement learning. The result is that XCS exists in an environment where the payoff for each (x, a) is a combination of external payoff (if there is any) and the system's current estimate of the maximum payoff available in the succeeding state y. Thus the payoff environment for XCS is a combination of payoffs from the "true" external environment and an internal estimation process modeled after Q-learning. With this qualification, we shall continue to characterize the environment by the mapping $\mathrm{X} \times \mathrm{A} \Rightarrow \mathrm{P}$, at least until the next qualification!

3 Representation

In general, the objective of XCS is to capture, in the form of classifiers, the predictive regularities of the environment, $\mathcal{E}$, so the system can attain payoff at optimal rates. A second objective is to evolve a classifier population that captures $\mathcal{E}$'s categorical regularities compactly and accurately. In principle, compactness of representation can occur because classifier syntax permits the formation of *generalizations*, classifiers whose conditions match more than one state. In the traditional syntax, where the condition is a string from $\{1, 0, \#\}$, a condition can match more than one state if it contains #'s.

Ideally, for each categorical regularity $(\{x\}, a) \rightarrow p$ in $\mathcal{E}$, the system should evolve a classifier $\langle condition \rangle : \langle action \rangle \Rightarrow \langle prediction \rangle$ in which the action and prediction are a and p, respectively, and the condition exactly corresponds to $\{x\}$; that is, all and only the states in $\{x\}$ are matched by the condition. Put another way, the generalization expressed by the classifier should correspond exactly to the categorical regularity. Evolving such classifiers is important because it minimizes population size in terms of distinct classifiers (or *macroclassifiers*—see Section 7.4), resulting in rapid matching and visibility of the system's knowledge to human beings. Equally important, in many problems the mapping $\mathrm{X} \times \mathrm{A} \Rightarrow \mathrm{P}$ cannot be represented by a reasonable number of classifiers *unless* its categorical regularities are captured. Finally, if classifiers are capable of expressing generalizations, it is important that the generalizations be *accurate*, i.e., that all of the state-action combinations expressed by the classifier indeed result in the same payoff. Otherwise performance will usually be less than optimal, since some classifier predictions will be incorrect, causing the system to choose suboptimal actions.

In light of these objectives, there are two broad respects in which the traditional syntax appears limited. First, the individual variables of the condition

can take just two values, 1 and 0 (plus #). This is not a limitation where the environment's variables are also strictly two-valued. But in many environments, the variables of interest are continuous, or may take on more than two discrete values. Clearly, if $\mathcal{E}$ has continuous variables, it will only be in fortuitous cases that its categorical regularities can be accurately captured by thresholding to fit the {1,0,#} coding, and then only with just the right thresholds.

This problem can be alleviated to some extent by allowing variables in the condition to be expressed by binary numbers greater than 1, e.g., by four bits to represent a range from 0 to 15. The problem here is that many possible regularities of $\mathcal{E}$ are still not expressible. For example, 111# will match the range 14-15, but a range such as 13-14 cannot be expressed as a four-position string from {1,0,#}.

One solution [30,32] would be to explore condition encodings where each variable is expressed by a real interval. For instance, the interval ($13.0 \leq v < 14.0$) would be expressed by the numbers 13.0 and 14.0 as an ordered pair, or, alternatively, by 13.5 as a center value and 0.5 as a "spread" or delta value around it. The classifier condition would be a concatenation of such pairs. The condition would be satisfied, or match, if each component of the input, a real vector, fell within the interval denoted by the corresponding ordered pair. The condition would thus express a conjunct of interval predicates. The interval predicates would be restricted to connected intervals (two or more separated intervals could not be expressed in one classifier), but that will in many cases be satisfactory. Classifiers with interval predicates would be evolved using genetic operators appropriate to real-valued variables. The application to practical areas like "data mining" is clear, since the variables there are often continuous.

The second limitation of the traditional syntax stems from the fact that the condition is a conjunct of individual predicates, i.e., an AND. For problems, such as Boolean functions, where the categorical regularities are expressible as ANDs of the variables, the syntax is appropriate because it matches $\mathcal{E}$'s own "syntax" directly. However, in other environments, the categorical regularities can be quite different, and therefore so is the optimal syntax.

For example, consider a robot having a linear array of light sensors around its periphery. Appropriate actions may depend on how much total stimulus is coming from one direction vs. another. A natural kind of classifier condition might compare the sum of certain sensor values with the sum of certain other values; if the first sum was greater, then a certain payoff would be predicted for an action like rotating to the left. Such a condition is impossible to express with the traditional syntax, and in fact would require a large number of separate classifiers. Instead, one would like a condition syntax that could directly express a predicate involving comparison of sums. Such a predicate would represent a categorical regularity with respect to all sensor value combinations that make it true. More generally, one would like a syntax permitting expression of *any* predicate, or truth-function of the input variables.

Predicates of any type can be constructed as Lisp s-expressions of primitive functions appropriate to the given environment. For example, in the pro-

blem just mentioned, appropriate primitives would include '+' and '>'. Genetic programming [10] has shown in principle how such predicates might be evolved, leading to the suggestion [29] of *s-classifiers*, classifiers with s-expression conditions. Recently, Lanzi and Perrucci [17] reported learning of the Boolean 6-multiplexer using XCS with s-classifiers. Learning times in terms of number of examples seen were approximately equal to times with the traditional syntax, suggesting—perhaps surprisingly—that use of s-expressions may not bring greater learning complexity. Their experiments with sequential problems in a "woods" environment were equally promising in this respect. The results encourage extension of XCS and s-classifiers to environments with a wide range of appropriate predicate functions.

It is worth noting that genetic programming and XCS use similar fitness measures. While in GP the fitness measure can vary with the problem domain, very often it involves the degree of fit between the candidate function and a number of pre-specified "fitness cases". That is, the fitness of the candidate is measured by the accuracy with which its output matches the values of the fitness cases. In XCS, the fitness of a classifier is measured by the accuracy with which it predicts the payoff that is received when it matches the input and the system chooses its action. The classifier's "fitness cases", so to speak, are the states it matches and the associated payoffs (for its action). This similarity suggests that much of the technique developed for GP should carry over well to XCS. It also suggests thinking of the classifier's condition primarily as a payoff-predicting function instead of as a predicate testing for a match. It is in fact reasonable to consider extensions in which the classifier predicts not a constant payoff value, but a value that varies with input state. The fitness of such a classifier would still be measured by the accuracy of the predicted values. Evolution of such classifiers could lead to further reductions in population size, since each could cover (x, a) pairs having different payoff values. In the limit, the population might evolve to consist of just one species of classifier, predicting all payoffs. This speculative outcome would be like the result of standard genetic programming, in which just one best individual is sought, but would occur by a different and possibly faster route.

S-classifier conditions are variable-length and, unlike traditional conditions, not all input variables need be represented. Missing variables are in effect "don't cares". The simplest kind of variable-length condition would concatenate the variables that are included as an unordered string of $\langle variable, value \rangle$ pairs. The format is the same as that employed by Goldberg, Korb and Deb [3] in the messy genetic algorithm. Lanzi [13] experimented with this condition format using genetic operators and other techniques similar to those in the messy GA, while leaving unchanged the rest of XCS. After encountering and solving certain problems of under- and over-specification, Lanzi was able to achieve optimal performance on problems also solved optimally by regular XCS. Lanzi's *messy classifier system* has the property that a population evolved for a certain set of sensors can be installed in a system having additional sensors and through mutation alone will continue evolving to take advantage of the additional sensors.

This follows from the position-independence and lack of fixed format of the messy CS, and is experimentally demonstrated in the paper. Lanzi suggests that position-independence "can improve the portability of the behaviors learned between different agents", a property that would presumably also hold for the s-classifier system.

The work on representation in XCS is aimed at increasing its learning power and ability to deal with payoff landscapes of any reasonable sort. Each payoff landscape has regularities. For high performance and representational efficiency, XCS must not only detect the regularities, but must also be able to represent them accurately. Pilot success with s-classifiers hints that this will be possible for any environment, but there is great room for further advances and understanding.

4 Predictive Modeling

XCS learns the mapping $\mathrm{X} \times \mathrm{A} \Rightarrow \mathrm{P}$, a payoff model of $\mathcal{E}$, but the environment offers additional information that XCS could usefully predict. Specifically, given a state x and an action a, there results not only a payoff p, but a next state y. If XCS were to learn the mapping $\mathrm{X} \times \mathrm{A} \Rightarrow \mathrm{Y}$, with elements $(x, a) \rightarrow y$, it could, from any present state x, consider chains of actions extending into the future; that is, it could plan. A representation of the mapping $\mathrm{X} \times \mathrm{A} \Rightarrow \mathrm{Y}$ is usually termed a *predictive model*.

Holland [5] and Riolo [20] proposed classifier systems in which each classifier has a condition, an action, and a prediction of the resulting next state. The systems (which differed somewhat) were to learn predictive models of $\mathcal{E}$, but because the experiments did not involve the discovery or GA component, the systems's workability in practice is unknown. Sutton's [24] *Dyna* system used a predictive model coupled with standard reinforcement learning algorithms to speed up learning in goal-finding problems. Dyna learned the payoff mapping $\mathrm{X} \times \mathrm{A} \Rightarrow \mathrm{P}$ not only by trial and error in the actual environment, but also through hypothetical actions carried out in the predictive model as it was being built up, resulting in much faster learning of the payoff map, at least in terms of real actions in $\mathcal{E}$. Sutton experimented in environments with uniquely labeled states, and used tabular learning algorithms such as tabular Q-learning, so that no generalization over states occurred. However, the Dyna concept would carry over directly to XCS. Stolzmann's [23] Anticipatory Classifier System (ACS) predicts the next state and has been demonstrated experimentally in small problems. ACS does not use the GA, but instead generates a new, correct, classifier when existing classifiers fail to predict correctly, i.e., are inaccurate. The system has many interesting features that deserve further investigation.

An XCS classifier suitable for both payoff and predictive modeling could have the form $\langle condition \rangle : \langle action \rangle \Rightarrow \langle prediction \rangle : \langle expecton \rangle$, in which $\langle expecton \rangle$ is a vector of variables describing the succeeding state [30]. The fitness of this classifier, as in regular XCS, would be measured by the accuracy of the prediction of both payoff and the next state. Questions arise, of course. How is the accuracy

of the expecton measured? Suppose $(x, a) \rightarrow p$ in $\mathcal{E}$, but the next state, y, is not definitely predictable; what form should the corresponding classifier have? To address these questions, suppose that the expecton can contain unspecified positions, along the lines of don't cares but in this case more like "don't knows". Then the expecton's accuracy would be measured by a function of the number of specified positions and the agreement between the values in those positions and the actual next state's values. In situations where the next state was not completely predictable, but some state variables were, a high fitness classifier could evolve which successfully predicted just those variables and left the others as "don't knows". The expecton would in effect generalize over next states.

It is not clear, however, that the payoff and predictive mappings should be learned by the same set of classifiers. Perhaps it is better to have classifiers of the form $\langle condition \rangle : \langle action \rangle \Rightarrow \langle prediction \rangle$ and of the form $\langle condition \rangle : \langle action \rangle \Rightarrow \langle expecton \rangle$. Then the underlying state regularities could be captured independently of the payoff configuration, which might change. Too, it could be that the payoff map was simple compared with the state map, and having separate classifier populations would exploit this. If the environment contained several kinds of payoff, with distinct payoff maps, use of a separate state classifier system would mean the state information had only to be learned once.

The best approaches to adding predictive modeling to XCS are completely open at present. The benefits include faster learning along the lines of Dyna, but incorporating generalization. Other benefits may stem from the fact that the state classifier system, if it involves significant generalization in its conditions and expectons, would yield a sort of simplified recoding of $\mathcal{E}$, that is, a simplified description of the state landscape. How this might be useful is not clear at present.

5 Internal State

At this point we must disturb the neat $\mathrm{X} \times \mathrm{A} \Rightarrow \mathrm{P}$ environment model by introducing the Markov/non-Markov distinction. An environment has the *Markov property* if the prediction of p given x and a cannot be improved by knowledge of states preceding x. Strictly speaking, this is the Markov property with respect to payoff. Such an environment might or might not be Markov with respect to next states. But, for simplicity, by Markov we shall mean Markov with respect to payoff. An environment is *non-Markov* if it is not Markov.

The Markov/non-Markov distinction is crucial in reinforcement learning because it says whether an environment can or cannot be predicted on the basis of current input information. If so, the system can rely entirely on that information. If not, it must remember, or encode, something about the past and use that in conjunction with current input information in order to make correct decisions. If $\mathcal{E}$ is Markov (and deterministic), its payoff properties can be represented by $\mathrm{X} \times \mathrm{A} \Rightarrow \mathrm{P}$. If $\mathcal{E}$ is non-Markov, the mapping is not well defined because the p resulting from (x, a) is not predictable.

A very simple example of the distinction is the following [19]. Imagine a gift packing task in which a gift must be placed in an initially closed box after which the box is wrapped. Initially the box is closed with no gift inside and must be opened and filled. Later, with gift inside and again closed, it must be wrapped. In both cases, the system's sensors see a closed box, but the actions called for are different. If the system only sees the box, it can't perform correctly unless it remembers what it has done, or has made some other mental note, i.e., unless it maintains some form of internal state. The environment is non-Markov. If, however, the system's sensors include one that weighs the box, then for a high enough gift to box weight ratio, the environment becomes Markov and no internal state is required.

Note how the distinction very much depends on the system's sensors. Environments can be "made" Markov if there are enough sensors. Markov environments will become non-Markov if the number of sensors (or their ability to make distinctions) is reduced. Thus it is not the environment itself that is Markov or non-Markov, but the environment as sensed by the system. In speaking of an environment as Markov or non-Markov, we shall mean the environment in conjunction with the system's sensors.

How can XCS learn non-Markov environments? In reinforcement learning, there are two major approaches. One, the "history window", extends the input vector to include inputs from previous time-steps, enlarging the effective state space until the mapping $\mathrm{X}' \times \mathrm{A} \Rightarrow \mathrm{P}$, with X' the extended space, becomes well-defined, and the problem is effectively Markov. Though X' grows exponentially with the length of the history window, this approach can be satisfactory if the window is very short. The other approach attempts to identify and remember past events that will disambiguate $\mathrm{X} \times \mathrm{A} \Rightarrow \mathrm{P}$, and these are combined with the current x to make decisions. Some versions of this approach employ recurrent neural networks. The problem here is that the complexity of the search is again exponential with distance into the past.

Holland's classifier system contains an internal message list, where information from one time-step can be posted by a classifier and then be read by another classifier on a later time-step. It was hoped that classifiers would evolve whose postings and readings led to high performance in situations where past information was needed (i.e, in non-Markov environments). Holland's idea was neither a history-window approach, nor one that searched purposively for critical past information. Rather, the approach was Darwinian, relying on variation and selection. Unfortunately, because of the complexity of the message list structure, and also—in our opinion—because fitness was based on "strength" and not accuracy as in XCS, Holland's system showed only limited success on non-Markov problems [21,22].

Wilson [29] suggested that coupling between posting and reading classifiers would be enhanced if the internal state were embodied in a simple bit register instead of a list of messages. He observed that in many problems, decisions needing past information often require only one or a few bits. Classifier structure could take the form, $\langle \textit{external condition} \rangle \langle \textit{internal condition} \rangle$:

⟨*external action*⟩ ⟨*internal action*⟩ ⇒ ⟨*prediction*⟩. The internal condition would apply to the internal bit register; the internal action would set, reset, or ignore bits there.

The suggestion was taken up by Cliff and Ross [2]. They added an internal register to Wilson's [29] ZCS system (a classifier system with fitness based traditionally on strength). The resulting system, ZCSM, performed quite well, though not optimally, in several simple non-Markov mazes. Examination of evolved populations showed that ZCSM indeed evolved classifiers that employed the internal register to discriminate the mazes' "aliased" (indistinguishable) states.

Lanzi [11] obtained improved results with XCSM, which added an internal register to XCS, but more difficult mazes could still not be learned optimally. In further work [12], he proposed that exploration of internal actions should be carried out differently than exploration of external actions. He found that too much exploration of internal actions would prevent the stable formation of classifiers that acted appropriately on the internal register contents. To use the register effectively implies that the system must both create an appropriate internal "language" (of register settings) and also learn how to interpret that language. Lanzi experimented with XCSMH, a modified XCSM in which the internal register settings (the "language") were explored only by the genetic algorithm and were not explored as part of the action selection process. The language was thus explored more slowly than its interpretation. This resulted in optimal performance on more difficult (more aliased states) mazes. In further work [14], he discovered that still improved performance would occur if the internal register was somewhat redundant, that is, had size somewhat greater than the minimum number of bits needed in principle to disambiguate the environment's aliased states. This last result confirms, in classifier systems, a similar result observed by Teller [26] who used an internal register with genetic programming.

Cliff and Ross [2] worried that systems using an internal register would not scale up well, because the amount of needed exploration would grow exponentially with the register size. Lanzi's results suggest this may not be a concern, since better performance results from less internal exploration than might have been thought. In effect, it is not necessary to explore all possible settings of the register. Many settings will be interpretable to yield high or optimal performance; it is only necessary to find one of them. A broad discussion of the non-Markov work is given in [18].

Good learning in non-Markov environments is perhaps the largest outstanding problem in RL. Progress using an internal register with XCS suggests this approach should be pursued vigorously. Most important would seem to be to study further how best to do "internal exploration", to understand better the complexity implications of the register and its size, and of course to investigate increasingly difficult environments. Learning of hierarchical behavior is another outstanding RL problem. It is possible that internal "languages" will evolve that use the register in a hierarchical manner, i.e., certain bit positions will encode longer-term contexts while others encode behavioral details. This might open the way to realization of "hierarchical classifier systems" [28].

6 Noise and Uncertainty

The vast majority of classifier system research has used noiseless, deterministic environments, but these ideal properties do not hold in many potential applications. Robotic sensors are often extremely noisy, as are the data in most "data mining" problems. Besides noise inherent in the environment, XCS is vulnerable to noise-like uncertainty in two other respects. First, if the environment is non-Markov and the system has insufficient internal state to disambiguate it, the system's classifiers will not be able to arrive at stable predictions. Classifier errors will not go to small values or zero and this error will be indistinguishable from error that results from "true" environmental noise. Second, if a classifier is overgeneral, it will make prediction errors that are also not distinguishable from the other kinds, at least until the overgenerality is eliminated.

Since in XCS fitness is based on accuracy, it is important to determine XCS's sensitivity to external noise. Lanzi and Colombetti [16] tested XCS in Markov environments where the system's probability of carrying out its intended action was less than 1.0. Specifically, in a grid-world maze, the system would with probability ϵ "slip" to one or the other of the two cells adjacent to its intended destination cell. Thus the system was subject to a kind of action noise. For values of ϵ up to about 0.25, Lanzi and Colombetti found that performance was nearly as good as XCS without generalization (no #'s) or tabular Q-learning, all subject to the same action noise. Thus even with generalization "turned on", XCS was able evolve classifiers as accurate as those evolved when generalization was not enabled. The system was able to eliminate errors due to overgeneralization even in the presence of external noise over which it had no control. However, at an ϵ of 0.50, performance broke down drastically.

The authors then introduced a technique that resulted in nearly optimal performance even when ϵ was 0.50. A new parameter μ was given each classifier for keeping an estimate of the minimum prediction error among the classifiers in its action set [A]. Each time a classifier takes part in [A], it looks at the other classifiers in [A] and notes the value of the prediction error parameter of the classifier with the lowest such parameter. Our original classifier then updates its μ parameter using the value just identified. Finally, when it updates its own prediction error parameter, it uses not its current prediction error, but that error minus its current value of μ. The procedure is based on the heuristic that the minimum prediction error in the action set is a good estimate of the actual external noise, since all the classifiers are subject to it. Subtracting μ out means that each classifier's error estimate approaches the value it would have in a noiseless environment, so that overgeneralization errors can be eliminated and performance is maintained. The technique is a new kind of example of how the population-based nature of XCS—the fact that it supports multiple hypotheses for each situation—can markedly aid the search.

It is important to test the technique on the other kinds of environmental noise, namely noise in the sensors and in external payoffs. Both are important for prediction from real data, since data sets used for learning will often contain errors in data (sensor) values and inconsistencies in outcomes (payoffs). Lanzi

and Colombetti's technique promises to be valuable in learning predictions from noisy data while retaining the ability to detect generalizations, properties which will be of great interest to human users of XCS in data applications.

7 XCS Theory and Technique

7.1 Generalization

From the beginning, XCS has displayed a strong tendency to evolve classifiers that detect and accurately represent the categorical regularities of its environment. This is based, first, on making fitness depend on accuracy of prediction (in contrast to "strength" in the Holland framework). Second, it depends on the use of a niche GA [1]. Together, these have the consequence that of two equally accurate classifiers where one matches a subset of the states matched by the other, the more general classifier will win out because it has more reproductive opportunities (for a detailed discussion, see [30]).

The drive to evolve accurate, maximally general classifiers at the same time as it learns the mapping $X \times A \Rightarrow P$ suggests that XCS tends to evolve populations that consist of the smallest possible set of non-overlapping classifiers, thus representing the environment's payoff landscape optimally. Kovacs [7] termed this the *XCS Optimality Hypothesis*, and demonstrated it for Boolean multiplexer problems. It is important to explore this direction further, in particular by trying Boolean problems which are less symmetrical than the multiplexers. In the multiplexer function, each categorical regularity covers an equal portion of the input domain. As a result, for random inputs, all parts of the population are updated and subjected to GA processes at approximately the same rate. For a function with categorical regularities of unequal size, this equality of rates would not hold and one can expect that XCS's functionality will be affected to a degree that should be determined. A similar kind of test would be achieved with the multiplexer by changing to an input distribution that was not uniformly random.

XCS's tendency to evolve accurate, maximally general classifiers is not guaranteed. Situations can arise, particularly in sequential problems using discounting, in which overgeneral classifiers may fail to be eliminated, even though their accuracy is low. Elimination depends on the existence of a more accurate competitor classifier in every action set where the overgeneral occurs. Normally this will be the case, due to the genetic mechanisms. However, if populations are too small relative to the number of distinct categorical regularities in $\mathcal{E}$, action set sizes will be small so that the GA will be slow to generate the competitors and an overgeneral may not be eliminated before it causes trouble, i.e., before action errors occur that in the worst case bring a breakdown of overall performance.

Overgenerals will also not be eliminated if the errors in an action set are so small that the system's accuracy mechanism cannot differentiate the fitnesses. Discounting can have this effect, since the predictions—and thus errors—become smaller farther from sources of external payoff. XCS originally defined accuracy as a negative exponential function of prediction error. If one very small error is

nevertheless twice as big as another one in the same action set, an exponential function will not differentiate them well since the ratio of two exponentials is an exponential of the difference of their arguments, and may be small. The problem is substantially reduced by changing the accuracy function to a negative power function of the error [31]. Then, if one error is twice another, the accuracy ratio is a power of two, independent of the actual error values.

Despite larger populations and an improved accuracy function, performance can still sometimes break down due to failure to eliminate overgenerals rapidly enough. For such "emergencies", Lanzi [15] developed a mechanism termed *specify*. Every action set is tested to see if its average error relative to the population average error exceeds a threshold. If so, a new classifier is added to the action set that covers the current input and whose condition has a predetermined probability of don't care positions. The intention is to add a fairly specific classifier to the high-error action set that will compete in accuracy with the overgeneral classifiers that are probably present. Specify works well in that performance breakdowns due to overgeneralization are eliminated. At the same time, because it acts locally, specify does not restrain the system from forming accurate generalizations where warranted.

In the first work on XCS, the GA occurred in the match set [M]. Later [31] it was moved to the action set [A]. The result was that in certain problems the population size—in terms of macroclassifiers—evolved to be smaller. The smaller populations contained the same accurate general classifiers as before, but there were fewer more-specific versions of these present, and fewer inaccurate classifiers. The apparent reason for this improvement is fairly subtle.

Consider a match set, and suppose that the state x being matched participates in two categorical regularities of $\mathcal{E}$: $(\{x\}_1, a_1) \rightarrow p_1$ and $(\{x\}_2, a_2) \rightarrow p_2$. Suppose classifiers C1 and C2 in [M] perfectly represent these regularities. That is, the condition of C1 exactly fits $\{x\}_1$ and the condition of C2 exactly fits $\{x\}_2$. Both classifiers are accurate and maximally general. What if the GA crosses them? If the two conditions are identical, then the offspring conditions will be identical to the parent conditions. However, if the two conditions are *not* identical, the offspring conditions may be different from those of both of the parents (e.g., cross #1 and 1#). Since by hypothesis the parents were accurate and maximally general, the offspring, if different from the parents, will not be and will thus constitute a kind of crossover "noise" that the system will have to eliminate. Thus even if the system has found accurate, maximally general classifiers that match the present input, the action of crossover will—if $\{x\}_1$ and $\{x\}_2$ differ—continue to generate suboptimal classifiers.

Suppose however that the GA, and thus crossover, occurs in [A] instead of [M]. Now only one of C1 and C2 is present and not the other (they have different actions). Since they cannot be crossed, the offspring just mentioned will not be formed and do not have to be eliminated. C1 (suppose it's the one present) will still cross with more-specific and more-general versions of itself that may be present too in the action set; the same crosses would occur in the match set.

But the "noise" due to crosses with C2 will not. The argument is clearest in an extreme case: assume the match set consists *only* of C1 and C2.

The reduction in population size due to moving the GA to [A] was observed in problems where, if a classifier was accurate and maximally general, it was often not the case that another classifier with a different action but the same condition would also be accurate and maximally general (another way of saying that C1's condition is different from C2's). However, in problems where this was the case (conditions of C1 and C2 always the same), the population size was unchanged in moving the GA from [M] to [A]. Thus the shift to the action set appears justified both in principle and experimentally. But further examination in both respects is needed to understand fully what is happening.

If the GA does occur in the action set, what can we say that crossover is actually doing? Note that all classifiers in both the match and action sets must match the input. However, all classifiers in the action set have the same action. Thus the GA may be regarded as searching for the best—i.e. most general and accurate—classifier to represent the regularity $(\{x\}, a) \rightarrow p$ that the present (x, a) pair belongs to. The classifiers in the action set have conditions of varying specificity. They exist somewhere along the line between completely specific and completely general. Crossover produces offspring that in general occupy points on the line different from their parents. Clearly the GA—using crossover, mutation, and selection—is searching along this line, driven by a fitness measure, accuracy, that is strongly correlated with specificity. This is the heart of XCS's discovery process, and a detailed theory is definitely called for.

Also called for is a theory of how fitness based on accuracy interacts with the reproductive opportunity afforded by greater generality to drive the system toward classifiers that are both accurate and maximally general. Wilson [30] presented a heuristic argument: of two equally accurate classifiers, the more general one would win out because being more general it occurs in more action sets and thus has greater chance to reproduce. But the theory needs to be put on a quantitative basis.

Wilson [31] reported a secondary generalization method, *subsumption deletion*, that further reduces, or "condenses", population size. The primary method, as discussed above, causes classifiers to generalize up to the point where further generalization (e.g., addition of #'s in the traditional syntax) would result in errors. However, the process may actually stop short of this point, since a formally more general classifier (more #'s) will only win out if it can match more states of the environment (yielding more reproductive opportunities). But those states may not actually be present in $\mathcal{E}$, i.e., $\mathcal{E}$ may not contain all the states permitted by the encoding. So the system will not evolve classifiers that are as formally general as they could in fact be.

To see this, suppose $\mathcal{E}$ contains the categorical regularity $(\{000, 001\}, a) \rightarrow p_1$. Suppose $\mathcal{E}$ also contains $(100, a) \rightarrow p_2$, where p_2 and p_1 are different. Suppose further that the states 010 and 011 are not present. The classifier $00\# : a \Rightarrow p_1$ is clearly accurate. So also is $0\#\# : a \Rightarrow p_1$. The second classifier is more general, but since it matches no more actual states than the first, it will not win out

reproductively, and the two will coexist in the population. Note, however, that the classifier $\#\#\# : a \Rightarrow p_1$, being inaccurate, will not survive. (See also the discussion in [15]).

In many problems, it is desirable to end up with classifiers that are as formally general as possible while still being accurate. The resulting population is smaller, and the key differentiating variables (e.g., the first bit in the example above) are more perspicuous. Subsumption deletion accomplishes this by checking, whenever a new classifier is generated by the GA, whether its condition is logically subsumed by the condition of an accurate classifier already in the action set. If so, the new classifier is abandoned and not added to the population. Subsumption deletion is a powerful addition to the primary generalization mechanism, but it is somewhat risky. For each categorical regularity, it tends to eliminate all but the formally most general, accurate classifier. If the environment should change, such that certain states (e.g., 010 and 011 above) now occur, that classifier could fail drastically resulting in a breakdown of performance. Retaining classifiers like $00\# : a \Rightarrow p_1$ above—i.e., using only the primary process— prevents this.

7.2 Connection with RL

XCS's learning algorithm is a straightforward adaptation of basic Q-learning. The prediction of each classifier in the action set is updated by the current reward (if any) plus the discounted value of the maximum system prediction on the next time-step. The system prediction for a particular action is a fitness-weighted average of the predictions of each classifier in the match set that advocates that action. The maximum system prediction is the maximum, over the match set, of the system prediction for each action. The essential difference with Q-learning is that classifier—that is, rule—predictions are updated from predictions of other rules. In Q-learning, action-values—that is, predictions for pairs (x, a)—are updated from other action-values. XCS's memory is contained in rule sets, whereas Q-learning's memory is stored in a table with one entry for each state-action pair.

Of course, Q-learning-like algorithms have also been employed, for example, in neural-net learning. But it is most interesting to compare XCS with tabular Q-learning. That is the only case for which convergence proofs are known. In addition, there is an important sense in which XCS's algorithm is like tabular Q-learning, but with generalization over the entries of the table.

If XCS is applied to a problem, but with generalization turned off (no initial #'s, and none introduced by mutation), the result will be a set of classifiers corresponding to the equivalent Q-learning table, except that for most state-action pairs that do not actually occur in $\mathcal{E}$ there will be no classifier. Thus, without generalization, XCS will in effect build the Q-learning state-action table "from scratch", but only for (x,a) pairs that actually occur in $\mathcal{E}$. Performance, apart from some noise introduced by the GA, will be identical to that of tabular Q-learning. If the GA is completely turned off, and classifiers are only created by covering, performance will be the same as for tabular Q-learning, and the final population will correspond precisely to Q-table entries that actually occur in $\mathcal{E}$.

With generalization turned on, performance by XCS will normally reach the same level as for tabular Q-learning, but will take longer due to errors as accurate general classifiers are discovered and emphasized. If the problem contains categorical regularities and these can be represented by XCS's syntax, then generalization will result in a population (in macroclassifiers) that is smaller than the corresponding Q-table, often by a significant factor. If one examines classifier predictions once the system has essentially stopped evolving, they are found to equal the predictions of the corresponding Q-table. Thus while no proofs of convergence are available, empirically XCS converges like Q-learning, but with generalization as a bonus. The same sort of parallel might hold with other reinforcement learning algorithms, provided XCS's update procedures were correspondingly modified. (See [9,14] for detailed discussion of the relation between XCS and Q-learning.)

7.3 Complexity

Real-world problems often have very large state spaces. Reinforcement learning methods such as Q-learning that depend on tables of values scale exponentially with the dimensionality of the state space. Not only does the table grow exponentially, but so does the time needed to fill in the values. Tabular Q-learning is not sensitive to categorical regularities in the environment and so cannot avoid the exponential explosion. But research with XCS has suggested that because it detects the regularities—and when it can represent them syntactically—XCS's learning complexity is dependent not on the dimensionality of the space, but on the complexity of the underlying regularities. In this respect it may differ significantly from other approaches to handling large state-spaces such as radial basis functions, tiling (CMAC) methods, and neural networks, whose complexities are ultimately tied to state-space size [25].

Information about XCS's learning complexity comes from experiments with the family of Boolean multiplexer functions [31]. Three functions were learned: the 6-, 11-, and 20-multiplexers, where the numbers indicate the lengths of the input bit-string l or in other words, the dimensionality of the state space. The disjunctive normal forms (DNF) for the functions contain, respectively, 4, 8, and 16 terms. Associated with each term are exactly four payoff regularities, so there are, respectively, 16, 32, and 64 payoff regularities in the three spaces. An example of a payoff regularity for the 6-multiplexer is
$(\{000000, 000001, 000010, 000011, 000100, 000101, 000110, 000111\}, 0) \Rightarrow p$,
where p is the payoff associated with a correct decision. The accurate, maximally general classifier corresponding to this regularity is $000\#\#\# : 0 \Rightarrow p$.

XCS reached 100% performance on the three functions after seeing, on average, 2,000, 10,000, and 50,000 random input strings. Final population sizes in macroclassifiers were, respectively, 55, 148, and 345. In contrast, the state space sizes are, respectively, 64, 2,048, and 1,048,576, growing exponentially. If one infers from these limited data that the learning times grow by a factor of five from one multiplexer to the next, then the times can be fit to a function cg^p, where g is the number of regularities in the function, $p = \log 5 = 2.32$, and $c =$

3.22. Thus the learning time complexity would be a low power of the number of regularities in the space.

A very tentative theory of this result can be proposed. The search for a set of classifiers that accurately represent the regularities may involve two factors. One is the number of regularities itself. The other is the length of the classifier condition, because each position of the condition must be "set" correctly. The length of the condition is l, but the number of regularities is approximately proportional to l—it approaches an exact proportionality in the limit of large multiplexers. It seems reasonable to conclude that the search time should depend on product of these two factors, which would be consistent with the function just proposed.

Further research is needed to test these ideas. In particular, XCS should be applied to larger multiplexers, and to other function families. At this point, however, XCS promises to scale up well in problems for which its representational syntax is appropriate.

7.4 Techniques

Traditionally, classifier system populations contain many classifiers that, due to reproduction without associated crossover or mutation, are structurally identical, i.e., they have the same conditions and actions. Wilson [30] introduced *macroclassifiers* in order to eliminate this redundancy as well as reduce processing time and save storage. In a population of macroclassifiers, each newly generated classifier is examined to see if a structurally identical classifier already exists. If so, the existing classifier's numerosity parameter is increased by one and the new classifier is abandoned. If not, the new classifier is added to the population with its numerosity initialized at one. XCS uses a macroclassifier population, but all operations are otherwise conducted as though the population consists of ordinary classifiers ("microclassifiers"); that is, numerosities are taken into account. In a macroclassifier population, the sum of the numerosities, N, equals the number of underlying microclassifiers and is a constant. N is the number that in traditional systems denotes the population size.

A population of macroclassifiers consists entirely of structurally unique individuals. As such it permits a much better view of the system's "knowledge" than does a traditional population. Classifiers with high numerosity tend to be those with maximal accurate generalizations, so the most significant knowledge can be determined readily by sorting the population by numerosity [7]. Furthermore, the size of the population in macroclassifiers measures the system's ability to detect and represent the regularities in $\mathcal{E}$. The macroclassifier idea is in fact what permits realization of the improvement in space complexity that XCS achieves through generalization. A traditional population would maintain size N regardless of the generalizations within it.

Kovacs [6] compared the behavior of XCS with and without macroclassifiers on the 6-multiplexer problem. He found no significant difference in performance. A slight improvement in system prediction error using macroclassifiers was explainable since when a new classifier already exists it is not introduced into the

population, so its arbitrary initial parameters can't affect system predictions. It therefore appears that the macroclassifier technique is valid and uniformly beneficial.

Whenever XCS generates a new classifier, either through the GA or covering, a classifier is effectively deleted in order to maintain the microclassifier population size at N. Selecting a classifier for deletion is done as though the population consists of microclassifiers. The actual "deletion" is carried out by simply decrementing some classifier's numerosity by one; if the classifier's numerosity is exactly one, it is removed from the population.

Kovacs [8] studied techniques for making the deletion selection and introduced a new one that is superior. Previously [30], two techniques had been used. Each classifier kept an estimate of the number of classifiers in the action sets it occurred in. In the first technique, the probability of deletion was proportional to the estimate. In the second technique, it was proportional to the estimate but multiplied by an integer greater than one if the classifier's fitness was below the population average fitness by a certain factor. Use of the action set size estimate tends to maintain action sets at about the same size, thus equally distributing system resources (classifiers). The integer multiplier in the second technique was designed to penalize very low fitness classifiers and eliminate them rapidly.

The second technique worked better than the first in that it resulted in smaller population sizes. But it often deleted newly generated classifiers before they had a chance to gain fitness. Kovacs's new technique resolved this by combining the two previous ones. Each classifier was given an additional parameter keeping track of the number of times it had been a member of an action set. In the Kovacs technique, if that number was less than, e.g., 20, the probability of deletion was as in the first old technique. If the number was equal to or greater than 20, the second old technique was used. The new technique exhibited the advantages of the earlier two without their disadvantages.

8 Conclusions

We have reviewed the current state of XCS classifier system research, identifying the main accomplishments and suggesting future directions. A notation for the environment's predictive and categorical regularities was introduced to aid the discussion.

Promising areas for further research include: (1) extending condition syntax so as to accept input vectors with continuous, ordinal, nominal, and mixed components, and to enable representation of a greater range of environmental regularities; (2) continued exploration of the full representational generality of s-classifiers; (3) experimentation with XCS systems that predict the next state, so as to develop more complete models of the environment; (4) continued work on internal state to determine the scope of the register method and its potential for supporting hierarchical behavior; (5) further research on filtering noise of all types, with applicability to learning from real data sets; (6) continued basic work on accurate generalization, aimed at understanding all relevant phenomena; (7) development of a schema theory for XCS's genetic search process; (8) exploration of RL techniques other than Q-learning in XCS; (9) further investigation of

XCS's learning complexity experimentally and theoretically; and (10) continued examination of the basic XCS component algorithms in search of improvements.

Important areas not discussed because next to nothing is presently known include: (1) use of continuous actions, e.g., turn exactly 34 degrees; and (2) basing XCS on continuous time, instead of discrete time-steps. Existing RL work in both of these areas could be brought to bear. In addition, it is important to elucidate the relationship between XCS and traditional classifier systems [9].

XCS is a new kind of classifier system showing significant promise as a reinforcement learner and accurate generalizer. Its full potential is just beginning to be explored.

References

1. Booker, L.: Intelligent behavior as an adaptation to the task environment. Ph.D. Dissertation (Computer and Communication Sciences). The University of Michigan (1982).
2. Cliff, D. and Ross, S.: Adding temporary memory to ZCS. Adaptive Behavior **3**(2) (1994) 101-150.
3. Goldberg, D. E., Korb, B., and Deb, K.: Messy genetic algorithms: motivation, analysis, and first results. Complex Systems **3** (1989) 493-530.
4. Holland, J. H.: Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In: R. S. Michalski, J. G. Carbonell & T. M. Mitchell (Eds.), Machine Learning, An Artificial Intelligence Approach. Volume II. Los Altos, California: Morgan Kaufmann (1986).
5. Holland, J. H.: Concerning the emergence of tag-mediated lookahead in classifier systems. Physica D, **41** (1990) 188-201.
6. Kovacs, T.: Evolving Optimal Populations with XCS Classifier Systems. MSc. Dissertation, Univ. of Birmingham, UK (1996).
7. Kovacs, T.: XCS classifier system reliably evolves accurate, complete, and minimal representations for Boolean functions. In: Roy, Chawdhry and Pant (Eds), Soft Computing in Engineering Design and Manufacturing, Springer-Verlag London (1997) 59-68.
8. Kovacs, T.: Deletion schemes for classifier systems. In: Banzhaf, W., et al, (eds.), Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99). Morgan Kaufmann: San Francisco, CA (1999) 329-336.
9. Kovacs, T.: Strength or accuracy? A comparison of two approaches to fitness calculation in learning classifier systems. This volume.
10. Koza, J. R.: Genetic Programming. Cambridge, MA: The MIT Press/Bradford Books (1992).
11. Lanzi, P. L.: Adding memory to XCS. In: Proceedings of the IEEE Conference on Evolutionary Computation (ICEC98). IEEE Press (1998).
12. Lanzi, P. L.: An analysis of the memory mechanism of XCSM In J. Koza et al (eds.), Proceedings of the Third Annual Genetic Programming Conference, San Francisco, CA: Morgan Kaufmann (1998) 643-651.
13. Lanzi, P. L.: Extending the representation of classifier conditions, part I: from binary to messy coding. In: Banzhaf, W., et al, (eds.), Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99). Morgan Kaufmann: San Francisco, CA (1999) 337-344.

14. Lanzi, P. L.: Reinforcement Learning with Classifier Systems. PhD Thesis, Politecnico di Milano (1999).
15. Lanzi, P. L.: An analysis of generalization in the XCS classifier system. Evolutionary Computation, **7**(2) (1999) 125-149.
16. Lanzi, P. L. and Colombetti, M.: An extension to the XCS classifier system for stochastic environments. In: Banzhaf, W., et al, (eds.), Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99). Morgan Kaufmann: San Francisco, CA (1999) 353-360.
17. Lanzi, P. L. and Perrucci, A.: Extending the representation of classifier conditions, part II: from messy coding to s-expressions. In: Banzhaf, W., et al, (eds.), Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99). Morgan Kaufmann: San Francisco, CA (1999) 345-352.
18. Lanzi, P. L. and Wilson, S. W.: Toward optimal classifier system performance in non-Markov environments. Evolutionary Computation, to appear (2000).
19. Lin, L.-J.: Reinforcement Learning for Robots Using Neural Networks. Ph.D. Dissertation (School of Computer Science), Carnegie Mellon University (1993).
20. Riolo, R. L.: Lookahead planning and latent learning in a classifier system. In J.-A. Meyer and S. W. Wilson (eds.), From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior. Cambridge, MA: MIT Press (1991) 316-326.
21. Robertson, G. G. and Riolo, R. L.: A tale of two classifier systems. Machine Learning, **3** (1988) 139-159.
22. Smith, R. E.: Default Hierarchy Formation and Memory Expoitation in Learning Classifier Systems. Ph. D. Dissertation, The University of Alabama, Tuscaloosa, Alabama (1991).
23. Stolzmann, W.: Anticipatory classifier systems. In: Proceedings of the Third Annual Genetic Programming Conference, J. Koza et al (eds.). San Francisco, CA: Morgan Kaufmann (1988) 658-664.
24. Sutton, R. S.: Reinforcement learning architectures for animats. In: J.-A. Meyer and S. W. Wilson (eds.), From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior. Cambridge, MA: MIT Press (1991) 288-296.
25. Sutton, R. S. and Barto A. G.: Reinforcement Learning: An Introduction. Cambridge, MA: The MIT Press/Bradford Books (1998).
26. Teller, A.: The evolution of mental models. In: K. E. Kinnear, Jr. (ed.), Advances in Genetic Programming. Cambridge, MA: The MIT Press/Bradford Books (1994).
27. Watkins, C. J. C. H.: Learning from Delayed Rewards. Ph.D. Dissertation, Cambridge University (1989).
28. Wilson, S. W.: Hierarchical credit allocation in a classifier system. In: Proceedings of the Tenth International Joint Conference on Artificial Intelligence. Los Altos, CA: Morgan Kaufmann (1987) 217-220.
29. Wilson, S. W.: ZCS: a zeroth level classifier system. Evolutionary Computation **2**(1) (1994) 1-18.
30. Wilson, S. W.: Classifier fitness based on accuracy. Evolutionary Computation **3**(2) (1995) 149-175.
31. Wilson, S. W.: Generalization in the XCS classifier system. In Proceedings of the Third Annual Genetic Programming Conference, J. Koza et al (eds.). San Francisco, CA: Morgan Kaufmann (1998) 665-674.
32. Wilson, S. W.: Get real! XCS with continuous-valued inputs. This volume.

An Introduction to Learning Fuzzy Classifier Systems

Andrea Bonarini

Politecnico di Milano AI and Robotics Project,
Dipartimento di Elettronica e Informazione, Politecnico di Milano
Piazza Leonardo da Vinci, 32 , 20133 Milano, Italy
bonarini@elet.polimi.it
WWW home page: http://www.elet.polimi.it/people/bonarini

Abstract. We present a class of Learning Classifier Systems that learn fuzzy rule-based models, instead of interval-based or Boolean models. We discuss some motivations to consider Learning Fuzzy Classifier Systems (LFCS) as a promising approach to learn mappings from real-valued input to real-valued output, basing on data interpretation implemented by fuzzy sets. We describe some of the approaches explicitly or implicitly referring to this research area, presented in literature since the beginning of the last decade. We also show how the general LFCS model can be considered as a framework for a wide range of systems, each implementing in a different way the modules composing the basic architecture. We also mention some of the applications of LFCS presented in literature, which show the potentialities of this type of systems. Finally, we introduce a general methodology to extend reinforcement distribution algorithms usually not designed to learn fuzzy models. This opens new application possibilities.

1 Introduction

Learning Classifier Systems learn rules whose clauses are strings of bits. Each bit may represent a Boolean value for the corresponding variable [11, 72]; alternatively, groups of bits may denote intervals for the variable [18]. A genetic algorithm operates on these strings of bits evolving the best solution.

Since the beginning of the last decade [70, 40], an alternative model has been considered for rules learned by similar algorithms, opening a new research track about *Learning Fuzzy Classifier Systems* (*LFCS*). The main idea is to consider the symbols in the rule clauses as labels associated to fuzzy sets. It is thus possible to learn fuzzy rules that implement a mapping from input to output where these are represented by real-valued variables. In the next section we will provide a background about fuzzy sets and fuzzy rules, introducing also the main motivations supporting the adoption of fuzzy models. In section 3, we introduce the general architecture of a LFCS, pointing out the similarities with the standard LCS. We mention in section 4 some of the proposals in this field presented in the last years either under the label "LFCS" [70, 2], or under related

P.L. Lanzi, W. Stolzmann, and S.W. Wilson (Eds.): LCS '99, LNAI 1813, pp. 83–104, 2000.

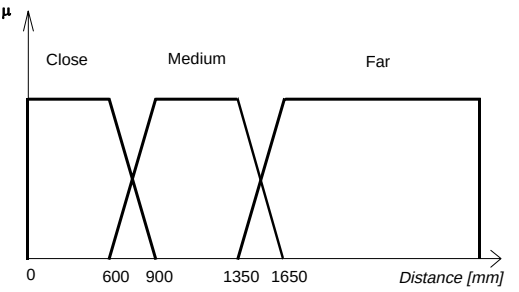


Fig. 1. Some membership functions defining fuzzy sets on the distance dimension

labels such as "genetic learning of fuzzy rules" [49], "fuzzy Q-learning" [1, 22, 3], and others. We consider a set of features characterizing these systems, and we show some of their most interesting aspects. Then, in section 5, we focus on one of these aspects and we introduce a general methodology [6] to extend the reinforcement distribution algorithms designed for crisp models [35], such that they can be adopted also in some classes of LFCS. This provides a wide number of possible learning algorithms, most of which have to be fully explored, yet; each of them potentially matches different application needs.

2 Fuzzy Sets and Fuzzy Rules

The theory of fuzzy sets has been introduced in 1965 by Lotfi Zadeh [73] as an extension of the standard set theory, which defines the so called *crisp sets*. Crisp sets are characterized by a Boolean membership function: an element $\bar{x}$ of the universe of discourse X (say, an interval of real numbers) is either a member of the set or it is not. Fuzzy sets are characterized by a continuous membership function $\mu_l(x)$ which maps $\bar{x}$ to a *"membership degree"* taking values in the interval $[0, 1]$. For instance, in figure 1 you may see membership functions for some fuzzy sets. Each of them maps elements having a certain distance d (in millimeters) to the corresponding degree of membership to a fuzzy set. Let us consider the fuzzy set *"close"*. In the example reported in figure, an element $\bar{x}$ of the universe of discourse (in our case the distances) belongs to the fuzzy set *close* to the extent $\mu_{close}(\bar{x})$. For instance, for the fuzzy set depicted in figure 1, $\mu_{close}(690) = 0.7$. Giving a semantic value to the label associated to the fuzzy set, we can say that the value $\bar{x} = 690$ can be $classified$ as *close* with membership degree 0.7. Although fuzzy sets can be used in many types of domain, in this paper we focus on a real-valued domains, which are interesting for many applications.

Fuzzy sets have been introduced to represent all those classes that people does not consider crisp: in this concern, they are an important tool to represent conceptual models which match the expert's and designer's needs. However, there are many other motivations to consider fuzzy models.

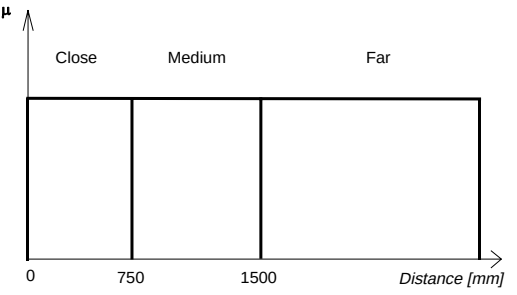


Fig. 2. Some membership functions defining crisp sets on the distance dimension

Their nature makes it possible to define partially overlapping classes, as in figure 1, thus making it possible a gradual classification of elements in the universe of discourse. For instance, a distance of 690 mm is classified as *close* with a membership degree $\mu_{close}(690) = 0.7$, and, at the same time, as *medium* with a membership degree $\mu_{medium}(690) = 0.3$. A close distance, say $\check{x} = 691$ can be classified in a very similar way. This has an impact on the quality of a fuzzy model. For instance, if the real values to be classified are measured values, affected by noise, the fuzzy model shows a graceful degradation with the increasing noise. By contrast, a model based on crisp sets (intervals) shows abrupt changes when the real value falls in the close interval, due to noise. In figure 2, you may see how a value $\bar{x} = 749$ is classified in a completely different way than the close value $\check{x} = 751$. The same considerations also hold for possible small errors in modeling. A fuzzy set model is known to be robust with respect to modeling imprecision and input noise [46]. Imperfect learning and adaptation may affect the quality of the resulting model, and the intrinsic robustness of a fuzzy model plays an important role to smooth this effect.

Another important feature of the fuzzy set definition is that it gives the possibility to define models where each real value is related to one or more classes with a specific degree of membership for each of them. This can be considered an alternative representation for a real value, which contains information that can be used to reason at a higher level of abstraction (sets instead of numbers). At the same time the quantity of information brought by the real value can be maintained in the fuzzy representation, by the set of membership degrees. If we want to maintain the precision of the real values, we can with fuzzy models, and we cannot with interval models. Moreover, fuzzy models give us the possibility to decide to loose precision, when we want to do that to implement modeling needs. For instance, in figure 1, all the values $\bar{x}$ between 0 and 600 are considered the same by the fuzzy model, since they all match the fuzzy set *close* with the same degree $\mu_{close}(\bar{x}) = 1$. In this case, the designer of the system has decided that all those values are mapped to a unique configuration in the fuzzy model. For instance, this make it possible to associate the same action of all these values, in case we are implementing a controller whose input variable is this distance.

Being an alternative representation of real values, fuzzy sets can be used to implement mappings between real values, with an information quality similar to that of real-valued mathematical functions. The main difference is that conceptual aspects are explicitly represented in fuzzy models, whereas they are embedded in mathematical functions. Moreover, this gives to a learning system the possibility to operate on a small number of symbols, the labels associated to the fuzzy sets, while trying to learn real-valued mappings.

Often, fuzzy models are implemented by fuzzy rules analogous to:

```
  IF (FrontDistance IS Low) AND
     (LeftDistance is High) AND
     (RightDistance is Don_T_Care)
THEN (TurnDirection IS Left)
```

This rule implements a mapping from the input of the controller of an autonomous agent to the control action. The rule implements a simple strategy such as: if there is something in front of the agent, and there is a free way on the left, whatever there is on the right, turn left. A *don't care* symbol may replace any of the labels denoting fuzzy sets, meaning that the value of the corresponding variable is not relevant, that is any real value matches the antecedent. We may define many rules, whose antecedents are defined by fuzzy sets, and may partially overlap. Usually, this is a desired property, since it guarantees graceful degradation and robustness [46].

Rules are local models, each covering a fuzzy subset of the input. The overall mapping comes from the interaction among the rules that match a state. Each rule has a matching degree computed as a *T-norm* [19] (for instance, the *minimum*) among the matching degrees of the antecedent clauses. The matching degree is considered a weight for the rule: as much the rule matches the state, as much its proposed output has to be considered relevant. Rules proposing the same fuzzy set as output, should aggregate the corresponding weight by a *T-conorm* [19](for instance, the maximum). At the end it is possible to compute real values for the output variables, by defuzzyfying the composition of the pairs $< l, \mu_l(\bar{x}) >$ for each fuzzy output. This is done by adopting one of the many defuzzyfication methods available, among which the *Center Of Gravity* (*COG*) method is probably the most common. This considers each fuzzy set as an area cut at the level given by the corresponding weight, and computes the weighted average among the so obtained areas.

Finally, a fuzzy model consists of relationships among symbols related to data interpretation. This is a compact way to represent local models (one model for each fuzzy rule) and their interaction. Learning this type of models may be effective, since learning algorithms may focus on local, very simple models; therefore, the search space is usually relatively small, and the interaction between close models is ruled by the above mentioned, traditional fuzzy operators. The interaction among local models, due to the intersection of neighbor fuzzy sets guarantees that local learning reflects on global performance [4].

3 Comparing LCS and LFCS

A generic architecture for a LCS is shown in figure 3. We will show that it is general enough to be adopted also for LFCS, discussing where the fuzzy model impacts on the architecture. This is also the architecture of the system we have implemented to study both LCS and LFCS [9, 10].

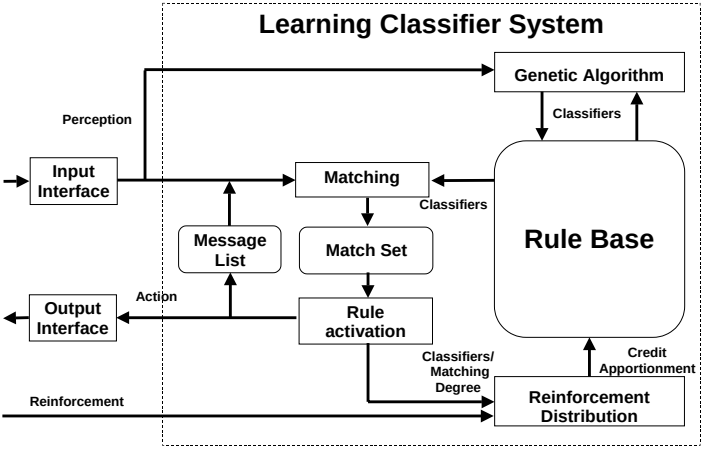


Fig. 3. The generic architecture of a LCS

3.1 Matching

Signals coming from the environment through the Input Interface, and messages present in the internal Message List are matched against the rules. We assume, as the most general case, that the input values are real numbers. The matching activity depends on the classifier model: if it is based on intervals (or on Boolean variables), real numbers are classified accordingly and the matching rules inserted in the match set. If the classifier model is based on fuzzy sets, also the matching degree is computed, and associated to the classifiers in the match set.

3.2 Rule Activation

The rule activation module has to select the rule(s) that will produce the output. Usually, in standard LCS, only one rule is selected, whereas in LFCS all the rules matching with a degree greater than a given threshold (which may also be 0) are triggered and the action is computed by a fuzzy combination of the output of

the single rules; this is equivalent to an aggregation of the fuzzy output values, followed by a defuzzyfication, often implemented as an integral of the proposed output fuzzy values, weighted by the corresponding matching degrees.

3.3 Reinforcement Distribution

Reinforcement has to be distributed according to a reinforcement distribution algorithm. We discuss in detail in section 5 the wide range of possibilities we have. In general, reinforcement distribution algorithms for fuzzy rules may consider to distribute reinforcement proportionally to the degree of matching of the rules that have contributed to the action(s) that bring the system to receive reinforcement.

3.4 Genetic Algorithm

The population update is done by a genetic algorithm which may apply many operators in addition to the standard mutation and crossover, such as creep mutation [61], cover detector [72], and many others [27]. In general, the most appropriate operators are selected by considering the different solution models to be learned, as we will see in the next section.

4 Features and Approaches

In this section, we introduce some of the features along which it is possible to classify LFCS proposals presented in literature. We discuss the problems related to each of them, and we put in evidence how these have been tackled.

4.1 Solution Models

The population members on which the genetic algorithm operates may represent different aspects, all leading to the definition of a rule base. In general, with a fuzzy model we have to define: the number, the shape, the position and the extension of the membership functions, both for the input and the output variables, the number of the fuzzy rules and their shape, stating the relationship between input and output. The proposals presented in literature focus on one or the other, or also on many aspects at the same time, in relationship with the application faced and the personal attitudes of the researchers. It is impossible to state which representation is better, but we can make general considerations about the dimension of the search space, and the number of trials needed to explore it in a suitable way. In general, models representing more aspects imply a larger search space and can be applied only in simple, or simplified, applications. In complex application, it is possible to focus the learning activity only on the most relevant aspects, fixing a priori some of the many parameters involved.

Rule Bases and Single Rules A large part of the LFCS proposals is based on the explicit representation of rules. As with standard LCS, there are two main tracks: either a chromosome represents a whole rule base [69, 13, 58, 14, 33], as in the Pittsburgh approach [64], or it represents a single rule [70, 61, 2, 5, 10, 44], as in the Michigan approach [31]. In the first case, the system is optimized in its globality, but each chromosome has to be evaluated during a long trial, thus requiring a large amount of computational effort. Moreover, it is possible that genetic activity destroys local equilibria also not related to the portion of the search space explored at a given time; this is even more critical in LFCS, since all the rules are interrelated by the partial overlapping of the membership functions defining their input clauses. Following the Michigan approach, each rule is evaluated only when it actually contributes to obtain reinforcement. Genetic operators modify single rules, thus performing local updating. The overall optimization comes from the fact that the rules strongly interact with each other. In LCS, the interaction between rules is only indirect and depends on the state reached by the system after the activation of a rule. In LFCS there is another mechanism, due to the coordinated activation of rules whose input membership function partially overlap. This implies that each action comes from the cooperation among rules, that in another situation may cooperate with different relative weights, or even with different rules. The best rule to cover a portion of the input space is thus a rule which cooperates with the rules that trigger in the same sub-space to produce the best action [4]. This is a problem different from that faced by LCS. Local learning of fuzzy models, such as comes with the Michigan approach, is suitable for a robust and effective overall optimization. The value of each rule comes not only from its suitability to accomplish the task, but also from its cooperation with neighbor, cooperating rules. A local approach may explore efficiently the combination between different close rules.

The Nagoya approach [21] tries to consider the advantages of both the Pittsburgh and the Michigan approaches: the whole rule base is coded in one chromosome, but the genetic operations and evaluation is done locally only on the rules activated at a given step.

Proposals explicitly learning the mapping between fuzzy input and output often reduce the search space by considering the membership function as given a priori [4, 5, 8, 70, 28, 24], and thus implementing a sort of *structural learning* on the shape of the rules. This is similar to what happens in LCS, and can be accepted when the partition of the input space can be obtained by experts, or by other optimization procedures. Other proposals learn membership functions and rules in two steps [45, 43], possibly cyclically activated in subsequent refinements [62]. Finally, there are some systems trying to optimize rules and membership functions at the same time [49, 47, 13, 54, 68, 17]. These approaches are usually feasible only for small application models due to the dimension of the search space.

Membership Functions An indirect way to represent a rule base is to define the set of the possible input values (in our case, membership functions), and

then consider that the rule base consists of all the rules having the evolved input values [57] and appropriate output, maybe learnt with other methods. This leads to the generation of complete rule bases, each considering all the combinations of all the input values. This may be a problem in large applications, where we would also like to identify a small number of effective rules among all the possible, which may be a large number. In other applications most of the possible input combination never occur in the environment, and their insertion in a rule base brings only a computational burden.

Some LFCS proposals focus on the optimization of membership functions, then considering a complete rule base generated from them. In general, the number, the position of the MF centers, and their coverage are learned. The last two values may be considered as real numbers [60, 43], and thus optimized by appropriate GAs, considering, for instance, special operators such as creep mutation. One of the problems with such an approach concerns the constraint we may like to impose on the MF definition: for instance, it is desirable that any point in the real variable range be covered by at least one MF (two would guarantee robustness), and it is rare that an application requires MFs completely including other MFs. These constraints have to extend the genetic operators in order to guarantee correct solutions [60, 68]. Another possible approach, that avoids the management of such constraints, consist of considering the MFs as following pre-defined patterns designed to satisfy the constraints; for instance, neighbor MFs should always cross at membership degree 0.5, the highest point of a MF being the lowest of the left and right neighbors. Then it is possible to represent the pattern configuration by binary strings to be optimized. For instance, [57] adopt such a schema to learn the controller of a mobile robot that has to avoid moving obstacles. A third approach considers pre-defined, alternative sets of membership functions (for instance, differing by the number of fuzzy sets defined for each variable, or by their shapes) [34]; this reduces the computational burden, but requires a priori knowledge to define the MF sets, or acceptance of a comparably low accuracy, due to partitioning introduced a priori and not learned.

The LFCS proposals working on MFs follow two approaches mutuated by fuzzy modeling: they either consider the same set of membership functions for all the rules in the rule base, as common in fuzzy control [55], or a set of membership functions different for each rule, as common in fuzzy model identification [67]. The first approach [36, 49, 48, 60, 54] is less computationally complex, but produces a set of membership functions optimized with the constraint of being the best set for all the rules. The second approach [53, 16, 13] requires much more computing time, but produces optimal results for each rule. This approach is only viable when the model is simple (few rules and few variables), and it is appropriate when a high level of precision is required, such as, for instance, in prediction or fine control applications. In some cases, adopting particular strategies suggested by the application domain (such as sliding mode approach for control [16, 15]), may help to reduce the search space.

Unusual Models A way of reducing the length of the model to be learned consists of representing only the aspects which are discovered to be relevant for the application. This may be done by adopting a *messy* representation [25]. On this line, Hoffmann et al. [29, 28, 27] represent sets of rules by a messy schema, including unordered pairs of name of variable and name of label for each rule. It is thus possible to define rules covering sparsely a wide range of variables and labels. The authors introduce ad hoc genetic operators, and succeed in developing a fuzzy controller for a real, mobile robot.

Another interesting coding approach has been taken by Leitch and Probert [53, 51, 52, 50] who represent a fuzzy rule base, including the membership functions for each rule, by a variable-length string; each semantically different section (a membership function, antecedents and consequents) is represented by bits, and is separated from the others by a symbol different from binary values. This gives the possibility to represent a variable number of elements of the rule base at different precision degrees. A parser interprets this fairly free evolving string as a rule base and makes it possible the generation of output from real-valued input.

4.2 Reinforcement Function

As it is common in reinforcement learning, the reinforcement function is selected following the designer's mood, without any theoretical considerations. Thus, we can see very simple reinforcement functions, such as providing reinforcement when an autonomous robot is not close to any object, in an obstacle avoidance task [7], or giving a reinforcement proportional to the distance from the corridor walls, in a robot navigation task [10]. On the other side we may find reinforcement functions that represent so much of the problem solution to make it difficult to justify the learning activity [58, 59].

The reinforcement function also reflects the theoretical background of the designer and the kind of application. Thus, people from Control Theory or from Identification Theory often propose reinforcement functions based on some *error* evaluation, which provide reinforcement at each evaluation step, people from Classification Theory proposes Boolean reinforcement functions to be applied at each classification, people from Artificial Life tend to propose high-level, general reinforcement functions, often producing delayed reinforcement.

4.3 Reinforcement Distribution

The most popular reinforcement distribution algorithm in the first LFCS proposals [70] was *bucket brigade* [30], probably due to the influence of the first seminal papers about LCS [11, 12]. In the last years, there have been many proposals [22, 24, 23, 1, 3] to consider Q-learning [71] as a valid alternative. Recently, the author has established a research line on the adaptation of reinforcement distribution algorithms to the case of LFCS [6, 7], which gives the possibility to modify the algorithms proposed in the reinforcement learning area (such as the

two above mentioned, but also $TD(\lambda)$ [65], and its modifications [63] and hybridizations [35, 66]) to match the needs of learning fuzzy models. We discuss in detail some aspects of this proposal in section 5.

4.4 Messages

The original LCS proposal considered the output of the classifiers as messages to be sent either to the environment (thus implementing actions) or to an internal storage structure (*the message list*) which had the role of maintaining a sort of state memory. In turn, the classifier input was matched against messages coming from the environment (sensors) and from the message list. The first LFCS proposal [70] has considered messages in an anlogous way. Furuhashi et al. [20]noticed that message passing involving fuzzy sets may lead to excessive value approximation, thus suggesting to defuzzyfy the output messages, and to put the so-obtained values in the message list to be fuzzyfied again when matched by the input set. Most of the others implement reactive input/output models. In principle, messages are a powerful tool to implement dynamical systems, giving the possibility to maintain a kind of internal memory. However, they also increase the learning complexity, and in most of the faced applications a dynamical model is not needed.

4.5 Applications

In this section we mention some of the applications faced by LFCS. Most of them are benchmarks, widely recognized in the respective reference communities, others are real applications, where LFCS have shown their power. In most cases the dimension of the search space is relatively small (two-three input variables, with three to five membership functions each). This is also due to the abstraction possibilities provided by fuzzy modeling: a fuzzy model can maintain its effectiveness also when it describes the world at a very high level of abstraction, since the label used to denote high-level concepts are always grounded on numerical data, by means of the membership functions.

Control Control is one of the most successful application fields for the fuzzy technology. The most popular benchmark in this field is the inverted pendulum control, or cart-pole. The controller has to bring a cart, on which is mounted a rotating pole, to an equilibrium state, with the pole vertical, and, in some cases, the cart in a given position. The basic problem consists of four input variables (position and speed of both the cart and the pole, in a 2-D representation) and one output (the force to be applied to the cart). The problem is non-linear, thus providing a justification for fuzzy control, and it is complex enough to justify learning. Among the many proposals facing this problem, we mention [37, 69, 49, 48, 17].

Another popular problem is the control of a mobile robot to achieve simple tasks such as navigation [10, 8, 29, 28, 27, 53, 21], obstacle avoidance [57][56],

or target tracking [5, 14, 33]. In these cases the input variables may correspond to a representation of the sensor data (e.g., distances in six directions and contact [10]), or to a high level representation such as the robot position in a cartesian plane [21, 56].

Other specific problems successfully faced are: the docking control for spaceships [40, 59] and for mobile robots equipped with arms [58], control of underwater vehicle [16], control of a robot arm [62], plant control [38, 39, 41, 42, 68].

Data Classification Another wide class of LFCS applications is data classification. Here, real or symbolic input values have to be classified as belonging to one or more classes. The interest in fuzzy models comes here both from robustness to noisy input and to the possibility to provide not only the reference to a class corresponding to the input, but also a degree of membership to it. Also in this field there is a well known set of benchmarks, some of which have been faced by LFCS [34]. Some other specific applications concern: cancer diagnosis [43]

Prediction and Identification Prediction concerns building a model of a system whose past behavior is known in terms of historical series, such that it is possible to predict future behaviors of the system. Prediction may concern natural systems, such as the whether in a given region, as well as artificial, complex systems such as the financial system [44] or non-linear plants [54, 32]. Identification consists of identify a relationship from input to output that fits given data. Again, fuzzy systems are suitable to face these classes of problems for their robustness, and for the possibility to provide a real-valued output coming from reasoning on grounded symbols. Some of the proposals in this field consider different MFs for each rule [13, 32], which provide, in general a better fitting with the data. When MFs are considered to be the same for all the rules [70, 61, 62] the fitting may show undesired regularities, such as a staircase effect when trying to match a parabola.

5 Reinforcement Distribution Algorithms

In this section, we focus on algorithms whose role is to distribute the reinforcement, obtained by the learning system from the environment, to the rules that have been used to achieve the current state.

Many reinforcement distribution algorithms have been proposed over the years: Q-Learning [71], $TD(\lambda)$ [65] and its various extensions [63, 35, 66], Bucket Brigade [12], and the reinforcement distribution algorithm of XCS [72]. All of them are used to learn *discrete* models, where input and output are Boolean variables or intervals. We have already discussed the limitations of these models in real world applications, and the motivations to learn fuzzy models, instead. The impact of fuzzy modeling on the reinforcement learning process is discussed in this section together with an approach to modify the mentioned algorithms in order to effectively apply them on fuzzy models. Here we focus the presentation

on a model where fuzzy rules are chromosomes in a population (the Michigan approach) and MFs are pre-defined. Most of the general considerations done in this section apply also to other models.

5.1 Fuzzy Input Classification

In discrete models, the real-valued input is classified as belonging to an interval, independently from its position in the interval. A fuzzy classification, instead, associates to the label corresponding to the fuzzy set matching the real-valued input also the corresponding membership value. This makes it possible to quantify the degree of matching of the rule, and, thus to compute its relevance in the current situation. In standard fuzzy inference, this also influences the strength of the output proposed by the rule, and, as a consequence, its contribution to the action that will receive the reinforcement. Therefore, it makes sense to distribute the reinforcement proportionally to the degree of matching of the rule, which reflects on its contribution to the output. This is the first modification to the standard reinforcement distribution algorithms that we consider.

5.2 Overlapping Fuzzy Sets

Since fuzzy sets classifying the input are usually designed on purpose to overlap in pairs, each real-valued state is matched in general by more than one fuzzy rule with different antecedents. This is not true for the interval-based rules, since the intervals are usually disjoint and all the rules matching the current state have the same antecedents. In this discussion we do not consider "don't care" symbols, which may be present in both interval and fuzzy antecedents, for sake of clarity. This does not affect the considerations we are making. Fuzzy rules having different antecedents should cooperate to produce the best output. Therefore, whereas only one of the interval-based rules is selected to be triggered, we have to select one fuzzy rule from each subpopulation corresponding to each antecedent configuration. This also means that reinforcement should be distributed to more than one rule. If the common mid-crossing configuration of the membership functions is implemented, and reinforcement is distributed proportionally to the degree of matching of the antecedents, the distribution is automatically normalized, and this is also a desirable property.

5.3 Composition of Fuzzy Output

Due to the composition of fuzzy output and its defuzzyfication, the output of the fuzzy system is in principle real-valued. Actually, it is fine grained, since membership values, as any real value in computing, are represented with some finite precision, but we can consider them as real-valued. The output of the interval-based system is coarse-grained, consisting in a number usually quite small of different, possible values. This does not affect only the performance of the system (a fuzzy system shows usually a smoother behavior), but also the

learning mechanism, since the difference between the output of the fuzzy system in close time points is usually small, and this may reflect on small differences in reinforcement. This is not always desired, since it makes more difficult to distinguish among rules proposing different output values, since these are in some way averaged with the values proposed by the other, triggering, fuzzy rules. A detailed discussion about this problem is included in another article [9].

5.4 Fuzzy and Crisp Classifier Systems

In this section we consider a simplified type of LCSs [6]. Each classifier is a rule with antecedents and consequents. Each antecedent corresponds to a value for a *linguistic variable* associated to a real-valued variable. In *crisp LCS* we consider that each value is one of the symbols representing intervals of values of the real-valued variable. Note that this representation includes the Boolean representation, where the intervals are only two, corresponding to *true* and *false*. In FLCS each value is a symbol representing a fuzzy subset [46] of the set of the values the real-valued variable can take. A *don't care* symbol may replace any of the mentioned symbols, meaning that the value of the corresponding variable is not relevant, that is any real value matches the antecedent. The consequents are symbols corresponding to crisp values for each of the consequent variable. A classifier of this kind as a conceptual shape like the one presented in section 2.

We call the set of real values, one for each variable, in input to the learning system a *real-valued state*. At each *control step*, one crisp classifier is selected among the ones matching the current real-valued state, and its consequents are considered as the output of the system.

We call the set of fuzzy sets, one for each variable, matching a real-valued state, a *fuzzy state*. When operating with FLCSs, the current state matches different fuzzy states, i.e., different fuzzy classifiers. We consider subpopulations of classifiers having the same antecedent (eventually including don't cares) and different consequents. At each control step, we select one classifier for each matching subpopulation (i.e., one for each fuzzy state), we combine the proposed output values by a fuzzy aggregation operator [46], and we output the real value thus obtained.

In this section we focus on some reinforcement distribution algorithms that we have implemented as modules of our system to be applied to learn the rule-based models described in the last section. We firstly introduce the ones used with crisp models, then presenting a methodology to extend them to fuzzy models by considering their intrinsic nature. We associate to each classifier many parameters, among which we mention here only *strength*, and *accuracy*.

5.5 Algorithms for Crisp Classifiers

Here, we consider only the three algorithms, among the eleven we have implemented, which show the most characteristic features.

Crisp Q-Learning At time t the system is in the real-valued state s_t, after having executed action a_{t-1} from a state s_{t-1} matched by the interval-valued antecedent $\hat{s}_{t-1}$, and receives the reinforcement r_t. The Q-value given to the pair $\langle \hat{s}_{t-1}, a_{t-1} \rangle$ is updated by:

$$Q_t(\hat{s}_{t-1}, a_{t-1}) = Q_{t-1}(\hat{s}_{t-1}, a_{t-1}) + \alpha \left(r_t + \gamma \, max_a(Q_t(\hat{s}_t, a)) - Q_t(\hat{s}_{t-1}, a_{t-1}) \right)$$

with the learning rate $0 < \alpha < 1$ and the discount factor $0 < \gamma \leq 1$. The strength of the classifier c_{t-1} selected by the exploration policy at time t is $Q(\hat{s}_{t-1}, a_{t-1})$.

The accuracy of the classifier c_{t-1} is updated by the formula:

$$E_t(c_{t-1}) = E_{t-1}(c_{t-1}) + \omega \left(|r_t + \gamma \, max_a(Q_t(\hat{s}_{t-1}, a)) - Q_t(\hat{s}_{t-1}, a_{t-1})| - E_{t-1}(c_{t-1}) \right)$$

with the learning rate $0 < \omega \leq 1$.

Crisp $TD(\lambda)$ At time t the reinforcement distribution algorithm performs these steps.

1. Receive the eventual reinforcement r_{t-1} due to the action a_{t-1} selected by the policy in state s_{t-1}.

2. Compute the estimation error of the evaluation function $V(\cdot)$ in s_{t-1}, equivalent to the immediate prediction error ΔE of the corresponding classifier c_{t-1} by:

$$\Delta E(c_{t-1}) = r_t + \gamma \, V_t(s_t) - V_{t-1}(s_{t-1})$$

with the discount factor $0 \leq \gamma \leq 1$.

3. For any state $\hat{s}$:

3.1. Update the *eligibility* $e(\hat{s})$ by the formula:

$$e_t(\hat{s}) = \begin{cases} 1 & if \hat{s} = \hat{s}_{t-1} \\ \gamma \lambda e_{t-1}(\hat{s}) & otherwise \end{cases}$$

with $0 \leq \lambda \leq 1$.

3.2. If the eligibility of the state $\hat{s}$ is greater than a threshold value ϵ (with $0 \leq \epsilon \leq 1$), update the value function $V(\cdot)$ for any $\hat{s}$ according to:

$$V_t(\hat{s}) = V_{t-1}(\hat{s}) \alpha \Delta E(c_{t-1}) e_t(\hat{s})$$

with $0 < \alpha < 1$.

4. Update the value of the policy for the pair $\langle \hat{s}_{t-1}, a_{t-1} \rangle$, that is the strength w of the classifier c_{t-1}, by the formula:

$$w_t(c_{t-1}) = w_{t-1}(c_{t-1}) + \beta \left(\Delta E(c_{t-1}) \right)$$

with $\beta > 0$.

5. Update the accuracy of the classifier c_{t-1} by:

$$E_t(c_{t-1}) = E_{t-1}(c_{t-1}) + \omega \left(|\Delta E(c_{t-1})| - E_{t-1}(c_{t-1}) \right)$$

with $0 < \omega \leq 1$.

Crisp Bucket Brigade At time t the reinforcement distribution algorithm follows these steps.
1. Receive the eventual reinforcement r_{t-1} due to the action a_{t-1} selected by the policy in state s_{t-1} and the new action a_t selected by the policy to be done in the current state s_t.
2. Update the strength of the classifier c_{t-1} belonging to the action set of the previous step A_{t-1}, by the formula:

$$w_t(c_{t-1}) = w_{t-1}(c_{t-1}) + r_t + \frac{1}{n} \sum_{c_{t-1} \in A_{t-1}} \beta w_t(c_t)$$

where the bid coefficient $0 < \beta < 1$, and n is the number of classifiers in A_{t-1}.
3. Update the accuracy of each classifier c_{t-1} in A_{t-1} by:

$$E_t(c_{t-1}) = E_{t-1}(c_{t-1}) + \omega \left(|w_t(c_{t-1}) - w_{t-1}(c_{t-1})| - E_{t-1}(c_{t-1}) \right)$$

with $0 < \omega \leq 1$.
4. Reduce the strength of any classifier c_t in A_t, by the formula:

$$w_t(c_t) = 1 - \beta w_t(c_t)$$

5.6 Algorithms for Fuzzy Classifiers

We have extended the above presented algorithms to learn fuzzy models, following the considerations discussed here below.

A *real-valued state* is a set of real values for the input variables. We say that a real-valued state s matches a fuzzy state $\tilde{s}^i$ to some extent $\mu_{\tilde{s}^i}(s)$, when each real value s^k belonging to s belongs to the corresponding fuzzy set belonging to $\tilde{s}^i$, and $\mu_{\tilde{s}^i}(s)$ comes from the conjunction of the values $\mu_{\tilde{s}^i}(s^k)$, computed by the selected conjunction operator. Such operators belong to the class known as *T-norms* [46], among which the most common are *min* and *product*.

A crisp state usually matches more than one fuzzy state, while in standard LCS, it matches the set of intervals that define only one *interval state*. Let us consider subpopulations of classifiers for each fuzzy (or interval) state. In crisp LCS, classifiers belonging to the only subpopulation that matches the current state put actions in the action set, whereas in fuzzy LCS, we have many subpopulations matching the same crisp state, each proposing an action with some strength, proportional to the degree of matching of the classifier selected within the subpopulation. The strength values of the proposed actions are composed by an *aggregation* operator, usually implemented by a *T-conorm* (such as *max*), and then defuzzyfied by one of the many methods available [46]. Whatever *T-norm*, *T-conorm*, and defuzzyfication method is selected, the resulting action proposed by the set of selected fuzzy classifiers $\bar{a}$ is a function of the actions a^i proposed by the matching classifiers and of their degrees of matching $\mu_{\tilde{s}^i}$.

Since all the matching classifiers contribute to the performance of a LFCS, the reinforcement should be distributed to all of them, proportionally to their

contribution. Therefore, the first step to extend the reinforcement distribution algorithms consists of considering $\mu_{\tilde{s}^i}$ as a weight for the contribution of each classifier. If we want to make this weight adapting to the learning process evolution, we may introduce a factor (let us call it ξ_{c^i}) that considers also that the contribution is relevant proportionally to the sum of the past contributions given by that classifier to the performance of the system. For each step, this factor is equal to the current contribution of the classifier (its degree of matching), weighted by the sum of the contribution given till now, saturated to a given threshold T. This enhances the learning rate of classifiers that do not have participated too much to the output, yet.

$$\xi_{c^i} = \frac{\mu_{\tilde{s}^i}(s_t)}{min\left(T, \sum_{k=1,t}\, \mu_{\tilde{s}^i}(s_k)\right)}$$

The other important point concerns the evaluation of the best value in a given state, used in the $Q-learning$ and $TD(\lambda)$ formulas. In this case, we cannot just take the value corresponding to the best value in the given state, since this state matches many fuzzy states, each contributing with different actions. We present our proposal for Q-learning, leaving to the reader the analogous passages for $TD(\lambda)$. Let us consider $Q^*(c_t^{*i})$, the highest values of the classifiers c_t^{ji} belonging to each subpopulation corresponding to the matching fuzzy states $\tilde{s}^i$ at time t.

$$Q^*(c_t^{*i}) = max_j(Q(c_t^{ji}))$$

Let us call $m_t(s)$ the sum of the $Q^*(c_t^{*i})$, each weighted by $\mu^*_{\tilde{s}^i}(s)$, the degree of matching between the corresponding classifier and the current, real-valued state s. This means that we consider as the reference best value in the current state, the combination of the best classifiers that can trigger in this state, one for each matching subpopulation. The corresponding formula is:

$$m_t(s) = \sum_{k=1,K} \mu^*_{\tilde{s}^k}(s)\, Q^*(c_t^{*k})$$

Fuzzy Q-Learning At time t the system is in the state s_t, after having executed action a_{t-1} from a state s_{t-1} matched by the fuzzy antecedent $\tilde{s}_{t-1}$, and receives the reinforcement r_t. The Q-value given to the classifiers c^i_{t-1} selected by the policy for each pair $\langle \tilde{s}^i_{t-1}, a^i_{t-1}\rangle$ is updated by the formula:

$$Q_t(c^i_{t-1}) = Q_{t-1}(c^i_{t-1}) + \alpha\, \xi_{c^i_{t-1}}\, \left(r_t + \gamma\, m_{t-1}(s_{t-1}) - Q_t(c^i_{t-1})\right)$$

The accuracy of the classifier c^i_{t-1} is updated by the formula:

$$E_t(c^i_{t-1}) = E_{t-1}(c^i_{t-1}) + \omega\, \xi_{c^i_{t-1}}\, (|r_t + \gamma\, m_{t-1}(s_{t-1}) - Q_t(c^i_{t-1})| - E_{t-1}(c^i_{t-1}))$$

Fuzzy $TD(\lambda)$ The extension of the $TD(\lambda)$ algorithm presented for crisp LCS is straightforward, given the considerations done at the beginning of this section. We do not present again all the steps, but only the key formulas.
The eligibility trace of the fuzzy state $\tilde{s}$ is updated by the formula:

$$e_t(\tilde{s}^i) = \begin{cases} \mu_{\tilde{s}^i}(s) & if\ \tilde{s}_t^i = \tilde{s}_{t-1}^i \\ \gamma\lambda\mu_{\tilde{s}^i}(s)e_{t-1}(\tilde{s}^i) & otherwise \end{cases}$$

The strength of the classifier c_{t-1}, is updated by the formula:

$$w_t(c_{t-1}^i) = w_{t-1}(c_{t-1}^i) + \beta\,\xi_{c_{t-1}^i}(r_t + \gamma \sum_{\forall i} \mu_{\tilde{s}^i}(s_t)V(\tilde{s}^i) - V_{t-1}(\tilde{s}_{t-1}))$$

The accuracy of the classifier c_{t-1}^i is updated by:

$$E_t(c_{t-1}^i) = E_{t-1}(c_{t-1}^i) + \omega\,\xi_{c_{t-1}^i}(|\sum_{\forall i} \mu_{\tilde{s}^i}(s_t)V(\tilde{s}^i) - V_{t-1}(\tilde{s}_{t-1})| - E_{t-1}(c_{t-1}^i))$$

Fuzzy Bucket Brigade At time t the reinforcement distribution algorithm executes these steps.
1. Receive the eventual reinforcement r_{t-1} due to the action a_{t-1} selected by the policy in state s_{t-1} and the new action a_t selected by the policy to be done in the current state s_t.
2. Compute the bid basing on the action sets of each subpopulation:

$$Bid_t = \sum_{\forall i, \forall c_t^i \in A_t^i} \beta\mu_{\tilde{s}^i}(s)w_t(c_t^i)$$

3. $\forall i, \forall c_t^i \in A_t^i$ reduce the strength of c_t^i, by a quantity equal to its contribution to the bid:

$$w_t(c_t^i) = w_{t-1}(c_t^i) + (1 - \beta w_t \mu_{\tilde{s}^i}(s))$$

4. Distribute the bid to the classifiers belonging to the action set of the previous step $\forall i, \forall c_{t-1}^i \in A_{t-1}^i$

$$w_t(c_{t-1}^i) = w_{t-1}(c_{t-1}^i) + \xi_{c_{t-1}^i}\,(Bid_t + r_t)$$

5. Update the accuracy of any classifier c_{t-1} in A_{t-1} by:

$$E_t(c_{t-1}^i) = E_{t-1}(c_{t-1}^i) + \omega\,\xi_{c_{t-1}^i}\,(|w_t(c_{t-1}^i) - w_{t-1}(c_{t-1}^i)| - E_{t-1}(c_{t-1}^i))$$

6 Conclusions

We have presented a picture of the research in Learning Fuzzy Classifier Systems that gives an idea of the contribution the results obtained in this field can give both to the application side and to the theoretical aspects of Learning Classifier Systems, and Reinforcement Learning in general. A lot of work has to be done, yet, to deeply understand all the potentiality of FLCS, and to build a methodological background to make the application of FLCS easy in real world problems.

Acknowledgments. This research is partially supported by the Politecnico di Milano Research Grant "Development of autonomous agents through machine learning", and partially by the project "CERTAMEN" co-funded by the Italian Ministry of University and Scientific and Technological Research. I have to thank some of my former students who have dedicated a significant part of their lives to work on this topic, in particular: F. Giorlandino, A. M. Carpani, G. Calegari, F. Bortolotti, N. De Marco, F. Basso, C. Bonacina, M. Matteucci

References

[1] H. Berenji. Fuzzy q-learning: a new approach for fuzzy dynamic programming. In *Proceedings Third IEEE International Conference on Fuzzy Systems*, pages 486–491, Piscataway, NJ, 1994. IEEE Computer Press.

[2] A. Bonarini. ELF: Learning incomplete fuzzy rule sets for an autonomous robot. In H.-J. Zimmermann, editor, *First European Congress on Fuzzy and Intelligent Technologies – EUFIT'93*, volume 1, pages 69–75, Aachen, 1993. Verlag der Augustinus Buchhandlung.

[3] A. Bonarini. Delayed reinforcement, fuzzy Q-learning and fuzzy logic controllers. In Herrera and Verdegay [26], pages 447–466.

[4] A. Bonarini. Evolutionary learning of fuzzy rules: competition and cooperation. In W. Pedrycz, editor, *Fuzzy Modelling: Paradigms and Practice*, pages 265–284. Kluwer Academic Press, Norwell, MA, 1996.

[5] A. Bonarini. Anytime learning and adaptation of hierarchical fuzzy logic behaviors. *Adaptive Behavior Journal*, 5(3–4):281–315, 1997.

[6] A. Bonarini. Reinforcement distribution to fuzzy classifiers: a methodology to extend crisp algorithms. In *IEEE International Conference on Evolutionary Computation – WCCI-ICEC'98*, volume 1, pages 51–56, Piscataway, NJ, 1998. IEEE Computer Press.

[7] A. Bonarini. Comparing reinforcement learning algorithms applied to crisp and fuzzy learning classifier systems. In *Proceedings of the Genetic and Evolutonary Computation Conference –GECCO99*, pages 52–59, San Francisco, CA, 1999. Morgan Kaufmann.

[8] A. Bonarini and F. Basso. Learning to to coordinate fuzzy behaviors for autonomous agents. *International Journal of Approximate Reasoning*, 17(4):409–432, 1997.

[9] A. Bonarini, C. Bonacina, and M. Matteucci. A framework to support the reinforcement function design in real-world agent-based applications. Technical Report 99-73, Politecnico di Milano - Department of Electronics and Information, Milan, I, 1999.

[10] A. Bonarini, C. Bonacina, and M. Matteucci. Fuzzy and crisp representation of real-valued input for learning classifier systems. In *this book*, 2000.

[11] L. Booker. Classifier systems that learn internal world models. *Machine Learning*, 1(2):161–192, 1988.

[12] L. Booker, D. E. Goldberg, and J. H. Holland. Classifier systems and genetic algorithms. *Artificial Intelligence*, 40(1–3):235–282, 1989.

[13] B. Carse, T. Fogarty, and A. Munro. Evolving fuzzy rule based controllers using genetic algorithms. *Fuzzy Sets and Systems*, 80(3):273–293, 1996.

[14] K. C. C. Chan, V. Lee, and H. Leung. Generating fuzzy rules for target tracking using a steady-state genetic algorithm. *IEEE Transactions on Evolutionary Computation*, 1(3):189–200, 1997.

[15] Sinn-Cheng Lin; Yung-Yaw Chen. A ga-based fuzzy controller with sliding mode. In *Proceedings of the IEEE International Conference on Fuzzy Systems*, pages 1103–1110, Piscataway, NJ, 1995. IEEE Computer Press.

[16] Sinn-Cheng Lin; Yung-Yaw Chen. On ga-based optimal fuzzy control. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 846–851, Piscataway, NJ, 1995. IEEE Computer Press.

[17] M. Cooper and J. Vidal. Genetic design of fuzzy controllers: The cart and jointed-pole problem. In *Proceedings Third IEEE International Conference on Fuzzy Systems*, pages 1332–1337, Piscataway, NJ, 1994. IEEE Computer Press.

[18] M. Dorigo and M. Colombetti. *Robot shaping: an experiment in behavior engineering.* MIT Press / Bradford Books, Cambridge, MA, 1997.

[19] D. Dubois and H. Prade. *Fuzzy Sets and Systems: Theory and Applications.* Academic Press, London, 1980.

[20] T. Furuhashi, K. Nakaoka, K. Morikawa, and Y. Uchikawa. Controlling execessive fuzziness in a fuzzy classifier system. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 635–642, San Mateo, CA, 1993. Morgan Kaufmann.

[21] T. Furuhashi, K. Nakaoka, and Y. Uchikawa. An efficient finding of fuzzy rules using a new approach to genetic based machine learning. In *Proceedings Fourth IEEE International Conference on Fuzzy Systems*, pages 715–722, Piscataway, NJ, 1995. IEEE Computer Press.

[22] P.-Y. Glorennec. Fuzzy Q-learning and evolutionary strategy for adaptive fuzzy control. In H.-J. Zimmermann, editor, *Second European Congress on Intelligent Techniques and Soft Computing - EUFIT'94*, volume 1, pages 35–40, Aachen, D, 1994. Verlag der Augustinus Buchhandlung.

[23] P. Y. Glorennec. Constrained optimization ofFIS using an evolutionary method. In Herrera and Verdegay [26], pages 349–368.

[24] P.Y. Glorennec. Fuzzy Q-learning and dynamical fuzzy Q-learning. In *Proceedings Third IEEE International Conference on Fuzzy Systems*, Piscataway, NJ, 1994. IEEE Computer Press.

[25] D. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3(5):493–530, 1989.

[26] F. Herrera and J.L. Verdegay, editors. *Genetic Algorithms and Soft Computing (Studies in Fuzziness, 8).* Physica Verlag (Springer Verlag), Heidelberg, D, 1996.

[27] F. Hoffmann and G. Pfister. Learning of a fuzzy control rule base using messy genetic algorithms. In Herrera and Verdegay [26], pages 279–305.

[28] Frank Hoffmann, Oliver Malki, and Gerd Pfister. Evolutionary algorithms for learning of mobile robot controllers. In H.-J. Zimmermann, editor, *Fourth European Congress on Fuzzy and Intelligent Technologies – EUFIT'96*, volume 2, pages 1105–1109, Aachen, D, 1996. Verlag der Augustinus Buchhandlung.

[29] Frank Hoffmann and Gerd Pfister. Automatic design of hierarchical fuzzy controllers using genetic algorithm. In H.-J. Zimmermann, editor, *Second European Congress on Fuzzy and Intelligent Technologies – EUFIT'94*, volume 3, pages 1516–1522, Aachen, September 1994. Verlag der Augustinus Buchhandlung.

[30] J. Holland. Properties of the bucket-brigade algorithm. In J. J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, pages 1–7., Hillsdale, NJ, 1985. Lawrence Erlbaum Associates.

[31] J. Holland and J. Reitman. Cognitive systems based on adaptive algorithms. In D. A. Waterman and F. Hayes-Roth, editors, *Pattern-directed inference systems*, New York, NY, 1978. Academic Press.

[32] H.-S. Hwang and K.-B. Woo. Linguistic fuzzy model identification. *IEE Proceedings on control theory applications*, 142(6):537–545, 1995.

[33] W. Hwang and W. Thompson. Design of fuzzy logic controllers using genetic algorithms. In *Proceedings of the Third IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'94)*, pages 1383–1388, Piscataway, NJ, 1994. IEEE Computer Press.

[34] H. Ishibuchi, T. Nakashima, and T. Murata. A fuzzy classifier system for generating linguistic classification rules. In *Proceedings IEEE/Nagoya University WWW'95*, pages 22–27, Nagoya, J, 1995.

[35] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[36] C. L. Karr. Applying genetics to fuzzy logic. *AI Expert*, 6(3):38–43, 1991.

[37] C. L. Karr. Design of a cart-pole balancing fuzzy logic controller using a genetic algorithm. In *Proceedings SPIE Conference on the Applications of Artificial Intelligence*, pages 26–36, 1991.

[38] C. L. Karr. Adaptive process control with fuzzy logic and genetic algorithms. *Sci. Comput. Autom.*, 9(10):23–30, 1993.

[39] C. L. Karr. Real time process control with fuzzy logic and genetic algorithms. *Proceedings of the Symposium on Emerging Computer Techniques for the Minerals Industry*, pages 31–37, 1993.

[40] C. L. Karr, L. M. Freeman, and D. L. Meredith. Improved fuzzy process control of spacecraft autonomous rendezvous using a genetic algorithm. In G. Rodriguez, editor, *Intelligent Control and Adaptive Systems*, volume 1196, pages 274–288, Philadelphia, 1989. The International Society of Photo-Optics Instrumentation Engineers (SPIE).

[41] C. L. Karr and E. J. Gentry. Fuzzy control of pH using genetic algorithms. *IEEE Transactions on Fuzzy Systems*, 1(1):46–53, 1993.

[42] C. L. Karr, S. K. Sharma, W. J. Hatcher, and T. R. Harper. Fuzzy control of an exothermic chemical reaction using genetic algorithms. In *Engineering Applications of Artificial Intelligence 6*, volume 6, pages 575–582, 1993.

[43] J. Y. Ke, K. S. Tang, and K. F. Man. Genetic fuzzy classifier for benchmark cancer diagnosis. In *Proceedings of the 23rd International Conference on Industrial Electronics, Control and Instrumentation (IECON97*, volume 3, pages 1063–1067, 1997.

[44] M. Kingham and M. Mohammadian. Financial modelling and prediction of interest rate using fuzzy logic and genetic algorithms. In *Proceedings of the Australian and New Zealand Conference on Intelligent Information Systems*, volume 2, pages 233–236, Piscataway, NJ, 1996. IEEE Computer Press.

[45] J. Kinzel, F. Klawonn, and R. Kruse. Modifications of genetic algorithms for designing and optimizing fuzzy controllers. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, volume 1, pages 28–33, Piscataway, NJ, 1994. IEEE Computer Press.

[46] G. J. Klir, B. Yuan, and U. St. Clair. *Fuzzy set theory: foundations and applicatons.* Prentice-Hall, Upper Saddle River, NY, 1997.

[47] M. Lee and H. Takagi. Hybrid genetic-fuzzy systems for intelligent systems design. In Herrera and Verdegay [26], pages 226–250.

[48] M. A. Lee and H. Takagi. Dynamic control of genetic algorithms using fuzzy logic techniques. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 76–93, San Mateo, CA, 1993. Morgan Kaufman.

[49] M. A. Lee and H. Takagi. Integrating design stages of fuzzy systems using genetic algorithms. In *Proceedings of the IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'93)*, pages 612–617, Piscataway, NJ, 1993. IEEE Computer Press.

[50] D. Leitch. Genetic algorithms for the evolution of behaviors in robotics. In Herrera and Verdegay [26], pages 306–328.

[51] D. Leitch and P. Probert. Genetic algorithms for the development of fuzzy controllers for autonomous guided vehicles. In H.-J. Zimmermann, editor, *Second European Congress on Fuzzy and Intelligent Technologies – EUFIT'94*, volume 1, pages 464–469, Aachen, September 1994. Verlag der Augustinus Buchhandlung.

[52] D. Leitch and P. Probert. Genetic algorithms for the development of fuzzy controllers for mobile robots. In *Lecture Notes in Computer Science*, volume 1011, pages 148–162, 1995.

[53] D. Leitch and P. Probert. New techniques for genetic development of fuzzy controllers. *IEEE Transactions on Systems, Man and Cybernetics*, 28(1):112–123, 1998.

[54] J. Liska and S. Melsheimer. Complete design of fuzzy logic systems using genetic algorithms. In *Proceedings of the Third IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'94)*, volume II, pages 1377–1382, Piscataway, NJ, 1994. IEEE Computer Press.

[55] E. Mamdani and S. Assilian. An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Man-Machine Studies*, 7:1–13, 1975.

[56] M. Mohammadian and R. J. Stonier. Adaptive two layer fuzzy control of a mobile robot system. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, page 204, Piscataway, NJ, 1995. IEEE Computer Press.

[57] H. Nomura, I. Hayashi, and N.Wakami. A self-tuning method of fuzzy reasoning by genetic algorithm. In *Proceedings of the International Fuzzy Systems and Intelligent Control Conference (IFSICC'92)*, pages 236–245, Louisville, KY, 1992.

[58] J. Ohwi, S.V. Ulyanov, and K. Yamafuji. Ga in continuous space and fuzzy classifier system for opening of door with manipulator of mobile robot: new benchmark of evolutionary intelligent computing. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 257–261, Piscataway, NJ, 1995. IEEE Computer Press.

[59] G. Ortega and J.M. Giron-Sierra. Genetic algorithms for fuzzy control of automatic docking with a space station. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 157–161, Piscataway, NJ, 1995. IEEE Computer Press.

[60] D. Park, A. Kandel, and G. Langholz. Genetic-based new fuzzy reasoning models with application to fuzzy control. *IEEE Transactions on Systems, Man and Cybernetics*, 24(1):39–47, 1994.

[61] A. Parodi and P. Bonelli. A new approach to fuzzy classifier systems. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 223–230, San Mateo, CA, 1993. Morgan Kaufman.

[62] A. Rahmoun and M. Benmohamed. Genetic algorithm methodology to generate optimal fuzzy systems. *IEE Proceedings on control theory applications*, 145(6):583–587, 1998.

[63] S. P. Singh and R. S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123–158, 1996.
[64] S. F. Smith. *A learning system based on genetic adaptive algorithms*. PhD thesis, University of Pittsburgh, Pittsburgh, PE, 1980.
[65] R. S. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3(1):9–44, 1988.
[66] R. S. Sutton and A. G. Barto. *Reinforcement Learning An Introduction*. MIT Press, Cambridge, Massachusetts, 1999.
[67] T. Takagi and M. Sugeno. Fuzzy identification of systems and its applications to modelling and control. *IEEE Transactions on Systems, Man and Cybernetics*, 15(1):116–132, 1985.
[68] K. Tang, K. Man, Z. Liu, and S. Kwong. Minimal fuzzy memberships and rules using hierarchical genetic algorithm. *IEEE Transactions on Industrial Electronics*, 45(1):162–169, 1998.
[69] P. Thrift. Fuzzy logic synthesis with genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 509–513, San Mateo, CA, 1993. Morgan Kaufman.
[70] Manuel Valenzuela-Rendón. The fuzzy classifier system: A classifier system for continuously varying variables. In R. Belew and L. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 346–353, San Mateo, CA, 1991. Morgan Kaufman.
[71] C. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.
[72] S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.
[73] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

Fuzzy and Crisp Representations of Real-Valued Input for Learning Classifier Systems

Andrea Bonarini, Claudio Bonacina, and Matteo Matteucci

Politecnico di Milano AI and Robotics Project,
Dipartimento di Elettronica e Informazione, Politecnico di Milano
Piazza Leonardo da Vinci, 32 , 20133 Milano, Italy
bonarini@elet.polimi.it,
WWW home page: http://www.elet.polimi.it/people/bonarini

Abstract. We discuss some issues concerning the application of learning classifier systems to real-valued applications. In particular, we focus on the possibility of classifying data by crisp and fuzzy intervals, showing the effect of their granularity on the learning performance. We introduce the concept of *sensorial cluster* and we discuss the difference between *cluster aliasing* and *perceptual aliasing*. We show the impact of different choices on the performance of both crisp and fuzzy learning classifier systems applied to a mobile, autonomous, robotic agent.

1 Introduction

There is an increasing interest in the Learning Classifier Systems (LCS) community about issues related to the application of LCS to real-valued environments. Most of the LCS applications proposed up to now work on gridworlds [14] or an interval-based representation of the input [7]. When interfacing to the real world by real-valued sensors, the selection of the interval granularity becomes a relevant issue. A fine-grained classification translates in a large search space, and a coarse-grained classification tends to induce *perceptual aliasing* [13]. We discuss some criteria to face this trade-off in section 2, where we also discuss the relationships between the interval-based representation partitioning the sensor data space, and the traditional gridworld, where the partition is done on spatial dimensions, strictly related to the *configuration space* (*c-space*) [9]. We remind the reader here that the c-space is the space of the variables needed to completely describe the relevant aspects of a system, in general, and of a robot, in our case. In section 3, we propose to adopt fuzzy intervals as a representation of the classifier input [2] [3]. Considering an interval-based model and a fuzzy one with the same number of intervals, the information content of a fuzzy representation is very close to that of a real-valued representation, and considerably higher than that of an interval-based. Moreover, the selection of certain well-known configurations of fuzzy intervals (discussed in Section 3) guarantees high robustness to noise and certain design errors [8]. However, the introduction

P.L. Lanzi, W. Stolzmann, and S.W. Wilson (Eds.): LCS '99, LNAI 1813, pp. 107–124, 2000.

of partially overlapping fuzzy intervals raises some learning problems that can be reduced by a careful redesign [4] of the classical reinforcement distribution algorithms. In section 4, we present the simplified learning classifier system that we have devised to test our proposal. In Section 5, we show the results we have obtained on a simulated mobile robot, where we consider most of the real world features, including stochastic and systematic noise, approximation, and uncertainty in sensors and actuators. The aim of this paper is to discuss the issues concerning learning crisp and fuzzy interval representations when operating with real-valued input/output; in particular we do not introduce new RL algorithms; they are presented and compared with other proposals elsewhere [6], [1], [2], [3], [4], nor we discuss about genetics, generalization, or the reinforcement function in our proposal. The considerations we present in this paper are general enough to be relevant for any RL algorithm operating on real values.

2 Grids, Granularity, and Real Values

The information needed by a LCS is usually discretized, in order to have classifiers working on a finite set of input and output values. Most of the work done in the past in this field is based on gridworlds, where an *animat* [14] moves from one cell to another connected to the first, and perceives the presence of elements fully occupying cells. The kind of discretization we are proposing in this paper is different, since it is done on the information coming from sensors. We call the set of real values, one for each input variable, a *real-valued state*. We classify the real input values into classes corresponding to either crisp or fuzzy intervals. In figure 1 you may see such a classification for the distance measured by a sonar sensor. On the ordinates the degree of membership μ of the distance reported on the abscissas in each interval (below in the figure) or fuzzy set (above).

We call the set of crisp intervals, one for each input variable, matching a real-valued state, a *crisp state*. We call the set of fuzzy sets, one for each input variable, matching a real-valued state, a *fuzzy state*. In the following, we will refer to both these interpretations of a real-valued state with the common term *state*. The main point is that this discretization is done on the animat's perception, whereas the discretization in gridworlds is usually done on the environment where the animat operates, which corresponds, in general, to a, eventually partial, view of the c-space.

The other main difference concerns the type of movement the animat can do. In gridworlds, it can move from one cell to another (taking one of at most 8 possible actions from one position), in a real-valued world it can change its position in the c-space (with a real-valued move), but this may, or may not, bring the animat to a different state. The shift from a state to another is not linearly related to the movement. Depending on the sensors, their distribution, the classification model (and on the position in the environment, of course), the same movement may either bring the animat in a different state or not. As we show in section 5, the fact that an action brings it quite often to a different state heavily influences the learning performance. The relative frequency of state

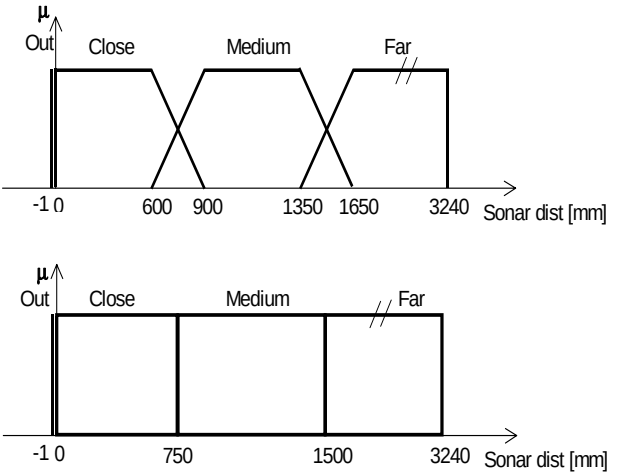


Fig. 1. The fuzzy membership functions (above) and the intervals (below) to interpret the distance measured by a sonar sensor.

change depends both on the granularity of the classification of the input space, and on the duration of the perceive-act (or *control*) step. This is the second type of discretization we need to do. In gridworlds, the duration of the control step is just one tick of an ideal clock; things may change at each tick, and, usually, the animat can move, in a tick, only from one cell to a connected one: its speed is one cell per tick. In the real world, we have real time, and real-valued output (in our case, speed and steering). The control step duration has a lower bound in the physical time needed to perceive (i.e., acquire and elaborate data from sensors), select the action, and send it to the actuators. The actual duration of the control step may be selected as a trade-off between the need for fast reaction, to select at each moment the best action, and the desire of changing state at each step; this improves the effectiveness of the learning process since reduces cluster aliasing (see section 3) and enhances the effect of the exploration policy. The results we present in section 5 show the impact of the length of the control step on the learning activity. This becomes even more important when the LCS receives so-called *continuous reinforcement*, that is when it is reinforced at each control step. If we call the time interval between two reinforcements an *evaluation step*, we may say that, with continuous reinforcement, the control step is identical to the evaluation step. A possibility to keep a fast reaction, but also a good chance of changing state between two subsequent evaluation steps, is to select an evaluation step larger than the control step. A first intuition about the importance of this problem was already present in [6] [2], where the term *episode* was used to name a sequence of control steps at the end of which a reinforcement is given. An even more radical approach is taken by many researchers, who evaluate the performance only when the system reaches a different interval (or

fuzzy) state. The difference with respect to the episode approach is that, during an episode, the system may visit different states, while here the performance is evaluated every time the system changes state. In both cases, the action selected for a state is kept the same until the performance is evaluated, at the end of an episode, or when the system enters a new state.

Another important problem related to discretization concerns the *aliasing* phenomenon. Let us call the set of real values belonging to the same (crisp or fuzzy) interval state a *sensorial cluster*. All the real-valued states belonging to a sensorial cluster are classified in the same state, and correspond to the same subpopulation of classifiers. From each real-valued state it is possible to select one of the competing classifiers in the subpopulation matching the state, thus obtaining, in general, different reinforcement. Notice that the same classifier may be triggered by any of the real-valued states belonging to the sensorial cluster, and that, in general, the reinforcement function varies inside a sensorial cluster. Let us call the situation where it is possible to do an action that does not bring us out of the sensorial cluster *cluster aliasing*. This may cause problems to the learning algorithm since the same classifiers may have different reinforcements. The larger the cluster, the higher the possibility to obtain different reinforcement for the classifiers belonging to the corresponding subpopulation, and the lower is the accuracy of the LCS to learn the correct action to take from a sensorial cluster. This is another motivation to keep the cluster (and the corresponding state) as small as possible. Let us notice that cluster aliasing is slightly different from *perceptual aliasing* [13], defined as having the same internal representation for different states. Perceptual aliasing concerns the representation of the percepts, cluster aliasing concerns the fact that actions done in a state do not bring us out from it. In perceptual aliasing the problem is that there is the possibility to take the same reinforcement in different situations, in cluster aliasing is the opposite: taking different reinforcements for the same classifier causes problems to the learning activity.

3 Motivations for Fuzzy Intervals

Since their introduction in the late sixties [15], *fuzzy sets* have been adopted to map real numbers to symbolic labels. Elements of a universe of discourse belong to a fuzzy set to a certain extent, according to the so-called *membership function* that defines it. Let us consider a universe of discourse X (say, an interval of real numbers), and a fuzzy set, associated to the label *close*, defined by a membership function $\mu_{close}(.)$ that ranges on elements of the universe of discourse and maps them to the real interval $[0 \quad 1]$ (see figure 1). We can say that an element $\bar{x}$ of the universe of discourse (in our case a distance in millimeters) belongs to the fuzzy set *close* to the extent $\mu_{close}(\bar{x})$. For the fuzzy set *close* in figure 1, $\mu_{close}(690) = 0.7$. Giving a semantic value to the fuzzy set, we can say that the value $\bar{x} = 690$ can be classified as *close* with a degree of 0.7.

Fuzzy sets are often used to classify real values in categories, thus making possible symbolic reasoning about the phenomena that underlie the incoming

numbers. Membership of a number (e.g., $\bar{x}$) to a fuzzy set (e.g., *close*) includes two kinds of information: the *name* of the fuzzy set, that brings information about the category to which the number is classified, and the *degree of membership* (e.g., $\mu_{close}(\bar{x})$) that comes from the definition of the fuzzy set. The membership value can be considered as an alternative representation of $\bar{x}$, although it is unique only for the class of monotonic membership functions. In general, the relationship between a real value $\bar{x}$ and its degree of membership to a fuzzy set $\bar{l}$, $\mu_{\bar{l}}(\bar{x})$, is not bijective. This potential problem is solved by defining partially overlapping fuzzy sets, and considering as representative of $\bar{x}$ the set of all the values $\mu_l(\bar{x})$, $\forall\ l$.

Fuzzy rules represent relationships among fuzzy sets. Since the mapping between a number and its classification is crisp and well-defined, we can say that a set of fuzzy rules is a compact representation of a relationship between the real values that can be classified by the fuzzy sets in their antecedents and consequents. A set of fuzzy rules implement a mapping among real values that combines the mappings implemented by each rule. So, a fuzzy LCS is a way to consider different local models (fuzzy rules) to obtain a complex, in general non linear, mapping [1] among real values. In particular, it is common to consider partially overlapping fuzzy sets, as the ones represented in figure 1. In this case, any set of real values for the input variables is matched to some extent by at least two rules. In particular, it can be demonstrated that if the sum of the membership values equals 1 for any value in the range of the variable, the robustness of the system with respect to noise is optimal [8]. To summarize: fuzzy sets can provide an alternative representation of real numbers, a fuzzy rule implements a model that maps real values to real values in given intervals, a set of fuzzy rules implement a complex mapping on the full range of the involved variables. These are the main reasons why fuzzy LCS are an interesting approach to learn relationships among real values.

With respect to a real-valued representation, fuzzy LCS introduce an abstraction level that reduces the search space and makes it possible to reason and learn on symbolic models. The accent on local models (shared with the interval representation) implies the possibility to learn by focusing at each step on small parts of the search space only. An interesting aspect of the fuzzy interpretation, when compared with the interval-based, is the possibility to have a smooth transition between close models: a real value in input is interpreted in a way similar to the close values in any region of the input range, while an even small difference in values across the crisp interval boundaries gives rise to radically different interpretations. The smooth transition among different models implemented by fuzzy rules implies robustness with respect to data noise [8].

This aspect has some impact also on the learning process. The interaction among local models, due to the intersection of neighbor fuzzy sets guarantees that local learning (in the *niche* corresponding to a fuzzy state) reflects on global performance [1], since the classifiers are judged for their performance not only inside the niche, but also on the boundaries shared with neighbor niches. In other terms, fuzzy classifiers are in competition with each other inside their

niche, but have to cooperate with classifiers belonging to neighbor niches. It is not so for traditional crisp LCS, where competition is local, and cooperation on the output generation is not present, since the selected action is proposed by a single classifier.

Another interesting implication for learning is the increased granularity that the fuzzy representation induces. Let us consider figure 2, where we represent the interval (above) and the fuzzy interpretations (below) for data coming from a sonar sensor, mapped on the c-space of our robot. The c-space is defined by the two variables: d, the distance from the left wall of the corridor, and θ, the heading of the robot.

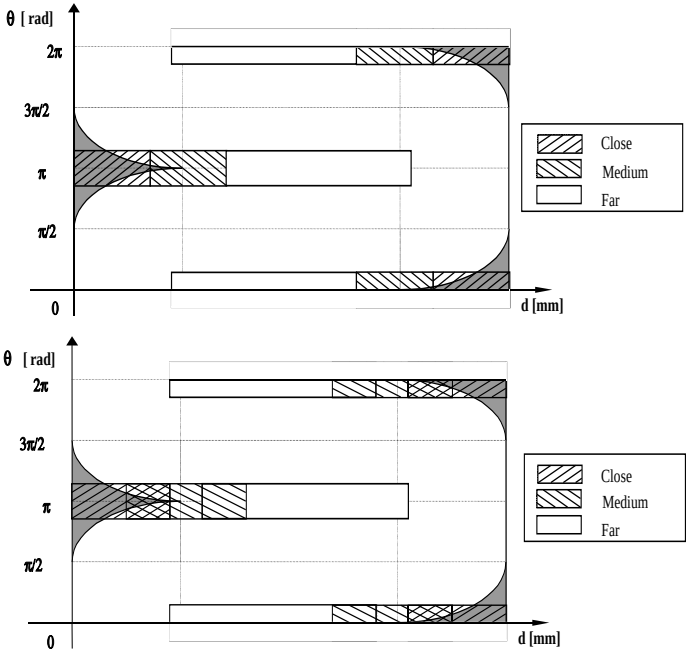


Fig. 2. The crisp (above) and fuzzy (below) interpretations of data from one sonar reported on the c-space.

Note that, since we consider the corridor as infinite, only lateral distances, such as d are relevant. From the dark gray areas of the c-space the robot can only bump on the wall, due to its limited steering ability (see Section 5.1). For each position of the robot in its c-space we have drawn the interpretation of data coming from the selected sonar (hatched rectangles in the figure - in general, the complete state is given by hypercubes). We see that each crisp interpretation is partitioned by the fuzzy one in many zones. This means that there are real-valued states that belong to one crisp state, but, at the same time, to a certain number of fuzzy states (two in the figure) with respective degrees of membership.

In other terms, although the number of classes is the same for both crisp and fuzzy classifications, this last partitions the input space in a larger number of distinct zones (five for each triple of classes in figure 2).

When the agent is in a real-valued state, it matches one crisp state, but more than one fuzzy state. Thus, the reinforcement obtained is distributed to a state (or an action done in a state) in the first case, and to a set of states (actions) in the second case. This means that the information obtained by the environment is distributed at each step to classifiers related to many states in the fuzzy approach. This increases the possibility to evaluate actions for real-valued states which can hardly been reached by the interval representation only. Let us consider, for instance, the *close* interval partially covering the "*no return*" zone of the c-space marked in dark gray in figure 2. It covers also a zone that the animat can leave, but this is very small, and the probability of visiting it is low, although the animat might learn interesting actions by going there. The same area is covered by fuzzy states that extend, with decreasing membership functions, also on the neighborhood. In the zone where membership functions overlap the animat movement comes from the composition of the action proposed by classifiers belonging to different subpopulations. In a sense, this reduces cluster aliasing, since actions done in the overlapping zone are in general different from actions proposed where there is no overlapping. Moreover, they are different with a kind of continuous shift from the action proposed in a state (e.g., *medium* in the figure) to that proposed in the other (e.g., *close*). Learning also takes into account the actions done in the overlapping areas, since reinforcement is also given because of actions done from that area. From figure 2, it is possible to imagine that classifiers operating from the *medium* state may interact positively with those (possibly highly inaccurate) operating from the *close* state in the overlapping area, possibly keeping the agent out from the *"no return"* zone.

Now, let us make some few remarks. Given the above mentioned phenomenon, it is easy to observe that fuzzy LCS converge to behaviors that keep the animat close to the overlapping zones. This means that the learning algorithms we have introduced tend to optimize the interactions among close states. This may be a suggestion for the design of membership functions. A second remark concerns the complexity of the states we have even in a simple application as the one we introduce in section 5. For instance, we have six sonars corresponding to six patterns analogous to those shown in figure 2, covering the c-space with many overlapping zones. We have found that, just by fuzzyfying the intervals as shown in figure 1, the number of the actual states reachable by our animat rises from 84 to 112, on the same c-space. This reduces cluster aliasing, per se, but also gives an idea of the number of the overlapping zones that may correspond to these fuzzy states. As a last remark, let us notice that, on the boundaries between fuzzy states, the contribution of the obtained reinforcement is mediated by the corresponding membership functions, thus propagating the smoothing effect, mentioned above with respect to the actions, also to the learning process.

4 A Simple LCS Framework

In this paper, we consider a simplified type of LCS [3]. Each classifier is a rule, whose antecedent is the conjunction of symbolic values for input variables. In *crisp LCS*, we consider that each of these symbols denotes an interval of real input values for the corresponding variable. In *fuzzy LCS*, each symbol denotes a fuzzy subset of the range of the real values the variable can take. A *don't care* symbol may replace any of the antecedent values, meaning that the value of the variable is not relevant for the classifier, that is that any real value matches the antecedent. The consequents are symbols corresponding to crisp values for each of the consequent variables. A classifier of this kind has a conceptual shape like this:

```
IF (FrontDistance IS Close)
   AND (LeftDistance IS Far)
   AND (RightDistance IS Don_T_Care)
THEN (TurnDirection IS Left)
```

actually implemented in a more compact string such as "130:3".

At each control step, one crisp classifier is selected among the ones whose antecedent is the crisp state matching the current real-valued state. The consequent values of the selected classifier are sent to the actuators of the robot.

When operating with fuzzy LCS, a real-valued state matches different fuzzy states, i.e., different fuzzy classifiers. We consider subpopulations of classifiers having the same antecedent (eventually including don't cares) and different consequents. At each control step, we select one classifier for each matching subpopulation (i.e., one for each fuzzy state). Each of the matching classifiers proposes an action with a weight proportional to its degree of matching. We combine the proposed actions with a fuzzy aggregation operator [8] (in the experiments here reported, *max*), we defuzzyfy the result [8] (here, by a *weighted sum*), and we send the obtained real value to the actuators. Notice that the output of crisp classifiers is selected in a small range of values; for instance, if we have only one output variable, the output is one of the k values defined for that variable. The output of fuzzy classifiers has a far larger range, since it comes from the weighted combination of the same number of output values (figure 3): the defuzzyfication process makes it possible to exploit the information about membership of the real-valued state to the fuzzy antecedent, thus producing a fine-grained output. Notice that this is not a real-valued output, only because the number of output values depends on the precision considered for the membership values.

We have implemented a tool [4] to study the impact on the learning process of different credit assignment algorithms, exploration strategies, and evolution algorithms. We have done many experiments exploiting the different choices offered by our tool, and the results of this extensive study will be presented in other papers. Here, we present results concerning the paper theme, and obtained by our versions of *Q-Learning* [12] and $TD(\lambda)$ [11], both for crisp and fuzzy LCS [4].

4.1 Reinforcement Distribution for Interval Models

Q-Learning At time t the system is in the real-valued state s_t, after having executed action a_{t-1} from a state s_{t-1} matched by the interval-valued antecedent $\hat{s}_{t-1}$, and receives the reinforcement r_t. The Q-value given to the pair $\langle \hat{s}_{t-1}, a_{t-1} \rangle$ is updated by:

$$Q_t(\hat{s}_{t-1}, a_{t-1}) = Q_{t-1}(\hat{s}_{t-1}, a_{t-1}) + \alpha \left(r_t + \gamma \, max_a(Q_t(\hat{s}_t, a)) - Q_t(\hat{s}_{t-1}, a_{t-1}) \right)$$

with the learning rate $0 < \alpha < 1$ and the discount factor $0 < \gamma \leq 1$. The strength of the classifier c_{t-1} selected by the exploration policy at time t is $Q(\hat{s}_{t-1}, a_{t-1})$.

***TD*($\boldsymbol{\lambda}$)** At time t the reinforcement distribution algorithm performs these steps.

1. Receive the eventual reinforcement r_{t-1} due to the action a_{t-1} selected by the policy in state s_{t-1}.

2. Compute the estimation error of the evaluation function $V(\cdot)$ in s_{t-1}, equivalent to the immediate prediction error ΔE of the corresponding classifier c_{t-1} by:

$$\Delta E(c_{t-1}) = r_t + \gamma \, V_t(s_t) - V_{t-1}(s_{t-1})$$

with the discount factor $0 \leq \gamma \leq 1$.

3. For any state $\hat{s}$:

3.1. Update the *eligibility* $e(\hat{s})$ by the formula [10]:

$$e_t(\hat{s}) = \begin{cases} 1 & if \hat{s} = \hat{s}_{t-1} \\ \gamma \lambda e_{t-1}(\hat{s}) & otherwise \end{cases}$$

with $0 \leq \lambda \leq 1$.

3.2. If the eligibility of the state $\hat{s}$ is greater than a threshold value ϵ, with $0 \leq \epsilon \leq 1$, update the value function $V(\cdot)$ for any $\hat{s}$ according to:

$$V_t(\hat{s}) = V_{t-1}(\hat{s}) + \alpha \Delta E(c_{t-1}) e_t(\hat{s})$$

with $0 < \alpha < 1$.

4. Update the value of the policy for the pair $\langle \hat{s}_{t-1}, a_{t-1} \rangle$, that is the strength w of the classifier c_{t-1}, by the formula:

$$w_t(c_{t-1}) = w_{t-1}(c_{t-1}) + \beta \left(\Delta E(c_{t-1}) \right)$$

with $\beta > 0$.

4.2 Reinforcement Distribution for Fuzzy Models

We have extended the above presented algorithms to learn fuzzy models, following the considerations discussed here below.

A *real-valued state* is a set of real values for the input variables. We say that a real-valued state s matches a fuzzy state $\tilde{s}^i$ to some extent $\mu_{\tilde{s}^i}(s)$, when each real value s^k belonging to s belongs to the corresponding fuzzy set belonging to $\tilde{s}^i$, and $\mu_{\tilde{s}^i}(s)$ comes from the conjunction of the values $\mu_{\tilde{s}^i}(s^k)$, computed by the selected conjunction operator. Such operators belong to the class known as *T-norms* [8], among which the most common are *min* and *product*.

A crisp state usually matches more than one fuzzy state, while in standard LCS, it matches the set of intervals that define only one *interval state*. Let us consider subpopulations of classifiers for each fuzzy (or interval) state. In crisp LCS, classifiers belonging to the only subpopulation that matches the current state put actions in the action set, whereas in fuzzy LCS, we have many subpopulations matching the same crisp state, each proposing an action with some strength, proportional to the degree of matching of the classifier selected within the subpopulation. The strength values of the proposed actions are composed by an *aggregation* operator, usually implemented by a *T-conorm* (such as *max*), and then defuzzyfied by one of the many methods available [8]. Whatever *T-norm*, *T-conorm*, and defuzzyfication method is selected, the resulting action proposed by the set of selected fuzzy classifiers $\bar{a}$ is a function of the actions a^i proposed by the matching classifiers and of their degrees of matching $\mu_{\tilde{s}^i}$.

Since all the matching classifiers contribute to the performance of a Fuzzy LCS, the reinforcement should be distributed to all of them, proportionally to their contribution. Therefore, the first step to extend the reinforcement distribution algorithms consists of introducing a factor (let us call it ξ_{c^i}) that weights each classifier contribution. For each step, this factor is equal to the current contribution of the classifier (its degree of matching), weighted by the sum of the contribution given till now, saturated to a given value T (in these experiments $T = 2$). This enhances the learning rate of classifiers that do not yet participated much to the performance, yet.

$$\xi_{c^i} = \frac{\mu_{\tilde{s}^i}(s_t)}{min\left(T, \sum_{k=1,t}\ \mu_{\tilde{s}^i}(s_k)\right)}$$

The other important point concerns the evaluation of the best value in a given state, used in the $Q-Learning$ and $TD(\lambda)$ formulas. In this case, we cannot just take the value corresponding to the best value in the given state, since this state matches many fuzzy states, each contributing with different actions. We present our proposal for Q-Learning, leaving to the reader the analogous passages for $TD(\lambda)$. Let us consider $Q^*(c_t^{*i})$, the highest values of the classifiers c_t^{ji} belonging to each subpopulation corresponding to the matching fuzzy states i at time t.

$$Q^*(c_t^{*i}) = max_j(Q(c_t^{ji}))$$

Let us call $m_t(s)$ the sum of the $Q^*(c_t^{*i})$, each weighted by $\mu^*_{\tilde{s}^i}(s)$, the degree of matching of the corresponding classifier to the current, real-valued state s. This means that we consider as the reference best value in the current state, the combination of the best classifiers that can trigger in this state, one for each matching subpopulation. The corresponding formula is:

$$m_t(s) = \sum_{k=1,K} \mu^*_{\tilde{s}^k}(s)\, Q^*(c_t^{*k})$$

Q-Learning At time t the system is in the state s_t, after having executed action a_{t-1} from a state s_{t-1} matched by the fuzzy antecedent $\tilde{s}_{t-1}$, and receives the reinforcement r_t. The Q-value given to the classifiers c^i_{t-1} selected by the policy for each pair $\langle \tilde{s}^i_{t-1}, a^i_{t-1} \rangle$ is updated by the formula:

$$Q_t(c^i_{t-1}) = Q_{t-1}(c^i_{t-1}) + \alpha\, \xi_{c^i_{t-1}} \left(r_t + \gamma\, m_{t-1}(s_{t-1}) - Q_t(c^i_{t-1})\right)$$

***TD*(λ)** The extension of the $TD(\lambda)$ algorithm presented for crisp LCS is straightforward, given the considerations done at the beginning of this section. We do not present again all the steps, but only the key formulas.

The eligibility trace of the fuzzy state $\tilde{s}$ is updated by the formula:

$$e_t(\tilde{s}^i) = \begin{cases} \mu_{\tilde{s}^i}(s) & if\ \tilde{s}^i_t = \tilde{s}^i_{t-1} \\ \gamma\lambda\mu_{\tilde{s}^i}(s) e_{t-1}(\tilde{s}^i) & otherwise \end{cases}$$

The strength of the classifier c_{t-1}, is updated by the formula:

$$w_t(c^i_{t-1}) = w_{t-1}(c^i_{t-1}) + \beta\, \xi_{c^i_{t-1}} (r_t + \gamma \sum_{\forall i} \mu_{\tilde{s}^i}(s_t) V(\tilde{s}^i) - V_{t-1}(\tilde{s}_{t-1}))$$

5 Experimental Results

In this section, we first present the application, then the experimental settings, and, finally we discuss some of the results we have obtained.

5.1 The Application

The application we have selected is navigation of a mobile robot in a corridor. The learning activity is done on a simulated model of the actual mobile robot CAT [1], which has a car-like kinematics, is 700 mm long, 600 mm wide, and can run at a maximum speed of 300 mm/sec both forward and backward. The maximum steering degree is 30° on each side, and this is an important limitation to the maneuvering capability. In the configuration we adopted for these experiments, CAT had 8 bumpers (on/off contact sensors) all around its body, and 6

sonar sensors, covering the frontal 210°. Each sonar produces an ultrasonic wave and the time between emission and detection of a reflected wave (*Time of Flight - ToF*) is measured. This is proportional to the distance from the closer surface orthogonal to one of the rays of a 60° cone originating from the sonar. The range is between 200 and 3,230 mm. The distance obtained by each sonar is affected by noise, uniformly distributed in the interval ±5% of the maximum range. This model is rough, but realistic, and corresponds to a pair of two Polaroid sensors available on the market, each having a detection cone of about 30°.

Data from sonars are interpreted either in terms of fuzzy sets, or in terms of crisp intervals, as described in figure 1. Notice the special singleton value used to represent a characteristic feature of sonars: when no echo returns within the maximum ToF, the sensor gives a specific *out of range* value. This happens either when no object is within the range, or when all the objects within the range deflect the ultrasonic wave away from the sensor, due to the relative direction of their surfaces. A state is thus represented by six variables, each corresponding to a distance measured by one sonar sensor, and one Boolean variable, which becomes true when any of the bumpers is activated. The only output variable is steering, represented by singletons, as in figure 3. In the experiments we present here, the robot goes at a constant speed of 150 mm/sec. Whenever the robot get stuck on the wall, we bring it three steps back, so that there is a probability that it comes in a different sensorial cluster and may try different actions.

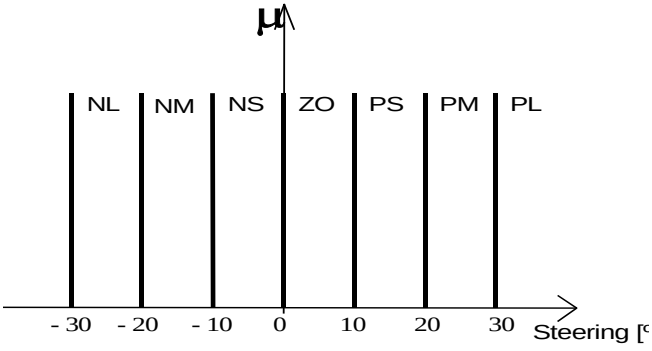


Fig. 3. The output values of steering.

5.2 Experimental Settings

Since the aim of these experiments is to discuss the impact of cluster aliasing on the learning performance, we keep fixed all the learning parameters and procedures not directly related to this aspect. The exploration strategy is: take the best matching classifier for each subpopulation with a probability equal to $(N-1)/N$, and a classifier randomly selected with a probability equal to $1/N$,

where N is a design parameter to tune exploration. In the experiments here presented, we excluded generalization, and, with it, the need for crossover: the classifiers are generated only by cover detection. The number of classifiers in a subpopulation matching a state is inversely proportional to the top performance of the subpopulation: the worst classifiers matching the current state are deleted when the performance of the subpopulation increases. Mutation is limited to the consequent part. The aim of the learning system is to find the best classifier for each state.

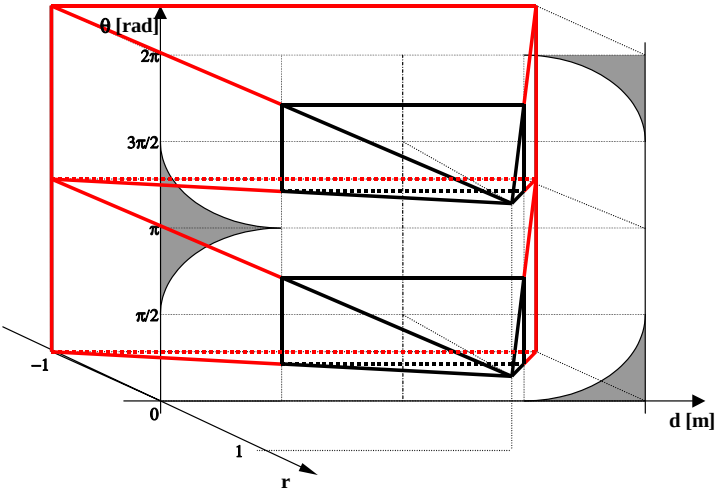


Fig. 4. The reinforcement function for the experiments reported in this paper, drawn on the c-space, along a third dimension coming towards the reader.

The reinforcement function r is the same in all the experiments, and it is designed to evolve a controller that keeps the agent moving as close as possible to the center of a corridor. In figure 4 we represent it on the c-space. You can see that it is maximum when the animat is to the center of the corridor, parallel to the walls, in any of the two directions, and minimum when the animat is bumping on the walls. Notice that it does not take into account the "no return" zones (in gray in the c-space). In this paper, we are not concerned with the optimization of the reinforcement function, and we only show the results obtained with this reinforcement function, that we consider to be *continuous*, since it can produce a reinforcement for any point in the c-space, and *complete*, since it completely describes the task, giving a reinforcement for any of the values of the variables in the c-space. A more detailed discussion about the design of the reinforcement function is presented in a forthcoming paper [5].

Each learning trial lasts 500 control steps, and sets of trials to be compared with each other have the same randomization seed. Each experiment consists of

trials sequentially starting from the 11 different positions shown in figure 5, in the order mentioned there; each sequence is repeated 10 times for a total of 110 trials and 55,000 control steps. From the set of classifiers we have at the end of the learning process, we consider only the best classifier for each subpopulation that has been tested enough, and we evaluate the performance of the controller they implement in 27 trials, lasting 700 control steps each, starting from the 11 positions shown in figure 5, and from 16 intermediate ones, in order to test the generality of the learned controller.

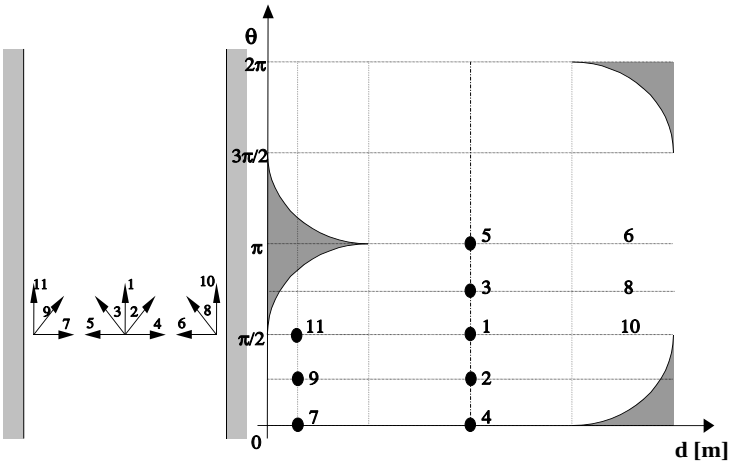


Fig. 5. The starting position for the learning trials, in the environment (left) and on the c-space (right).

5.3 Results

In this section, we discuss results from experiments each done as mentioned in section 5.2, since we believe these are more interesting than results averaged over a set of similar experiments. Moreover, we have run experiments for more than 8 millions control steps and the difference among analogous ones is non significant. In figure 6 we show the istantaneous performance (computed as the instantaneous reinforcement R), over the above mentioned 27 trials, of a learned controller whose control step lasts 600 ms. As you can see, the behavior is almost satisfactory for both the crisp and the fuzzy LCS, although the latter shows a better behavior, since its performance averaged over the 27 trials (covering all the reasonable starting positions) is higher, and it does not bring the animat on the walls, as it happens in one case (position 8) with the crisp LCS. This confirms what we have written in section 3: the fuzzy LCS is more robust with respect to cluster aliasing.

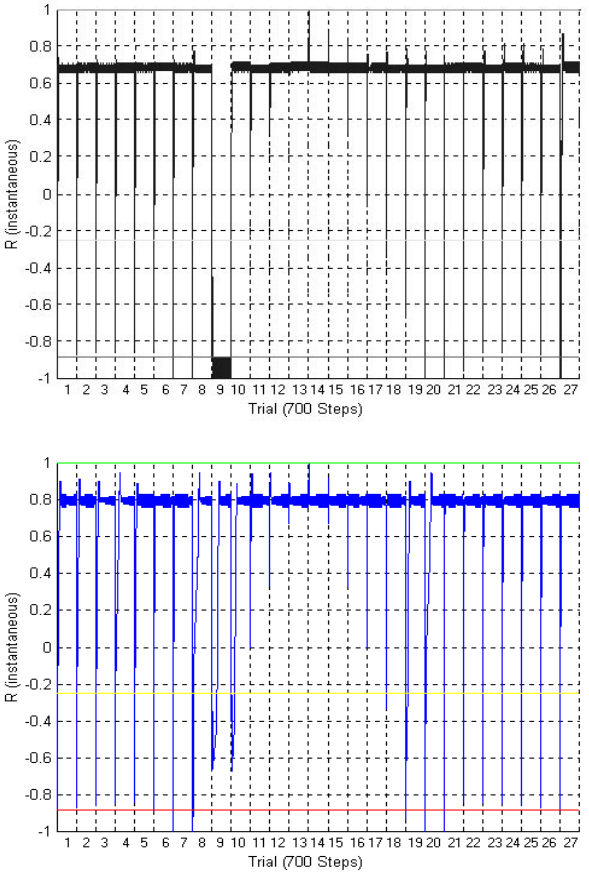


Fig. 6. Performance of a controller learnt by crisp (above) and fuzzy (below) Q-learning, control step 600ms.

In figure 7 we show the results of the same experiment, but with a control step of 100 ms. This is a way to increase cluster aliasing, since we select, and evaluate, actions at a higher rate: the probability that, at the next step, the animat is still in the same state is higher than in the previous case. In this situation, the next classifier will be selected from the same subpopulation. As we can see, the performance of both LCS decreases, although that of the crisp LCS is still slightly worse than that of the fuzzy LCS.

The same qualitative comments apply to the results obtained by adopting $TD(\lambda)$ instead of Q-learning. In this application $TD(\lambda)$ has a slightly worst performance with respect to Q-learning. This is due to our continuous reinforcement function, that slows down learning with $TD(\lambda)$, since this algorithm suffers more than Q-learning from the cluster aliasing problem. A continuous reinforcement

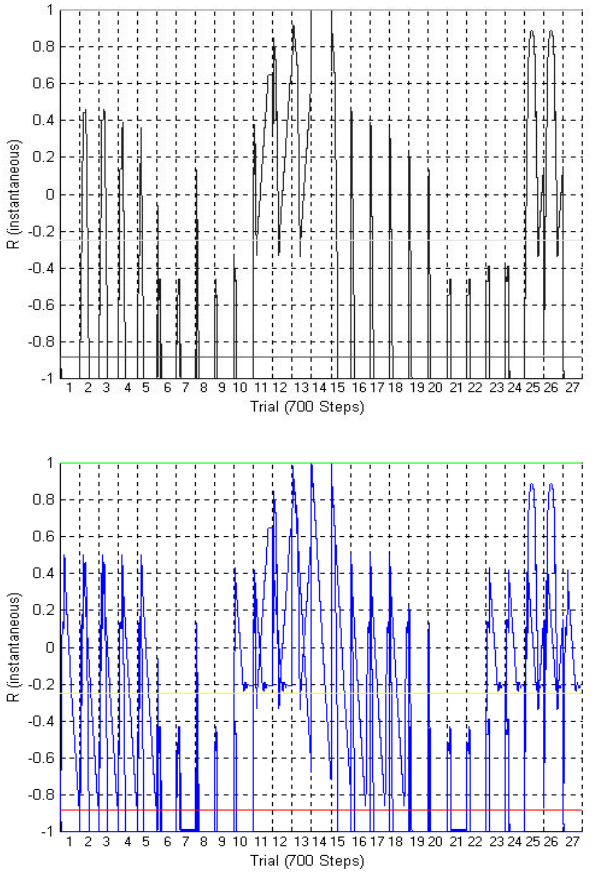


Fig. 7. Performance of a controller learnt by crisp (above) and fuzzy (below) Q-learning, control step 100ms.

function increases the confusion in the evaluation of the performance inside a cluster, since the reinforcement is similar for all the actions taken from close real-valued states, and quite different from actions taken from far real-valued states inside the same cluster.

In figure 8, we show the plots of the average reinforcement obtained during learning by Q-learning with control steps of 600ms and 100ms with the crisp and fuzzy models. In both cases, we may notice that learning reaches a stable point, but we see that they are different, due to the higher cluster aliasing present in the second situation.

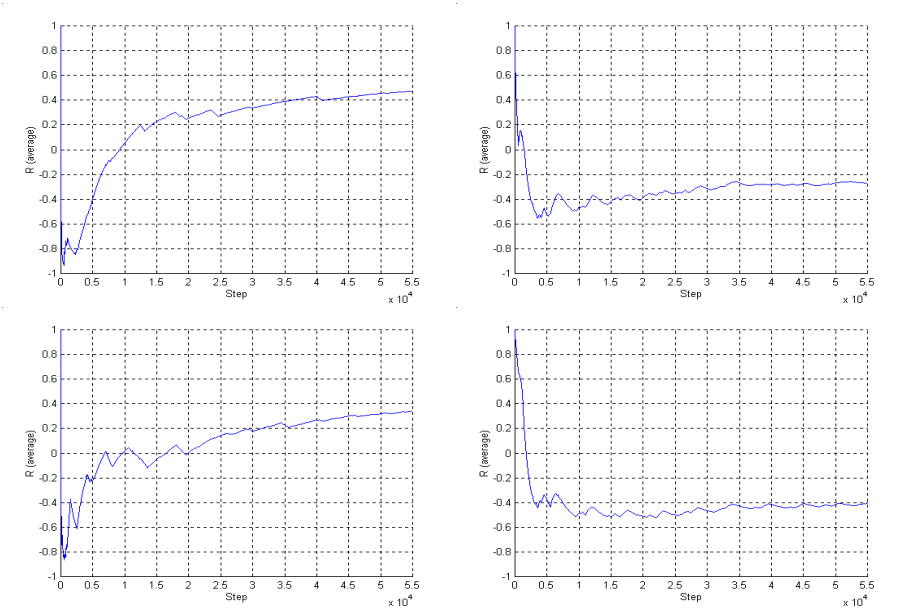


Fig. 8. Average reinforcement during learning: crisp (above) and fuzzy (below) Q-learning, control step 600ms (left), 100ms (right).

6 Conclusions

We have presented some of the problems arising when applying learning classifier systems to real-valued environments. We have focused on the impact of an interval-based representation on learning, discussing the problem of cluster aliasing that arises when more than one action is possible from one state and action execution does not imply state change.

We have introduced a fuzzy representation to get closer to a real-valued representation, while maintaining a small, symbolic, search space. Finally, we have discussed the results we have obtained to learn a relatively simple robotic task: fuzzy LCS seem more robust than interval-based LCS with respect to the problem of cluster aliasing, and, at least in the cases presented in this paper, it seems also more effective.

In future papers we will present the impact of other factors on the learning performance, such as the exploration strategy, generalization, discrete, incomplete, or accuracy-based reinforcement functions. The only consideration we would like to anticipate here is that fuzzy LCS require, in general, a higher computational effort, and a higher level of exploration; moreover they work better with crisp information, such as discrete reinforcement functions.

Acknowledgments. This project is partially supported by the Politecnico di Milano Research Grant "Development of autonomous agents through machine learning", and

partially by the project "CERTAMEN" co-funded by the Italian Ministry of University and Scientific and Technological Research. We are indebted with Nicolino De Marco, who implemented the first version of the tool supporting this research.

References

1. A. Bonarini. Evolutionary learning of fuzzy rules: competition and cooperation. In W. Pedrycz, editor, *Fuzzy modeling: paradigms and practice*, pages 265–284, Norwell, MA, 1996. Kluwer Academic Press.
2. A. Bonarini. Anytime learning and adaptation of hierarchical fuzzy logic behaviors. *Adaptive Behavior Journal*, 5(3–4):281–315, 1997.
3. A. Bonarini. Reinforcement distribution to fuzzy classifiers: a methodology to extend crisp algorithms. In *IEEE International Conference on Evolutionary Computation – WCCI-ICEC'98*, volume 1, pages 51–56, Piscataway, NJ, 1998. IEEE Computer Press.
4. A. Bonarini. Comparing reinforcement learning algorithms applied to crisp and fuzzy learning classifier systems. In *IWLCS99*, Cambridge, MA, 1999. AAAI Press.
5. A. Bonarini, C. Bonacina, and M. Matteucci. A framework to support the reinforcement function design. In preparation, 2000.
6. Andrea Bonarini. ELF: Learning incomplete fuzzy rule sets for an autonomous robot. In Hans-Jürgen Zimmermann, editor, *First European Congress on Fuzzy and Intelligent Technologies – EUFIT'93*, volume 1, pages 69–75, Aachen, D, 1993. Verlag der Augustinus Buchhandlung.
7. M. Dorigo and M. Colombetti. *Robot shaping: an experiment in behavior engineering.* MIT Press / Bradford Books, 1997.
8. G. J. Klir, B. Yuan, and U. St. Clair. *Fuzzy set theory: foundations and applicatons.* Prentice-Hall, Englewood Cliffs, MA, 1997.
9. Lozano-Perez. Spatial planning: A configuration space approach. *IEEE Transaction on Computers*, C-32(2):26–38, feb 1983.
10. S. P. Singh and R. S. Sutton. reinforcement learning with replacing eligibility traces. *Machine Learning*, 22(1):123–158, 1996.
11. R. S. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3(1):9–44, 1988.
12. C. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.
13. S. D. Whitehead and D. H. Ballard. Learning to perceive and act by trial and error. *Machine Learning*, 7:45–83, 1991.
14. S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.
15. L. A. Zadeh. Fuzzy sets. *Information and control*, 8:338–353, 1966.

Do We Really Need to Estimate Rule Utilities in Classifier Systems?

Lashon B. Booker

The MITRE Corporation
1820 Dolley Madison Blvd
McLean, VA 22102-3481, USA
booker@mitre.org

Abstract. Classifier systems have traditionally used explicit measures of utility (strength, predicted payoff, accuracy, etc.) to quantify the performance and fitness of classifier rules. Much of the effort in designing and implementing classifier systems has focused on getting these utilities "right". One alternative worth exploring is the idea of using endogenous fitness; that is, reinforcing successful performance with "resources" that rules need in order to reproduce. Under this regime, the best rules are those that accumulate the most resources over their lifetime and, consequently, have the most offspring. This paper describes a classifier system designed along these lines. Rules have no associated utility measure. Instead, each rule has one or more reservoirs that can be used to store resources. When enough resources have been accumulated, a rule utilizes some of its resources to reproduce and the reservoir level is reduced accordingly. Preliminary tests of this system on the multiplexor problem show that it performs as well as utility-based classifier systems such as XCS.

1 Introduction

One of the most attractive aspects of the classifier system framework is the way it treats rules as tentative hypotheses subject to testing and confirmation via experience with the environment [4]. A key requirement underlying this capability is a computational approach to assessing the strength of each hypothesis. In classifier systems, the strength of each hypothesis is determined from experienced-based predictions of rule performance. Hypothesis strength is used to provide a sound basis for assessing the fitness of rules, making rules a source of building blocks for generating plausible new hypotheses. Classifier system implementations have traditionally put this idea into practice by calculating parameters — such as predicted payoff, accuracy, payoff-derived strength, etc. — that explicitly estimate the utility and fitness of rules. We are investigating a classifier system that determines the utility and fitness of rules endogenously, without computing explicit estimates.

This research was prompted in part by difficulties encountered over the years trying to devise quantitative performance measures and fitness functions for

P.L. Lanzi, W. Stolzmann, and S.W. Wilson (Eds.): LCS '99, LNAI 1813, pp. 125–141, 2000.

traditional classifier systems. In a complex and uncertain environment, a single measure of what makes a hypothesis useful or reliable can be hard to come by. For some problem domains, it is sufficient if a system finds hypotheses that minimize the total number of classification errors. Sometimes, though, it is important to scrutinize hypotheses more carefully, insisting on properties such as a low rate of false positive errors. In other cases, the challenge could be to find clusters that cover the input space subject to various constraints. Designing a single performance measure that accommodates such a wide variety of performance requirements, including their context dependent interactions, can be a significant engineering challenge.

Holland recognized that many of the adaptive interactions that arise in classifier systems can be studied more directly in a simpler framework. The Echo class of models [5] provides such a setting. Echo offers a framework in which simply-defined agents interact in carefully structured ways that are more concrete than the general-purpose, pattern-directed procedures used in classifier systems. Another interesting difference between Echo and classifier systems is that there are no explicit fitness functions for the agents in Echo. The fitness of agents in Echo is determined endogenously. Agents interact with each other and with the environment to acquire the resources needed to survive and reproduce. While fitness is a function of the agent and the overall context the agent finds itself in, that function never has to be explicitly defined as a quantitative measure of adaptive proficiency.

We hypothesize that endogenous fitness can provide a similar advantage in the classifier system framework. This paper introduces ideas about how to implement endogenous fitness in classifier systems, and gives preliminary results on how well those ideas work. We begin with a brief discussion of why we expect endogenous fitness to be useful in classifier systems. That discussion is followed by a description of our initial implementation, and results from experiments with the Boolean multiplexor problem.

2 Endogenous Fitness in Classifier Systems

Our focus in this paper is on using endogenous fitness to address the issue of generalization accuracy. As Wilson [8] has pointed out, traditional classifier systems have no way of distinguishing between an accurate classifier having some modest payoff value, and an inaccurate classifier whose average payoff is equal to that value. Moreover, traditional classifier systems have few explicit mechanisms in place to promote the discovery and proliferation of accurate rules. As a result, overly general rules continually threaten to take over the population, since they match more messages than their less frequently activated, but potentially more accurate, counterparts.

Though measuring the accuracy of a rule can itself be a tricky endeavor [6], there are ways to define rule utilities so that overly general rules are less fit and less likely to dominate the population. For example, one of Holland's earliest discussions of classifier systems emphasized the importance of using prediction

errors to identify rules that overestimate their expected payoff [3]. In addition, Booker [2] defined a consistency parameter, based on the mean-squared prediction error, that was used to attenuate rule fitness and provide an advantage for accurate rules. More recently, Wilson [8] has devised the XCS system that uses a fitness criterion based solely on minimizing prediction error. The success of XCS demonstrates how important it is to effectively address the accuracy issue in any successful classifier system implementation.

The rationale for using endogenous fitness to address the problem of generalization accuracy is based on the following intuition: If rules need resources to reproduce, and accurate rules accumulate resources and reproduce at a faster rate than inaccurate rules, then accurate rules will tend to take over the population. The key is to find simple ways to adjust the relative rate at which accurate and inaccurate rules accumulate the resources needed to reproduce. Our approach therefore focuses on how to design such a competition for resources between accurate and inaccurate rules.

To begin with a simple case, assume that we are dealing with a stimulus-response problem in which the rules advocating an action on a given time step (i.e., the action set) take all of the credit for whatever reinforcement ("good" or "bad") is received. Assume further that each reinforcement event leads to the distribution of some nominal resources among those rules, and that the acquired resources accumulate over time in internal reservoirs. Following the way resources are used in Echo [5], we specify that a rule needs to accumulate resources in excess of some threshold level before it is allowed to reproduce. Note that the expected net accumulation of resources, reflecting the net balance of "good" and "bad" reinforcement received, does not depend on how frequently a rule is active. It depends most directly on the probability of reinforcement and, therefore, on the probability that the rule is "correct". Consequently, there is some hope that we can sidestep the issue faced by systems employing explicit fitness functions which must try to compensate for the fact that general rules typically have more opportunities to reproduce than specialized rules.

While this intuitive picture of how to proceed is fairly straightforward, there are many pragmatic details that need to be sorted out. The most important details are those governing the rate of accumulation of resources and the frequency of reproduction. Mathematically speaking, this approach tries to exploit the expected fluctuations in reservoir levels given a sequence of reinforcement events correlated with a rule's responses. The sequence of changes in reservoir levels can be viewed as a generalized random walk. Given that there are many analytical results available concerning the behavior of random walks and other fluctuation phenomena, it is reasonable to expect that some version of this endogenous fitness scheme should be amenable to mathematical analysis[1]. As a prelude to a formal analysis, we have conducted a series of empirical experiments to help determine an appropriate way to pose the problem in the classifier system setting. The remainder of this paper describes those experiments and the classifier system implementations that appear to be on the right track.

[1] This is not meant to suggest that such an analysis will be easy.

3 System Description

The classifier system used in this study has a fairly conventional design, borrowing elements from previous work on GOFER [1,2] and XCS [8]. The population of classifiers has a fixed size $\mathcal{N}$. Each classifier in the population has a single condition on the left side and a single action on the right side. In a departure from previous systems, there are no performance estimates associated with classifiers. Instead, each classifier ξ has two associated reservoirs: the $\Delta_+(\xi)$ reservoir that stores resources obtained from "good" reinforcement events, and the $\Delta_-(\xi)$ reservoir that stores resources obtained from "bad" reinforcement events. In all of our experiments the reservoirs were initialized to be empty and the initial classifiers were generated at random. The only other parameter stored with each classifier is its age $\alpha(\xi)$, which is used in the procedure for deleting classifiers.

3.1 Performance System

The system performance cycle is fairly routine. For each input message i, the system first determines the set of classifiers $\mathbf{M}$ eligible to classify the message. Matching classifiers are always included in $\mathbf{M}$. Following the procedures in GOFER, if there are fewer than $\mathcal{N}_m$ matching classifiers available, classifiers with the highest partial match scores are deterministically selected to fill out $\mathbf{M}$. We use the simple partial match score

$$\mu(\xi, i) = \begin{cases} s + l \text{ if } \xi \text{ matches the message } i \\ l - n \text{ otherwise} \end{cases}$$

where l is the length of the input condition in ξ, s is the specificity[2] , and n is the number of positions where the condition doesn't match the message.

For each action a represented in $\mathbf{M}$, the system computes an *action mandate* that is similar in intent to the system prediction computed in XCS. The action mandate is supposed to capture the system's knowledge about the likelihood of a "good" outcome if action a is chosen. There are several ways this could be computed. We currently use the relative amount of resources in the $\Delta_+(\xi)$ and $\Delta_-(\xi)$ reservoirs as an indicator of expected outcome. Each classifier ξ in $\mathbf{M}$ contributes

$$\lambda(\xi) = \frac{|\Delta_+(\xi)|}{(|\Delta_+(\xi)| + |\Delta_-(\xi)|)}$$

to the mandate for its action. The rationale for this particular approach is to give more weight to those actions that, based on previous experience, have the highest likelihood of being followed by a "good" reinforcement event. This contribution from each rule is added to an *action selection array* and, as in XCS, an action is selected using one of many possible selection methods. The members of $\mathbf{M}$ that agree with the selected action constitute the *action set* $\mathbf{A}$. The system then sends that action to the effectors, and the environment may respond with reinforcement.

[2] The number of non-#'s in the input condition

3.2 Reinforcement

For convenience, we assume for now that a zero reward indicates no reinforcement event has occurred. When a non-zero reward is received, a fixed amount of resource $\mathbf{R} \geq 0$ is made available to the classifiers in the action set. As in the GOFER system, each action set classifier is eligible to receive a share of that resource. We have experimented with several ways of allocating the resource among the classifiers in $\mathbf{A}$. The most effective method so far divides the resource based on the largest likelihood that each classifier assigns to one of the outcomes. This likelihood is estimated by

$$\lambda_*(\xi) = \frac{|\Delta_*(\xi)|}{(|\Delta_+(\xi)| + |\Delta_-(\xi)|)}$$

where $\Delta_*(\xi)$ is the reservoir containing the largest amount of resources. Each classifier in $\mathbf{A}$ receives a share of the resource given by

$$\rho(\xi) = \left(\frac{\lambda_*(\xi)}{\Lambda}\right)\mathbf{R}$$

where Λ is the total likelihood in $\mathbf{A}$. When the reinforcement event is "good", $\rho(\xi)$ is added to $\Delta_+(\xi)$; otherwise, it is added to $\Delta_-(\xi)$. The idea is to bias the allocation of resources in favor of those classifiers that provide the most decisive hypothesis about the outcome associated with the selected action.

Under this regime, rules that are consistently associated with only one type of outcome will quickly achieve a large net accumulation of resources in the $\Delta_*(\xi)$ reservoir since all of their resources are stored in one place. Conversely, rules associated with both outcomes will distribute their resources over both reservoirs, taking longer to attain a large net accumulation in $\Delta_*(\xi)$. This is significant because the frequency of reproduction is tied to the net accumulation of resources in the $\Delta_*(\xi)$ reservoir.

3.3 Rule Discovery

After the rule reservoirs have been updated, any classifier in $\mathbf{M}$ having a sufficient excess of resources in the $\Delta_*(\xi)$ reservoir becomes eligible to reproduce. An excess of resources is indicated by

$$|\Delta_+(\xi) - \Delta_-(\xi)| > \tau$$

for some threshold τ. When reinforcement is correlated with "correct" and "incorrect" actions, the frequency of satisfying this reproduction criterion is correlated with the frequency that a rule's response is correct. Rules that are consistently correct (or consistently incorrect) will consequently enjoy a reproductive advantage over rules that are inconsistent. Note that this notion of directly assessing the performance of the decision policy leads to a policy-level view of what it means for a rule to be accurate. In GOFER, XCS and other systems that

try to estimate the expected reward associated with a state or state-action pair, accuracy refers to the precision of those value estimates.

If there is more than one classifier in **M** eligible to reproduce on a given cycle, all eligible classifiers are designated as parents and allowed to produce one copy of themselves. Parents then have their $\Delta_*(\xi)$ reservoir decremented by τ, which can be viewed as the cost of generating an offspring. The reproduced copies are modified by mutation and crossover, and the resulting offspring are inserted into the population. Classifiers are stochastically selected for deletion based on their age, so that older classifiers are more likely to be chosen. Note that this scheme triggers rule discovery without using any extra bookkeeping or monitoring of the state of the system.

4 Empirical Results

As a simple initial test of how well this system works, we apply it to the familiar Boolean multiplexor problem. The value of the multiplexor function is computed by using the first n bits of an l-bit string as an index. That index selects one of the remaining 2^n bits in the string (where $n + 2^n = l$), and the selected bit is returned as the function value. We initially restrict our attention to the case $n = 2$ (the 6-bit multiplexor). For this problem, the following set of rules provides a complete solution:

```
000### ==> 0   10##0# ==> 0
001### ==> 1   10##1# ==> 1
01#0## ==> 0   11###0 ==> 0
01#1## ==> 1   11###1 ==> 1
```

Each of these rules is maximally general in the sense that no more `#`'s can be added to their input conditions without introducing performance errors.

In all experiments, unless otherwise noted, the system constants were as follows: $\mathcal{N} = 400$, $\mathcal{N}_m = 16$, $\mathbf{R} = 500$, $\tau = 1000$, initial reservoir levels of 0 for new offspring, a mutation rate of 0.01, and a crossover rate of 1.0. We identify a correct system response as a "good" event and an incorrect response as a "bad" event. As a convenient simplification, we initially use a positive reward (+1000) to signal a "good" event and a negative reward (−1000) to signal a "bad" event. The initial version of the endogenous fitness algorithm only checks the sign of the reward, though, so the magnitude of these rewards is ignored. In subsequent sections we will describe a generalization of this basic scheme in which rewards with different magnitudes can be used and the sign of the reward does not automatically determine if the response is "good" or "bad".

Performance was measured by the proportion of correct decisions over a learning epoch consisting of 50 randomly generated input strings. Each experiment was run for 200 epochs (10,000 input strings). In order to facilitate comparisons with XCS on this problem, we used Wilson's [8] action-selection regime. This regime makes a random (probability 0.5) choice between "exploit" mode — where the system selects the best action as indicated by the action mandate

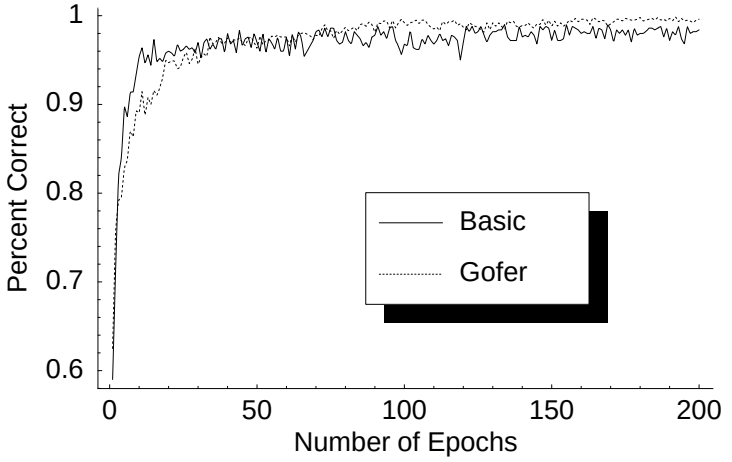


Fig. 1. Initial performance on 6-bit multiplexor

array — and "explore" mode where the system selects an action at random[3]. No rule discovery operations occur in exploit mode. All results are averaged over 10 independent runs.

4.1 Initial Test Of The Basic System

The results of applying the system described above to the 6-bit multiplexor problem are shown in Figure 1.

This initial implementation of the endogenous fitness algorithm has a performance trajectory that is very similar to the performance curve of the more conventional GOFER approach. At the end of 200 epochs, performance was approximately 98.2% correct as compared to 99.6% for GOFER. Direct comparison with the GOFER performance curve is somewhat misleading, since GOFER used a roulette-wheel action selection scheme that allowed learning on every trial. Moreover, GOFER only tried to learn the 8 maximally general classifiers that are always correct. Here, in order to assess generalization accuracy, we make our system learn all 16 maximally general classifiers, both those that are 100% correct and those that are 100% incorrect. Nevertheless, the performance of the endogenous fitness approach appears to be reasonably good on this task.

When we examine the rules that the endogenous fitness algorithm has learned after 200 epochs, however, we see that there is considerable room for improvement. Figure 2 shows the most prevalent rules in a population of classifiers at the end of a typical run. The numbers in parentheses after each rule give the number of rule instances and the reward expected when that rule controls the system

[3] During explore trials we record a correct decision whenever the system would have made the correct "exploit" decision

```
(State = ##1#1#) ===> 0 (8,-407.1)
(State = ##11##) ===> 0 (9,-89.9)
(State = #0##1#) ===> 0 (9,44.6)
(State = #1#1#1) ===> 1 (11,1000.0)
(State = 000###) ===> 1 (26,-1000.0)
(State = 01#1##) ===> 0 (14,-1000.0)
(State = 01#1##) ===> 1 (14,1000.0)
(State = 1###0#) ===> 1 (9,-95.4)
```

Fig. 2. Typical set of rules learned by the initial endogenous fitness scheme

response[4]. The system typically learns all of the maximally general classifiers, but only a few manage to attain a sizable foothold in the population during the first 200 epochs. In retrospect this is not too surprising. Over-general rules that help the system perform better than random in early epochs do not suffer an overwhelming disadvantage in the competition for resources with maximally general classifiers. The competition for resources is governed by the likelihoods $\lambda_*(\xi)$. An over-general classifier like

```
1###0# ===> 1
```

is correct on only 4 of the 16 messages it matches, and so we might naively[5] expect it to have an associated likelihood of $\lambda_*(\xi) = 0.75$. While this is smaller than the corresponding expectation of $\lambda_*(\xi) = 1.0$ for a maximally general classifier, it takes time for this difference to lead to the demise of the over-general classifier, especially when the over-general classifier has a significant number of instances in the population. Eventually the difference in likelihoods does take its toll. When we extend the experiments to 400 epochs we see that by 350 epochs the maximally general classifiers have taken over the population and the performance curves are steady at 100% correct. However, this is about twice as long as it takes systems like GOFER and XCS to generate a comparable population of maximally general rules.

4.2 Generating A Better Covering

One of the reasons that the basic system fails to quickly find and exploit all of the maximal generalizations is that the selection pressure against overly general or overly specific classifiers is weaker here than in GOFER and XCS. The most efficient population of classifiers for solving the multiplexor problem is one that consists solely of the 16 maximally general classifiers. These classifiers *cover*

[4] The statistics about expected reward were computed for diagnostic purposes only. They were not used by the performance system.

[5] Even if the algorithm were formulated as a simple, symmetric random walk, chance fluctuations make it unlikely that we would observe such "intuitively obvious" reservoir levels for any given classifier.

the set of input strings in the sense that they correctly indicate the outcome associated with each action and each input. The maximally general classifiers provide an *efficient* covering in the sense that they are the most compact set of rules that covers all of the input strings. The endogenous fitness scheme, in its current form, is not explicitly designed to learn an efficient covering. We can improve the performance of the algorithm by finding a way to provide more direct guidance about how to generate such a covering[6].

Smith *et al* [7] describe an intriguing approach to generating selection pressures for efficient coverings. Their technique is based on a simple model of the immune system in which antibodies compete with each other to match antigens. Both antigens and antibodies are represented as binary strings. The fitness of each antibody is determined by the number of times it is selected as the winner of a competition. Experiments show that, under this fitness regime, a genetic algorithm (GA) can generate and maintain a diverse population of antibodies that covers the set of antigens. Moreover, each antibody has a frequency in the population proportional to the frequency with which the corresponding antigens it covers are sampled. An analysis of this mechanism shows that it is strongly related to the familiar fitness-sharing schemes often used in GA implementations to solve multimodal optimization problems.

We can exploit this approach for our purposes by using the probability of winning such a competition to help bias the distribution of resources in **A**. More specifically, assume that we want to model the results of the following competition for each input message:

1. Select a random sample of size r without replacement from **M**.
2. Choose the classifier ξ in **A** with the largest value of $\lambda_*(\xi)$ in the sample as the winner on this iteration. Ties are broken at random. If there are no classifiers from **A** in the sample, then there is no winner and we repeat step 1.
3. Add $\lambda_*(\xi)$ to the total competition score for ξ.
4. Repeat the process for several iterations. The classifier that accumulates the highest overall score is the winner of the competition.

Smith *et al* show that the probability of winning this kind of competition can be expressed in terms of hypergeometric probabilities. The hypergeometric distribution $h(k, r, n, m)$, given by

$$h(k, r, n, m) = \frac{\binom{m}{k}\binom{n-m}{r-k}}{\binom{n}{r}}$$

is used to compute the probability that exactly k red balls will be in a sample of size r, drawn without replacement from a population of size n that contains m

[6] Since we rely on a genetic algorithm for rule discovery, we avoid doing anything that over-zealously eliminates diversity from the population. Diversity is the "grist for the mill" in genetic search.

red balls. In order for a classifier ξ to win the competition for a given message, no classifier φ with $\lambda_*(\varphi) > \lambda_*(\xi)$ can occur in the sample. Moreover, whenever there are classifiers φ with $\lambda_*(\varphi) = \lambda_*(\xi)$ in the sample, ξ has to be the one chosen by the tie-breaking procedure. The probability that these events all occur is given by

$$\mathbf{H}(\xi) = \frac{h(0, r, |\mathbf{M}|, G(\xi))(1 - h(0, r, |\mathbf{M}| - G(\xi), F(\xi)))}{F(\xi)}$$

where $F(\xi)$ is the number of classifiers φ in $\mathbf{A}$ with $\lambda_*(\varphi) = \lambda_*(\xi)$, and $G(\xi)$ is the number classifiers φ in $\mathbf{A}$ with $\lambda_*(\varphi) > \lambda_*(\xi)$. Given the way sample size controls the extent of resource sharing in this algorithm [7], the value $r = 0.5|\mathbf{M}|$ appears to be a reasonable choice for our classifier system implementation.

Directly computing the hypergeometric probabilities is a cost-effective alternative to explicitly simulating the multi-stage competition on each classifier system cycle. The log gamma function, available as a subroutine in standard math libraries, provides a direct way to compute factorials and binomial coefficients based on the fact that

$$\Gamma_{\ln}(n) \stackrel{\text{def}}{=} \log(\Gamma(n)) = \log((n-1)!)$$

for integers n. The hypergeometric probabilities $h(0, r, n, m)$ that we need are given by

$$\begin{aligned} h(0, r, n, m) = \exp(&\Gamma_{\ln}(n - m + 1) + \Gamma_{\ln}(n - r + 1) \\ &- \Gamma_{\ln}(n - m - r + 1) - \Gamma_{\ln}(n + 1)) \end{aligned}$$

where exp is the exponential function[7]. We use these probabilities to exert pressure for a more efficient covering by modifying the endogenous fitness scheme to use

$$\tilde{\rho}(\xi) = \left(\frac{\lambda_*(\xi)\mathbf{H}(\xi)}{\sum_{\xi \in \mathbf{A}} \lambda_*(\xi)\mathbf{H}(\xi)} \right) \mathbf{R}$$

when allocating resources to $\mathbf{A}$. This biases the allocation of resources in the desired way, making elements of a good covering more likely to reproduce.

This modified version of the endogenous fitness scheme was tested on the 6-bit multiplexor problem. The same parameters were used as before, except that we used an increased value $\mathbf{R} = 5,000$ to compensate for the change of scale caused by using the hypergeometric probabilities in the computation of $\tilde{\rho}$. Figure 3 shows that the selection pressure for a good covering does indeed improve performance.

Figure 4 shows that the performance improvements include a much better capability to generate an efficient covering. The system reliably finds all of the

[7] This computation is made more efficient by noting that we will use at most $\mathcal{N}$ different values of the factorial function. Accordingly, values can be computed once using the log gamma function and stored in a table for retrieval on subsequent function calls.

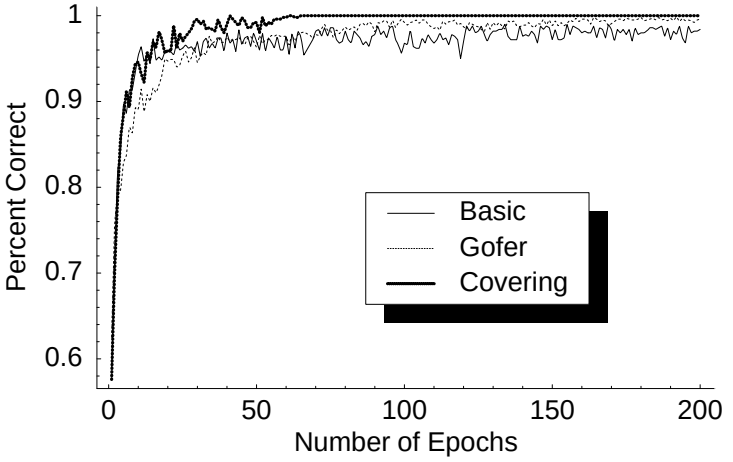


Fig. 3. Performance of the improved covering algorithm on the 6-bit multiplexor

maximally general classifiers, and they rapidly establish a strong presence in the population. Populations of this kind are comparable to what we expect from systems like XCS.

```
(State = 0#11##) ==> 1 (8,1000.0)
(State = 001###) ==> 0 (17,-1000.0)
(State = 001###) ==> 1 (23,1000.0)
(State = 000###) ==> 0 (21,1000.0)
(State = 000###) ==> 1 (16,-1000.0)
(State = 01#0##) ==> 0 (28,1000.0)
(State = 01#0##) ==> 1 (19,-1000.0)
(State = 01#1##) ==> 0 (9,-1000.0)
(State = 01#1##) ==> 1 (12,1000.0)
(State = 10##1#) ==> 0 (24,-1000.0)
(State = 10##1#) ==> 1 (16,1000.0)
(State = 10##0#) ==> 0 (25,1000.0)
(State = 10##0#) ==> 1 (19,-1000.0)
(State = 11###1) ==> 0 (28,-1000.0)
(State = 11###1) ==> 1 (17,1000.0)
(State = 11###0) ==> 0 (20,1000.0)
(State = 11###0) ==> 1 (24,-1000.0)
```

Fig. 4. Typical set of rules learned by the endogenous fitness scheme modified for improved covering

4.3 Arbitrary Reinforcement Schemes

Up to this point, we have focused on developing the endogenous fitness mechanisms in a simplified setting. The system has only been required to learn to distinguish between correct and incorrect responses given training experiences that provide explicit information about response categories. Most reinforcement learning problems, though, impose requirements that are much different. Generally speaking, a reinforcement learning problem forces a learning system to use scalar rewards without any accompanying labels or supplemental information as input. To put it simply, the system must learn to choose the action that results in the most reward in a given situation. Any approach to solving reinforcement learning problems that cannot meet this challenge is severely limited.

There is a fairly straightforward way to generalize the endogenous fitness algorithm we have described to handle such arbitrary reinforcement schemes. The key is to recognize that the system does not really need external guidance about which reservoir, $\Delta_+(\xi)$ or $\Delta_-(\xi)$,should be used to store a resource. Instead of thinking in absolute terms about "good" or "bad" outcomes, we can think in relative terms about "better" or "worse" outcomes given some reference level for rewards. The average reward is an easily computed reference level we can use for this purpose. When a reinforcement event involves above average reward, the resource can be added to the $\Delta_+(\xi)$ reservoir; otherwise, it goes to the $\Delta_-(\xi)$ reservoir.

This idea suggests that we make the following changes to our implementation of the endogenous fitness algorithm:

- Each classifier ξ gets a new parameter $\pi(\xi)$ that estimates the average reward available when ξ is included in $\mathbf{M}$. We have experimented with several approaches to computing $\pi(\xi)$. The most effective method so far computes it as the sample average of the rewards received on "explore" trials. This computation uses another new parameter $\nu(\xi)$ that counts the number of times ξ has been included in $\mathbf{M}$ on an "explore" trial. The $\pi(\xi)$ parameter is revised using the simple update rule

$$\pi_t(\xi) = \begin{cases} \frac{(\pi_{t-1}(\xi)\nu_{t-1}(\xi))+\mathcal{R}_t}{\nu_t(\xi)} & \text{if } \nu_t(\xi) \neq \nu_{t-1}(\xi) \\ \pi_{t-1}(\xi) & \text{otherwise} \end{cases}$$

  where $\mathcal{R}_t$ is the reward at time t, $\pi_0(\xi) = 0$, and $\nu_0(\xi) = 0$. In terms of standard approaches to solving reinforcement learning problems, we can think of $\pi(\xi)$ as an estimate for the utility of the states that ξ is used in. Note that it is not an estimate for the utility of ξ.
- The mean value of these estimates, weighted by likelihood, is used to determine the reference level for interpreting reinforcement events. More specifically, we use the members of $\mathbf{M}$ to collectively estimate the utility of the current state as

$$\Pi = \frac{\sum_{\xi \in \mathbf{M}} \lambda_*(\xi)\pi(\xi)}{\sum_{\xi \in \mathbf{M}} \lambda_*(\xi)}$$

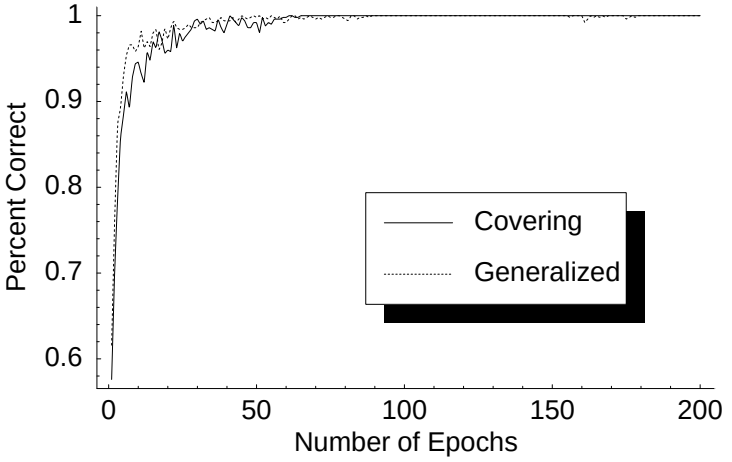


Fig. 5. Performance using the (1000,-1000) reward scheme

and determine if the current reward is above or below average by comparing it to Π. When the reward is above average, the resource is added to $\Delta_+(\xi)$. When the reward is below average, it is added to $\Delta_-(\xi)$.

– We linearly scale the resource $\mathbf{R}$ available on each time step to reflect how much the reward deviates from average, using

$$\mathbf{R}\left(0.5 + \frac{|\mathcal{R} - \Pi|}{2\delta}\right)$$

where δ is the range of the scalar rewards and represents the maximum possible deviation (δ is a system parameter that must be provided). This scaling yields $\mathbf{R}/2$ for average events and $\mathbf{R}$ for events involving a deviation of size δ. Given two classifiers that are consistently associated with above (or below) average rewards, this modification to $\mathbf{R}$ provides a modest selective advantage to the classifier that is best (or worst).

All other aspects of the endogenous fitness algorithm remain the same, including the aforementioned modifications for improved covering.

We conducted several experiments to determine if this generalized version of the endogenous fitness algorithm can learn the multiplexor problem given arbitrary reinforcement schemes that do not indicate correct or incorrect responses. Initially, we used two simple reinforcement schemes: a (1000,-1000) reward scheme giving 1000 for right answers and -1000 for wrong answers; and, a (1000,500) reward scheme. As shown in Figures 5 and 6, the performance of the generalized endogenous fitness algorithm using these reward schemes is almost identical the previous performance achieved using the covering enhancements with explicit designations of correct and incorrect actions.

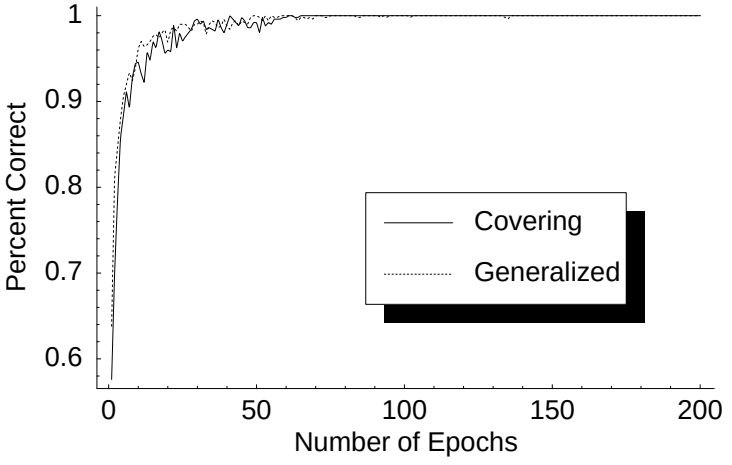


Fig. 6. Performance using the (1000,500) reward scheme

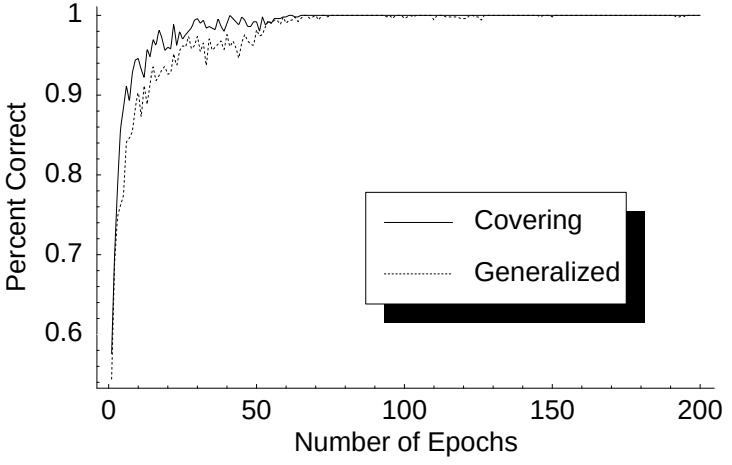


Fig. 7. Performance using Wilson's layered payoff landscape

A more challenging test is offered by Wilson's [8] layered payoff landscape for the multiplexor problem. This payoff landscape provides different rewards for each set of eight strings that matches one of the maximally general classifiers. The reward scheme for right and wrong answers associated with each set of strings is summarized below:

```
000### ==> 0 (300,0)     10##0# ==> 0 (700,400)
001### ==> 1 (400,100)   10##1# ==> 1 (800,500)
01#0## ==> 0 (500,200)   11###0 ==> 0 (900,600)
01#1## ==> 1 (600,300)   11###1 ==> 1 (1000,700)
```

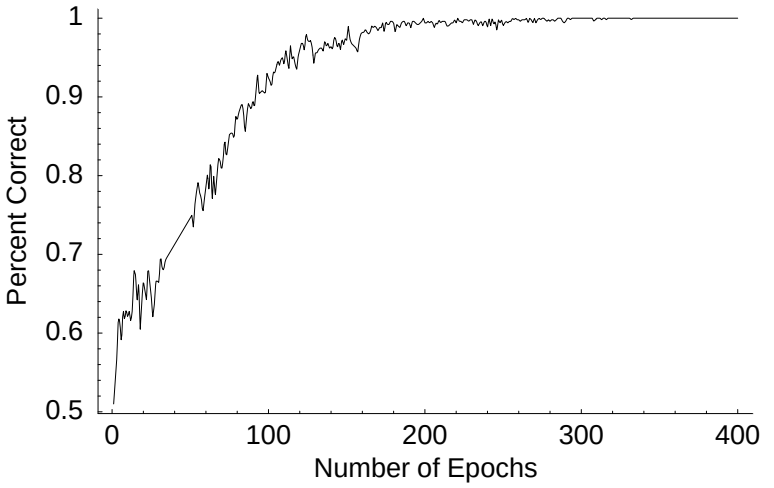


Fig. 8. Performance on the 11-bit multiplexor

Figure 7 shows that the generalized endogenous fitness algorithm does learn to solve the problem using this payoff landscape, though at a slightly slower pace than the previous version. One factor contributing to this discrepancy is the way we scaled resources. In this payoff landscape there is a different average payoff for each set of eight strings. The payoff range in each set is 300. However, since we used the overall payoff range of 1000 as δ when we scaled resources, reservoir levels were increased in increments less than $\mathbf{R}$ on each time step. This means it takes longer for reservoir levels to exceed their threshold for reproduction and, consequently, longer for the system to learn how to solve the problem. An additional complication is introduced by the presence of overgeneral rules, which can distort estimates of the state utility $\mathit{\Pi}$. These effects all need to be examined more carefully, and are an important topic for further research.

As a final test, we looked at the performance of our system on the more difficult 11-bit multiplexor problem. The only change we made to the system is to use a larger population size $\mathcal{N} = 800$. Figure 8 shows the performance using the (1000,500) reward scheme. The system solves the problem after about 12,000 input strings, which is consistent with results reported for XCS [8]. Figure 9 shows that the system generates a population full of maximally general classifiers as we would expect.

5 Summary

The results presented here are modest and preliminary. However, they strongly suggest that endogenous fitness schemes like those described in this paper are worth further investigation. Many important research issues remain to be investigated, particularly those related to multi-step reinforcement learning pro-

```
(State = 0000######) ===> 0 (28,1000.0)
(State = 0000######) ===> 1 (13,500.0)
(State = 0001######) ===> 0 (19,500.0)
(State = 0001######) ===> 1 (47,1000.0)
(State = 001#0#####) ===> 0 (44,1000.0)
(State = 001#0#####) ===> 1 (9,500.0)
(State = 001#1#####) ===> 0 (14,500.0)
(State = 001#1#####) ===> 1 (35,1000.0)
(State = 010##0####) ===> 0 (22,1000.0)
(State = 010##0####) ===> 1 (13,500.0)
(State = 010##1####) ===> 0 (13,500.0)
(State = 010##1####) ===> 1 (32,1000.0)
(State = 011###1####) ===> 0 (15,500.0)
(State = 011###1####) ===> 1 (23,1000.0)
(State = 011###0####) ===> 0 (30,1000.0)
(State = 011###0####) ===> 1 (10,500.0)
(State = 100####0###) ===> 0 (34,1000.0)
(State = 100####1###) ===> 1 (18,1000.0)
(State = 101#####0##) ===> 0 (29,1000.0)
(State = 101#####0##) ===> 1 (13,500.0)
(State = 101#####1##) ===> 1 (22,1000.0)
(State = 110######1#) ===> 0 (12,500.0)
(State = 110######1#) ===> 1 (30,1000.0)
(State = 110######0#) ===> 0 (33,1000.0)
(State = 110######0#) ===> 1 (12,500.0)
(State = 111#######1) ===> 0 (9,500.0)
(State = 111#######1) ===> 1 (35,1000.0)
(State = 111#######0) ===> 0 (43,1000.0)
(State = 111#######0) ===> 1 (13,500.0)
```

Fig. 9. Typical set of rules learned for the 11-bit multiplexor

blems involving delayed rewards. While more experiments are needed to build our intuitions about how this approach can be used, the most important thing to do is formally analyze how the fluctuations in reservoir levels are translated into reproduction rates. That kind of understanding is vital if these schemes are ever to become reliable approaches for building classifier systems. Our current research efforts are focused on conducting that analysis.

Acknowledgments. This research was funded by the MITRE Mission-Oriented Investigation and Experimentation (MOIE) research program. That support is gratefully acknowledged.

References

1. Booker, L. B. Classifier systems that learn internal world models. *Machine Learning 3* (1988), 161–192.

2. Booker, L. B. Triggered rule discovery in classifier systems. In *Proceedings of the Third International Conference on Genetic Algorithms (ICGA89)* (Fairfax, VA, 1989), J. D. Schaffer, Ed., Morgan Kaufmann, pp. 265–274.
3. Holland, J. H. Adaptation. In *Progress in theoretical biology*, R. Rosen and F. M. Snell, Eds., vol. 4. Academic Press, New York, 1976.
4. Holland, J. H. Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In *Machine learning: An artificial intelligence approach*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, Eds., vol. II. Morgan Kaufmann, Los Altos, CA, 1986, ch. 20, pp. 593–623.
5. Holland, J. H. Echoing emergence: Objectives, rough definitions, and speculations for Echo-class models. In *Complexity: Metaphors, Models, and Reality*, G. Cowan, D. Pines, and D. Melzner, Eds., vol. XIX of *Santa Fe Institute Studies in the Sciences of Complexity*. Addison-Wesley, Reading, MA, 1994, pp. 309–342.
6. Horn, J., Goldberg, D. E., and Deb, K. Implicit niching in a learning classifier system: Nature's way. *Evolutionary Computation 2*, 1 (1994), 37–66.
7. Smith, R. E., Forrest, S., and Perelson, A. S. Searching for diverse, cooperative populations with genetic algorithms. *Evolutionary Computation 1*, 2 (1993), 127–149.
8. Wilson, S. W. Classifier fitness based on accuracy. *Evolutionary Computation 3*, 2 (1995), 149–175.

Strength or Accuracy? Fitness Calculation in Learning Classifier Systems

Tim Kovacs

School of Computer Science University of Birmingham
Birmingham B15 2TT United Kingdom
T.Kovacs@cs.bham.ac.uk

Abstract. Wilson's XCS is a clear departure from earlier classifier systems in terms of the way it calculates the fitness of classifiers for use in the genetic algorithm. Despite the growing body of work on XCS and the advantages claimed for it there has been no detailed comparison of XCS and traditional strength-based systems. This work takes a step towards rectifying this situation by surveying a number of issues related to the change in fitness. I distinguish different definitions of overgenerality for strength and accuracy-based fitness and analyse some implications of the use of accuracy, including an apparent advantage in addressing the explore/exploit problem. I analyse the formation of strong overgenerals, a major problem for strength-based systems, and illustrate their dependence on biased reward functions. I consider motivations for biasing reward functions in single step environments, and show that non-trivial multi step environments have biased Q-functions. I conclude that XCS's accuracy-based fitness appears to have a number of significant advantages over traditional strength-based fitness.

Keywords: strong overgeneral classifiers, biased reward functions, accuracy-based fitness, XCS, complete covering maps, exploration

1 Introduction

Learning Classifier Systems (LCS) are a class of domain-independent learning production systems suitable for both supervised learning and Reinforcement Learning (RL). The LCS is most strongly characterised by control being distributed among a population of condition-action rules (called *classifiers*) which are incrementally re-evaluated by a learning mechanism and dynamically generated and deleted by a search mechanism. Traditionally the learning mechanism has been the *Bucket Brigade* algorithm (see [8]), but more recently a related algorithm called *Q-learning* (see [10]), which is very popular in some other types of reinforcement learning systems, has been incorporated. Typically the search mechanism in an LCS is a *Genetic Algorithm* (GA – see [8]).

Reinforcement learning environments are either *single step*, in which case the learner's actions have no influence on which states it encounters in the future,

P.L. Lanzi, W. Stolzmann, and S.W. Wilson (Eds.): LCS '99, LNAI 1813, pp. 143–160, 2000.

or *multi step*, in which case they do. Multi step environments are more difficult as the learner has to consider the long term effects of its actions if it is to learn to behave optimally. (Multi step environments are also more likely to involve sparse and/or delayed rewards, which tend to make learning more difficult.) In reinforcement learning we define a *reward function* to provide feedback on the quality of the learner's actions. The reward function associates a number with each possible state/action combination in the learning environment. The goal of the learner is to obtain as much reward as possible, and the reward function is where we indicate to it what we want done. Each rule in an LCS has a *strength* parameter which is updated towards the rewards it receives from the environment, and is (roughly) an estimate of the reward the system will receive if it is used. When rules advocate conflicting actions their strengths are used in conflict resolution.

LCS have traditionally been *strength-based*, meaning that the strength of a rule is also used as its fitness in the GA. Like traditional systems, Wilson's XCS calculates the strength of rules and treats them as predictions of the reward the system will receive if they are used. However, XCS differs from these systems in its use of *accuracy-based fitness*, in which the fitness of a rule is based on the accuracy with which it predicts the reward the system will receive if it is used. We could say that this bases fitness on the consistency of a rule's strength over time, as rules with consistent strengths have converged to predict a particular value, and so make accurate predictions of it (and so are fit). Rules with persistently varying strength are being successively updated towards different rewards (because they are overgeneral), and so make inaccurate predictions of them (and so are unfit). Section 2 has some examples.

The creator of XCS, Stewart Wilson, motivates the switch to accuracy-based fitness in [1], but this is the only discussion in the literature and a comparison of the two approaches has been lacking. A better understanding of the two approaches is important for a number of reasons. First, there is evidence that traditional strength-based fitness is unsuitable for some multi step environments [1,2]. Second, there is a growing body of evidence that accuracy-based fitness *is* suitable for multi step environments (e.g. [1,3]). Third, XCS has generated considerable interest and has become a major focus of LCS research (look at the proceedings of GECCO-99). Fourth, advantages of accuracy-based fitness have been claimed, including better (even optimal) generalisation (and consequently smaller population sizes), and complete mappings from inputs and actions to reward predictions [1,4,5]. Fifth, I suggest later in this work that XCS also has important advantages in handling the explore/exploit dilemma, and what I've called strong overgeneral rules. Sixth, in addition to its apparent advantages, XCS may have a disadvantage in terms of the population sizes it requires.

A consequence of accuracy-based fitness which soon becomes apparent is that *all* accurate (consistent) classifiers are maintained in the population. This includes both those which are consistently high rewarding and those which are consistently low rewarding. In contrast, strength-based fitness (ideally) only maintains those classifiers which are consistently high rewarding. As a consequence,

it seems the size of the population needs to be larger if we use accuracy-based fitness, which is a disadvantage as it requires more computational power. Why would we want to maintain classifiers which have consistent strengths but which consistently receive low rewards?

Despite the attention XCS is receiving, understanding of the differences between it and other LCS is limited. Although it has much in common with the earlier minimalist ZCS, XCS is a complex system and it is not fully clear how its features interact. The work presented here is part of an ongoing effort to understand the consequences of adopting different approaches to fitness calculation. A number of issues have been identified, but none thoroughly examined. Much of this work is speculative and needs not only further theoretical development but empirical validation.

To simplify the comparison of strength and accuracy-based fitness I looked at systems which were as similar as possible. I did this by starting with XCS and making the minimum changes required to convert it to a strength-based system. This involved a number of relatively minor changes to get the system to work in practice, but left the architecture of the system unchanged. This approach was necessary because there are a number of differences between XCS and other LCS, and I wanted to factor the others out and study only the difference in the fitness calculation. (I've focused on the accuracy-based fitness of XCS to the exclusion of its other valuable features.) In my strength-based system, a rule's fitness is simply its strength, which, in single step environments, is calculated using the delta rule: $s_{t+1} \leftarrow s_t + \beta(R_t - s_t)$ where s_t is the strength of rule s at time t, $0 < \beta \leq 1$ controls the learning rate, and R_t is the environmental reward at time t. Updates occur for all classifiers in [A], the set of rules matching the current input and advocating the action chosen by the system. In multi step environments the strength update is the Q-learning update adapted for LCS: $s_{t+1} \leftarrow s_t + \beta(R_t + \gamma \max_a P_{(a,t+1)} - s_t)$ where $0 < \gamma \leq 1$ is a discount factor controlling the value placed on future rewards and $\max_a P_{(a,t+1)}$ is the highest value associated with any action a on the following time step. The same updates are used in XCS, although XCS goes on to derive various other parameters from strength (see [1,6] for details).

The resulting strength-based system resembles Wilson's earlier ZCS [7] – indeed the strength calculation is almost equivalent – although a few minor differences remain. Unfortunately space does not permit consideration of other types of strength-based LCS, but hopefully much of the analysis will be more broadly applicable. Note that both the strength and accuracy-based systems considered here are Michigan type LCS, both use the same delta rule/Q-learning updates (rather than the older Bucket Brigade algorithm), neither uses any form of tax (though they use discounting in multi step environments), classifier bids are not deducted from their strengths, and that XCS does not support default hierarchies (see, e.g., [8]) because they involve inherently inaccurate classifiers.

2 Strength-Based Fitness

Before comparing strength and accuracy-based systems let us define some terms and look at some known problems with strength-based systems to further our understanding of them. First some terms. For strength-based systems, a classifier is *correct* when, in a given state, it advocates the best action, and *incorrect* when it advocates any other action. In single step environments the best action is the one which returns the most immediate reward, while in multi step environments the best action is that which leads to the most reward in the long run. Now for two problems which LCS researchers have been aware of for many years:

Overgeneral rules For strength-based systems let's define an overgeneral rule as one which matches in multiple states and is incorrect in some. E.g. an overgeneral which matches 10 states may be correct in as many as 9 or as few as 1. Even overgenerals which are most often correct are (by definition) sometimes incorrect, so using them can harm the performance of the system.

Greedy classifier creation Often the reward function will return different rewards for correct actions in different states, e.g. one classifier might receive a reward of 50 for acting correctly, while another might receive a reward of 100 for acting correctly. Classifiers which receive more reward from the environment have higher strength/fitness and so are more likely to be selected for reproduction than classifiers which receive less reward. If this tendency to reproduce fitter classifiers is too strong then there may be gaps in the system's "covering map", i.e. there may be lower reward states for which the system has no applicable classifiers. When the system encounters such states it must invent new rules on the spot, which results in poor performance (at least in the short term).

Cliff and Ross [2] showed that the strength-based ZCS can have serious difficulty even with simple multi step environments. They attributed ZCS's difficulties to the two problems above, and, in particular, to their interaction:

Interaction: strong overgenerals The problems of greedy classifier creation and overgeneral rules interact when an overgeneral rule acts correctly in a high reward state and incorrectly in a low reward state. Using traditional strength-based fitness, the overgeneral will always have higher strength than a correct rule which applies only in the lower reward state. This can cause two problems when rules compete. First, because action selection is done on the basis of strength the overgeneral rule will have more influence in the low reward state, despite being consistently incorrect there. Second, greedy classifier creation means that the overgeneral rule is more likely to be selected for reproduction, so the consistently correct rule may die out entirely. I've called this interaction effect the problem of *strong overgenerals* as Cliff and Ross did not give it a name. Strong overgenerals are disastrous for the system's performance because of their effects on rule competition.

State	Action	Reward	
0	0	1000	c
0	1	0	
1	0	0	i
1	1	1000	a

State	Action	Reward	
0	0	1000	c
0	1	0	
1	0	0	i
1	1	200	a

Fig. 1. Unbiased reward function (left) and biased reward function (right).

Classifier	Condition	Action	E[Strength]
A	0	0	1000
B	0	1	0
C	1	0	0
D	1	1	1000
E	#	0	500
F	#	1	500

Classifier	Condition	Action	E[Strength]
A	0	0	1000
B	0	1	0
C	1	0	0
D	1	1	200
E	#	0	500
F	#	1	100

Fig. 2. Expected strengths of all possible classifiers for the two functions.

Although Cliff and Ross discussed these problems with respect to the multi step case, they clearly also apply in the single step case, although not as severely as there is no possibility of disrupting sequences of actions in the single step case. Strength-based LCS have traditionally had some success with single step environments but relatively little with multi step environments. We might attribute the difference to the greater severity of the above problems in the multi step case. However, strength-based systems can also perform poorly in simple single step environments, a problem which Cliff and Ross did not discuss.

Let's extend the analysis of Cliff and Ross to see in which environments these problems can occur. First, for strength-based systems I define a strong overgeneral as an overgeneral rule which has greater strength than some correct rule with which it competes for action selection and/or for reproduction. Two conditions must be met for strong overgenerals to emerge: i) at least two states must return different rewards (so that we can have a high reward and a low reward state), and (trivially) ii) it must be possible to act incorrectly in a lower reward state (i.e. there must be more than one action available) so that having strong overgenerals makes a difference to the learner's performance.

The reward returned for acting in a state is defined by the experimenter's reward function. Figure 1 defines two reward functions for a simple environment: the binary state, binary action identity function. We'll call a reward function *unbiased* if all correct actions return the same reward regardless of state. The reward function on the left is unbiased. The one on the right is *biased*: different rewards are received for acting correctly in different states.

Figure 2 shows all possible classifiers for our example environment using the standard classifier representation. A & D always respond correctly, B & C always respond incorrectly, and E & F are overgeneral, responding correctly in one state and incorrectly in the other. If we use the XCS strength update and

assume that states and actions are chosen equiprobably we obtain the expected strength values shown for the classifiers using the two reward functions. Using the biased reward function E is a strong overgeneral: despite being incorrect half the time it has higher strength than D (which is always correct).[1]

Having a biased reward function *can* lead to the problems of greedy classifier creation and strong overgenerals which Cliff and Ross discussed, but it is difficult to determine just when these problems will occur, and how serious they will be, because there are many factors involved. The degree of bias in the reward function is certainly an important factor. Let's begin a simplified analysis by labelling the rewards in figure 1 from the overgeneral rule's perspective. Let c be the reward when the overgeneral acts correctly (state 0 action 0 in the biased reward function), i the reward when acting incorrectly (state 1 action 0), and a the reward for the accurate rule which applies only in the lower reward state (state 1 action 1).[2]

Now let's see what reward functions will cause E to be a strong overgeneral. Assuming E is updated towards c and i such that its strength is the average of the two (which is an oversimplification), it will be a strong overgeneral if $(c+i)\ /\ 2 > a$. Solving for c yields: $c > 2a - i$. If we use the values of 0 and 200 for i and a as in the biased reward function then $c > 400$ will result in E being a strong overgeneral.

Clearly the rewards defined by the reward function play a major role in determining whether strong overgenerals are possible. Unfortunately, the above analysis is a gross oversimplification of more realistic learning problems, in which it can be very difficult to determine how much of a problem strong overgenerals are likely to be. Many factors are involved, including:

the reward function - what the relationships between all possible c, i and a triplets are.

the classifiers - they often apply in many states, not only two which in isolation make strong overgenerals possible.

the explore/exploit policy - RL systems need to balance exploration of new possibilities with exploitation of existing knowledge. The strategy adopted affects how often classifiers are updated towards their different rewards.

the frequency with which given states are seen - in the single step case this depends on the training scheme, and on the learner and the environment itself in the multi step case.

the selection mechanisms - how high selective pressure is in reproduction and deletion.

the fitness landscape - to what extent strong overgenerals compete with stronger correct rules.

[1] We could have simplified the example somewhat by defining only action 0 for state 0. Action 1 has no effect on the development of strong overgenerals in this example.

[2] Note that i and a need not be associated with the same state in order to produce a strong overgeneral. If a panmictic GA is used, competition for reproduction occurs between all rules in the population.

As a simple example of these factors, an overgeneral might act correctly in 10 states with reward c and incorrectly in only 1 with reward i, so its strength would be more like $(10c + i)/11$, and it would be a strong overgeneral if this value exceeded some a. Similarly, it might match in 10 states with reward i and only 1 with reward c.

Although the complexity of the issue makes a more complete analysis difficult it should be clear that the nature of the reward function is an important factor in determining the prevalence of strong overgenerals, and that they are not uncommon.

In the mainstream RL literature strength-like values are often stored using look-up tables with an entry for each state/action pair. Such tabular systems are relatively insensitive to the form of the reward function, which may account for the lack of attention this subject has received in the mainstream RL literature. Strength-based LCS, however, are clearly sensitive to the form of the reward function, an important difference between them and tabular reinforcement learners. It is curious that the form of the reward function has not received more attention in the LCS literature given their sensitivity to it.

2.1 Multi Step Environments

In single step environments a reinforcement learner approximates the reward function defined by the experimenter, which is in principle enough to maximise the reward it receives. In multi step environments, however, consideration of the reward function alone is not sufficient, as it defines immediate reward (the reward on a given time step) only. In multi step environments the learner's actions influence the state of the environment and hence which rewards it may receive in the future. Consequently the learner must take into account future consequences of current actions if it is to maximise the total amount of reward it receives over multiple time steps. Q-learners do so by learning a *Q-function* (also called an *action-value function*) which maps state/action pairs to an estimate of their long term value (called their *Q-value*). (Q stands for quality of the state/action pair.) The value of a state is the value of the best action for that state, i.e. the state/action pair with the highest Q-value. The Q-value of a state/action pair is updated towards the immediate reward it receives plus some fraction of the Q-value of the state which follows it – see the update shown in section 1. (We say the value of the following state is discounted and passed back to its predecessor.) In this way a state/action pair which receives an immediate reward of 0 will have a Q-value greater than 0 if the state it leads to has a Q-value greater than 0. Discounting is illustrated in a simple environment in figure 3 which shows the immediate reward R and the Q-value Q for each state transition assuming a discount rate γ of 0.9. Knowledge of the true Q-function for an environment is sufficient to act optimally by simply taking the action with the highest associated Q-value.

In an LCS using Q-learning the approximated Q-function is represented by classifiers whose strengths are Q-values. Classifier systems have traditionally

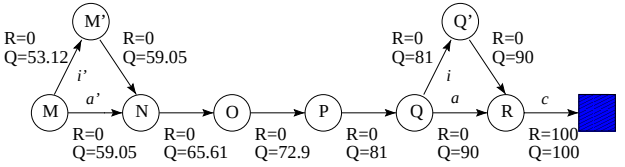


Fig. 3. A simple multi step environment showing immediate rewards R and Q-values Q for state transitions using $\gamma = 0.9$. M is the start state and the square state is terminal.

used the Bucket Brigade algorithm rather than Q-learning, but they both pass values back, and they have the same need for discounting.

The discount rate γ controls how much consideration the system gives to future rewards in making decisions. At $\gamma = 1.0$ no discounting occurs, and the system will learn the path which results in the most reward, regardless of how long the path is. This is often not what we want. For example, being paid £5 a year from now is not as desirable as being paid £5 today. If we used $\gamma = 1.0$ in figure 3 then both the i and a transitions would have Q-values of 100 and the system would be unable to choose between them. At the other extreme, if we set γ to 0.0 the system will be shortsighted and take no interest in the future consequences of its actions. This is often undesirable, as it would lead the system to choose £5 today rather than £1000 tomorrow. In figure 3, $\gamma = 0.0$ would give i and a Q-values of 0, and again the system would be unable to choose between them. Typically we will want to set γ to some value between 0 and 1 in order to give possible future rewards a suitable weighting.

Passing values back from one state to another is necessary for the system to take future consequences of its actions into account. Discounting of these values is necessary for the system to find shorter paths through the state space. However, passing values back and discounting tend to produce one of the two criteria for the production of strong overgenerals: a biased Q-function. Figure 3 demonstrates this effect. Notice that even though there are only two different immediate rewards (R is either 0 or 100), there are many Q-values. Even though the reward function has only two different rewards for correct actions (and so is biased) the Q function has many different rewards for correct actions (and so has more biases than the reward function).

The other criterion for the formation of strong overgenerals, that it be possible to act incorrectly, is met in all non-trivial environments, specifically, in all environments in which we have a choice of alternative actions in at least one state. Let's look at some examples. Imagine the situation where an overgeneral matches in state R and advocates a transition to the terminal state, and also matches in Q and advocates an incorrect (suboptimal) transition to Q′. Additionally, a different, correct, rule matches only in state Q and advocates the correct transition to state R. Assigning the Q-values from the transitions so labelled to c, i and a in the inequality $(c + i)/2 > a$ we saw earlier we obtain $(100 + 81)/2 > 90$ or $90.5 > 90$. In other words, the overgeneral is a strong

overgeneral as it has strength 90.5 which exceeds the correct rule's strength of 90. This demonstrates that strong overgenerals can be obtained even with very short sequences of states.

If the reward received at some state s is $R(s)$, discounting results in $R(s)\gamma$ being passed to the state preceding s. More generally, a state n steps ahead of s receives $R(s)\gamma^n$ from s. If we rewrite a as $c\gamma^n$ and i as $c\gamma^m$ where n and m are integer distances of a and i from c, we have $(c + c\gamma^m)/2 > c\gamma^n$. This expression is true for any $c > 0$ as long as $0 < \gamma < 1$ and $n \geq 1$. In other words, according to our approximate expression, *any* discounting will produce a Q-function capable of supporting strong overgenerals in this environment. Of course our calculations have been greatly oversimplified, but it should be clear that all but the simplest multi step environments can support at least some strong overgenerals.

Now let's look at another example, in which the overgeneral matches in states R and M, and the correct rule matches only in state M. We now use the Q-values labelled a' and i' for a and i in $(c + i)/2 > a$ and obtain $76.56 > 59.05$. Notice that in this example the overgeneral acts incorrectly farther from the goal than in the first example, but its strength exceeds the threshold required of a strong overgeneral by a greater amount. The farther i and a are from the goal, the stronger the strong overgeneral will be, compared to the correct classifier. Notice also that the farther i and a are from the goal, the easier it is to produce strong overgenerals because there are more state transitions in which c can occur and gain enough strength to produce a strong overgeneral.

2.2 Summary of Strength-Based Fitness

To summarise, strength-based classifier systems:

- allocate rules in proportion to the (estimate of the) reward (or, in multi step environments, Q-value) associated with a state/action pair.
  - This introduces the problem of greedy classifier creation.
- can produce overgeneral classifiers which are stronger than some correct ones with which they compete.
  - Despite being unreliable, strong overgenerals have more influence and better chances of survival than some correct rules with which they compete.
  - It can be difficult to predict just how much of a problem this will be.
- do not seem suitable for single step environments with immoderately biased reward functions as many strong overgenerals will arise in them. The more biased the reward function, the more difficulty the system will have.
- do not seem suitable for non-trivial multi step environments as many strong overgenerals will arise thanks to inherent biases in Q-functions.

3 Why Bias the Reward Function?

The reward function is the means by which we define the goals of our learning systems when we design reinforcement learning experiments (see [10]). The

reward function is part of the definition of the problem; choosing different reward functions results in different learning problems. The trick is to choose a reward function which results in the desired behaviour. Given that the experimenter defines the reward function, potential problems with biased reward functions can be avoided by using unbiased ones (or only slightly biased ones). Why use biased reward functions if they can cause problems for strength-based system? One plausible reason to bias the reward function is to get the system to allocate more resources (classifiers) to more important parts of the environment. For example, we might consider the detection of an impending meltdown to be more important than optimising the output of a reactor. This is analogous to giving more weight to minimising errors over parts of a training set in a neural network. Seen in this light, the problem of greedy classifier creation from section 2 is really a useful feature rather than a harmful bug. A rule allocation bias only becomes a problem when it becomes too extreme and causes gaps to appear in the covering map. In any case, if we're really concerned with gaps in the covering map we should be able to modify our rule allocation algorithms to ensure they don't occur. We could, for example, only delete a rule if all the states it matches are also matched by some other rule (or rules).

Even if greedy classifier creation is not a problem biased reward functions may still result in strong overgenerals. If we want to avoid biasing the reward function, we may be able to achieve the same effect on rule allocation by varying the frequency with which individual states are encountered, so that more important states are presented more frequently. In single step environments we may have complete control over which states are visited. In multi step environments, however, the agent and environment determine which environmental states are encountered, not the experimenter. When we cannot control the environment we may be able to record experiences and play them back in simulation with a frequency bias towards more important ones. Alternatively, we could split the classifiers into multiple subpopulations, each with a different size limit, and assign each to a subset of environmental messages. More important parts of the environment would get larger subpopulations. This makes the system more complicated, but has the advantage of allowing the use of domain knowledge to generalise a priori by limiting the input to each subpopulation (see [11,12]). A third alternative is simply to define a weighting function for each state/action pair, and to modify the system to take it into account when allocating rules.

In multi step environments we generally end up with a biased Q-function as a result of backing up values with Q-learning, regardless of whether our reward function is biased. There is an additional need to purposely bias the reward function in multi step environments because the learner's actions affect which states it will see in the future. If we define multiple goals we need to tell the learner about their relative importance, so it knows which goals to pursue if it must choose between them. The effect of discounting is to introduce a bias in rule allocation towards states which are closer to a source of reward. It is not clear to me whether this bias is desirable or not. (Wilson makes this same point in [7] in section 5.3.)

4 Accuracy-Based Fitness

Information on the accuracy of rules has been used in primarily strength-based systems (see [1] for a review), but the only fully accuracy-based LCSs are Wilson's XCS and Frey and Slate's system [13]. Although the latter is significant this analysis is based around XCS.

A theme running through this paper has been the effects the form of the reward function have on classifier systems. In the preceding sections we've seen that strength-based systems are very sensitive to bias in the reward function, that biases are very common, and even some reasons why we may want to introduce them. We've also seen that biases in the reward function can result in strong overgenerals in strength-based systems.

A very important property of XCS's accuracy-based fitness calculation is that such biases do not result in strong overgenerals which dominate the system. In this sense XCS is insensitive to bias in the reward function. Before we see why, let's reconsider some definitions.

4.1 Different Definitions of Overgenerality

I use different definitions for overgeneral rules to match the different objectives of the two types of LCS. Using the strength-based definition an overgeneral is a rule which matches in multiple states and is incorrect in some. However, for accuracy-based fitness I define an overgeneral rule as one which is updated towards sufficiently different values in different states. The values must be "sufficiently" different because the reward for a given state/action may be noisy, and in any case it may be desirable to consider generalisations over state/actions with different values to be accurate. This tolerance for differences in values is present in XCS in the form of various parameters of the classifier update rules. The definition of strong overgenerality also differs for accuracy-based systems: a strong overgeneral is an overgeneral which has greater strength than some accurate rule with which it competes for action selection, or greater fitness than some accurate rule with which it competes for reproduction.

XCS seems to deal well with overgenerals because, by definition, they are updated towards different values and so have low accuracy and low fitness. Further, it seems strong overgenerals have particular difficulty because overgenerals have lower fitness than their accurate competitors. Preliminary empirical results (not shown) support these statements, but more thorough tests are needed.

Our two definitions of overgenerality match differences in how the two types of system treat certain rules. Under accuracy-based fitness generalisation is sensitive to the magnitude of Q-values; in fact, it is differences in Q-values which limit generalisation (since a rule is only fit if its strength is sufficiently consistent). But under strength-based fitness generalisation has no such limitation. Let's look at some examples.

Suppose the reward function is biased in such a way that a rule receives the same reward when acting correctly in one state and incorrectly in another. We obtain this situation if we modify the reward function on the left of figure 1 so

that state 1 returns rewards of 1000 and 2000 for actions 0 and 1. Now rule E is overgeneral according to the strength definition (as it is correct in state 0 but incorrect in state 1), but not overgeneral according to the accuracy definition (as its strength is updated towards the same value and so is consistent).

As another example, a rule which matches in states O and P in figure 3 will be updated towards two Q-values (72.9 and 81) and its strength will oscillate somewhere between these values. With fitness based on strength this rule will have a reasonable fitness compared to others in this environment. But with fitness based on accuracy, the oscillations may result in low fitness – whether they do depends on the level of tolerance for oscillations in strength.

Although accuracy-based fitness is not susceptible to one effect of biases in the reward function (strong overgenerals), it is still very sensitive to the form of the reward function because it can only generalise over state/action pairs whose Q-values differ by some tolerable amount.[3] Strength-based fitness does not have this limitation: syntactic limitations notwithstanding, strength-based classifiers are free to match any and all states for which their action is optimal. So strength has an advantage in that it can express more useful generalisations than accuracy. Unfortunately, strength-based classifiers are also free to match states for which their action is suboptimal, and, in standard strength-based systems, there seems to be nothing preventing them from doing so – they are free to be overgeneral.

In strength-based systems the idea seems to be that these overgenerals will become the default rules in default hierarchies; that their inappropriate behaviour will be overriden by more specific exception rules.

4.2 Different Goals, Different Representations

Although the difference in fitness calculation between strength and accuracy-based LCS may seem minor it has profound implications. One is the form of the covering map of classifiers the system maintains. Strength-based systems, due to greedy classifier creation, tend to allocate more rules to higher rewarding states, and to higher rewarding actions within a state. In effect they attempt to find rules which advocate the best (i.e. highest rewarding) action for each state and in the extreme case each state would only have a single action (its best) advocated by some rule(s). Let's call this extreme case a *best action map*. (Note that best action maps do not entail gaps in the covering map, as the latter involve states with no matching classifiers at all.) In practice, however, strength-based systems only tend towards best action maps – often for a given state there is more than one matching classifier. We could say they maintain *partial maps*. The important point is that there tend to be states with (typically low-rewarding) actions unadvocated by any classifier. Since the action selection mechanism can only make an informed choice between advocated actions, the tendency towards a best action map means the rule allocation mechanism has a hand in action selection. Best action maps are in a sense ideal as they represent

[3] This seems a limitation of dynamic programming based systems rather than of XCS or LCS per se.

the solution with a minimal number of rules, but this doesn't imply partial maps are an ideal representation for finding solutions – see sections 5 and 6.

In contrast to the partial maps of strength-based systems, the idea in XCS is to find a population of rules such that each action in each state in advocated by at least one rule. Wilson calls this a *complete map*, and has noted that it resembles the representation of a tabular learner more closely than does a partial map [1]. When all actions are advocated, it is left entirely to the action selection mechanism to decide which action to take; the rule allocation mechanism is dissociated from action selection. This is consistent with the role of the GA in XCS, which is only to search for useful generalisations over states.

4.3 Different Population Size Requirements

There are several reasons why we might think strength-based systems should require fewer rules to solve a problem. First, as discussed in section 4.2, strength-based systems support partial maps, which involve fewer rules than the complete maps of accuracy-based systems. A complete map will be n times larger than a best action map for the same environment, where n is the number of actions available. In practice, however, the difference will be less as strength only tends towards best action maps. Second, as we saw in section 4.1, strength-based systems can express more useful generalisations than accuracy-based systems. Third, accuracy cannot support normal default hierarchies because default hierarchies involve classifiers which are inherently inaccurate.[4]

However, XCS may have an advantage in being better at generalising than other LCS. According to Wilson's *Generalization Hypothesis* [1] there is a tendency in accuracy-based XCS for the more general of two equally accurate rules to reproduce more. This results in a tendency to evolve *maximally general rules*: rules which cannot be made more general without becoming inaccurate. These accurate, general rules form a compact representation of the learned solution to the problem environment. Consequently, XCS seems to be able to evolve populations which are smaller than those of standard strength-based systems. In fact, in [4] XCS consistently evolved *optimal solutions* for boolean (multiplexer and parity) functions. These are solutions which are complete (cover all parts of the input/action space), non-overlapping (no part of the space is described more than once) and minimal (the minimal number of non-overlapping rules is used). With minor extensions XCS was able to extract the optimal solutions from the general classifier population.

It is not clear at present how the population sizes of the two types of system compare in practice, or whether the better generalisation of XCS is exclusive to its accuracy-based fitness. It may be possible to achieve similar success at generalisation in strength-based systems, at least on some tests. Limited comparisons (not shown here) using the 6 multiplexer test showed smaller population sizes

[4] It should be possible for accuracy to use classifiers with multiple conditions which form their own internal default hierarchies. However, it is not clear to me that this would improve the system.

using accuracy. However, this test has only two actions in each state. Strength-based systems may scale better than accuracy as the number of actions increases because of their partial mappings.

In any case, population sizes may not be terribly important. Memory is cheap, and since the differences in population size should be relatively small, so would be the differences in processing power required.

4.4 Accuracy and Biased Reward Functions

In section 3 we saw that we might actually want to bias the reward function in order to allocate more rules to more important parts of a problem. However, we can't do this with XCS because it doesn't share strength's rule allocation bias towards state/action pairs with higher rewards. XCS bases fitness on the accuracy (consistency) of strength rather than strength's magnitude, so, for example, a classifier which accurately predicts a reward of 0 will be as fit as one which equally accurately predicts a reward of 100. So there is no point in using a biased reward function with XCS – in fact, it's easier for XCS to generalise if the reward function is unbiased because differences in rewards limit the generalisations which can be made (section 4.1). However, we also saw in section 3 that we should be able to bias the allocation of rules in other ways, and these should apply to XCS as well as other LCS.

In multi step environments Q-learning almost always produces a biased Q-function. This affects the generalisation an accuracy-based system can do, but XCS appears to work well in multi step environments nonetheless (as demonstrated in a number of (admittedly simple) environments in, e.g., [1,3]).

In section 3 I suggested that we might need to bias the reward function in multi step environments in order to give different goals relative importance. Although XCS doesn't bias rule allocation according to the reward function, action selection is still (essentially) done on the basis of strength, not accuracy-based fitness. Thus, XCS can distinguish between high and low rewarding actions, high and low priority goals, and find shortest paths to goals.

4.5 Summary

Accuracy-based fitness:

- does not allocate rules in proportion to the reward (or Q-value) associated with state/action pairs (unlike strength-based fitness).
  - so we can't bias rule allocation by biasing the reward function.
  - but we can still bias rule allocation in other ways.
- appears to deal well with overgeneral and strong overgeneral rules. Consequently, unlike strength-based systems, it seems suitable for multi step environments and single step environments with highly biased reward functions.
- cannot express all the useful generalisations which strength can, but, unlike strength, can distinguish between accurate and overgeneral rules.

- tends to maintain both consistently correct and consistently incorrect classifiers. This constitutes a complete covering map.
- may require larger populations than strength because of the complete map, a restricted ability to express useful generalisations and a lack of default hierarchies. However, its better generalisation may offset this.

5 Rule Allocation Reconsidered

In section 3 I suggested we might want to bias the reward function in order to bias rule allocation. But do we really want to bias rule allocation according to the reward or Q-function? The goal of any reinforcement learner is to maximise the reward it receives, so a rule's importance is determined by its contribution to obtaining reward. This means rules which lead to higher rewards are more important. But rules can contribute indirectly to maximising reward, so the importance of a rule is not the same as the desirability of taking the action it advocates. Surely rules which can save us from disasters are also important; as well as knowing what to do, we may need to know what *not* to do.

For example, a robot might have a rule whose use would lead it to fall down a flight of stairs and damage itself. We definitely do not want to take the action advocated by this rule, so we associate a very low reward (or, equivalently, a large negative reward) with this outcome. At the same time, however, we definitely do not want the robot to forget this rule, because it is very important not to forget the consequences of its use. Not all environments have disasters like this, but some do.

Avoiding disasters seems like an extreme case of the more general problem of managing exploration.[5] After all, once we've learned to behave satisfactorily in our environment we no longer need to remember what not to do – all we need is to remember what to do. It's only when exploring – when there is some chance of taking what appears to be a suboptimal action – that we need to explicitly guard against taking very suboptimal actions.

But it is not only when there are potential disasters to avoid that we can benefit from keeping track of suboptimal actions. Rules may well appear suboptimal and yet if we haven't tried them (and the alternatives) sufficiently we can't be sure they really are suboptimal. A single trial is sufficient in deterministic single step environments, but in other cases many trials may be needed to find a good estimate of the true value of the rule. Exploration control is extremely important to efficient reinforcement learning, particularly in multi step environments, because inefficient exploration can be very wasteful of trials.

Recall that a complete map contains both consistently correct and consistently incorrect classifiers. However, maintaining consistently incorrect classifiers in the population does not mean we have to take an incorrect action each

[5] In reinforcement learning there is a tradeoff between exploiting current knowledge of the environment and exploring the environment so that we might exploit it better in the future. This is often referred to as the explore/exploit dilemma.

time one is applicable. In fact, quite the opposite is true. If a classifier is consistently incorrect, this suggests we should *not* take the action it advocates. If we did not keep this consistently incorrect classifier in the population, how would we know it was a bad idea to take its action? If we delete it, we have no record of the utility, or lack thereof, of taking that action in that state, and we might be tempted to try the suboptimal action again and again in the future. We need to keep incorrect classifiers in order to keep track of guesses which have not paid off; in other words to manage our exploration.

If we do maintain rules which tell us how to avoid particularly bad outcomes we are maintaing a more complete map than tranditional strength-based systems do. To keep track of all actions requires a complete map.

6 Complete Maps and Best Action Maps

A complete map has the disadvantage of requiring more classifiers but lets us control exploration better. The more difficult the exploration problem, the more advantageous a complete map will be. Two cases in which exploration is more difficult are i) when the environment changes over time, and ii) in multi step environments.

Hartley [14] trained two LCSs, XCS (which maintains a complete map) and NEWBOOLE (which maintains a partial map), on a binary categorisation task, then abruptly switched the category each stimulus belonged to. XCS quickly recovered from these changes by simply adjusting the strengths of the rules involved: consistently correct rules suddenly became consistently incorrect and vice versa. NEWBOOLE, in contrast, found that its rules suddenly all had low strength, and had to engage the genetic algorithm to generate new ones as it does not maintain low strength rules. A complete map should also be useful for adapting to less systematic and more gradual changes in the environment, although this has not been studied.

In multi step environments globally optimal behaviour often requires that the learner take locally suboptimal actions (i.e. actions which do not return the highest possible immediate reward). E.g. the learner must take action B, knowing that action A results in greater reward, because only action B will lead it to state Q where it can obtain even greater reward. The best way to ensure that such sequences of actions can be learnt is to use a complete covering map. Note that tabular Q-learners employ complete maps.

If more complete maps are useful, can we get strength-based systems to maintain them? We could give rewards of 90 for incorrect actions, and 100 for correct actions. If selective pressure is not too strong the system should be able to maintain classifiers for both correct and incorrect actions (i.e. a complete map). However, overgeneral rules would always have more strength than accurate incorrect rules.

Alternatively we could modify a strength-based system to distinguish between strength and fitness, and make the fitness of a rule something like the difference between its strength and (maximum strength - minimum strength)/2.

As a result, this strength-based system would no longer consider the importance of a rule proportional to the reward it leads to. We should be able to maintain complete maps in this way, although it will not work with arbitrary reward functions because of strong overgenerals. (The distinction between strength and fitness is one of the major features of XCS, and such a change would be one step towards an XCS-like system.)

Further study is needed to confirm and quantify the advantages of complete covering maps for exploration control, and to determine how much of an increase in population size they require.

7 The Big Picture

That an estimate of accuracy should be a useful fitness metric for LCS should not be surprising; accuracy is a measure of the utility of generalisation and LCS can be seen as searching a space of generalisations. Although LCS typically use GAs as search engines, the GA in an LCS differs from a *stand-alone GA* (i.e. one not incorporated into an LCS). Let's contrast the LCS GA with a stand-alone GA applied to the typical task of function optimisation. Function optimisation does not involve searching a space of generalisations, and consequently requires a different fitness metric. Function optimisation GAs attempt to find parameter settings which correspond to the extremum of the fitness function (or, in RL terminology, the reward function). They include no concept of environmental state (and so lack the condition part of a classifier), and only manipulate a set of parameters (corresponding to the action part of a classifier). Consequently, they have no notion of generalising across states and no need for a measure of the accuracy with which this is done. Their measure of fitness is just the strength value in a strength-based classifier system. For this purpose it is quite suitable.

The use of strength as a fitness measure in LCS is apparently due to confusion over what the role of the GA within an LCS should be. In function optimisation the GA is a function optimiser seeking only the extremum of the fitness function. Within an LCS, however, it should be seen as a kind of function approximator. Strength-based systems, because of their partial maps, tend to approximate the part of the reward function corresponding to correct actions. Accuracy-based systems, in contrast, approximate the entire reward function because of their complete maps. In both cases the LCS attempts to do so more efficiently by generalising over states. It was easy to make the mistake of retaining the same fitness measure when moving from stand-alone GAs to strength-based LCS.

8 Metasurvey

I've tried to outline some of the issues which arise when considering alternative fitness calculations in LCS. We've seen an extension to the theory of strong overgeneral rules which suggests strength-based LCS are unusually sensitive among reinforcement learners to bias in the reward function. We've seen why we might or might not want to bias our reward functions, and some ways of doing so. We've

seen that biased Q-functions are common, and that, consequently, strength-based systems seem unsuitable for non-trivial multi step environments. We've seen why complete maps may help with exploration control. Finally, we've seen that accuracy-based XCS does not suffer some of the problems which have plagued earlier systems. In particular, analysis suggests that it does not suffer from strong ovegenerals. All this strongly supports XCS as a major direction of LCS research.

Acknowledgements. I am grateful to Stewart Wilson, Manfred Kerber, Jeremy Wyatt, Adrian Hartley and three anonymous reviewers for their comments.

References

1. Stewart W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995. http://prediction-dynamics.com/
2. Dave Cliff and Susi Ross. Adding Temporary Memory to ZCS. *Adaptive Behavior*, 3(2):101–150, 1995.
3. Pier Luca Lanzi and Marco Colombetti. An Extension of XCS to Stochastic Environments. In W. Banzhaf et al., editors, *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 353–360. Morgan Kaufmann, 1999.
4. Tim Kovacs. XCS Classifier System Reliably Evolves Accurate, Complete, and Minimal Representations for Boolean Functions. In Roy, Chawdhry, and Pant, editors, *Soft Computing in Engineering Design and Manufacturing*, pages 59–68. Springer-Verlag, 1997. ftp://ftp.cs.bham.ac.uk/pub/authors/T.Kovacs/index.html
5. Stewart W. Wilson. Generalization in the XCS classifier system. In J. Koza et al., editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 665–674. Morgan Kaufmann, 1998. http://prediction-dynamics.com/
6. Tim Kovacs. Evolving Optimal Populations with XCS Classifier Systems. Technical Report CSR-96-17 and CSRP-96-17, School of Computer Science, Uni. of Birmingham, 1996. ftp://ftp.cs.bham.ac.uk/pub/tech-reports/1996/CSRP-96-17.ps.gz
7. Stewart W. Wilson. ZCS: A zeroth level classifier system. *Evolutionary Computation*, 2(1):1–18, 1994. http://prediction-dynamics.com/
8. David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.
9. Andrea Bonarini. Evolutionary Learning of Fuzzy rules: competition and cooperation. In W. Pedrycz, editor, *Fuzzy Modelling: Paradigms and Practice*, pages 265–284. Kluwer Academic Press, 1996.
10. Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.
11. Mark Humphrys. *Action Selection Methods using Reinforcement Learning*. PhD thesis, Cambridge University, 1997. http://www.compapp.dcu.ie/~humphrys/
12. Jonas Karlsson. *Learning to Solve Multiple Goals*. PhD thesis, University of Rochester, 1997. http://www.cs.rochester.edu/trs/ai-trs.html
13. Peter W. Frey and David J. Slate. Letter Recognition Using Holland-Style Adaptive Classifiers. *Machine Learning*, 6:161–182, 1991.
14. Adrian Hartley. Accuracy-based fitness allows similar performance to humans in static and dynamic classification environments. In W. Banzhaf et al., editors, *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 266–273. Morgan Kaufmann, 1999.

Non-homogeneous Classifier Systems in a Macro-evolution Process

Claude Lattaud

Laboratoire d'Intelligence Artificielle de Paris5,
Université René Descartes,
45, rue des Saints-Pères,
75006 Paris, France
Lattaud.Claude@math-info.univ-paris5.fr

Abstract. The synthesis of artifacts reproducing behaviors and properties of living beings is one of the main goals of Artificial Life. These artificial entities often evolve according to algorithms based on models of modern genetics. Evolutionary algorithms generally produce micro-evolution in these entities, by applying mutation and crossover on their genotype. The aim of this paper is to present Non-Homogeneous Classifier Systems, NHCS, integrating the process of macro-evolution. A NHCS is a derived type of classical Classifier Systems, CS. In a CS, all classifiers are built on the same structure and own the same properties. With a NHCS, the behavior of artificial creatures is defined by the co-evolution between several differently structured classifiers. These agents, moving in a 2D environment with obstacles and resources, must adapt themselves and breed to build viable populations. Finally, ecological niches and specific behaviors, individual and collective, appear according to initial parameters of agents and environment.

1 Introduction

In A-Life [18], artificial creatures like animats [29], generally evolve according to a single classifier system [17]. These entities use only one type of classifiers and one structure for them. Genetic Algorithms [12], or Evolution Strategies [2], controlling the evolution of a standard CS produces micro-mutations and crossovers inside classifiers. A Non-Homogeneous Classifier System allows agents to possess several kinds of classifiers with different structures. Moreover, creatures can undergo a structural mutation: the macro-mutation, [21]. It acts first on the genotype of the agent by modifying the classifiers architecture, and secondly on the phenotype by adding/removing sensors or other capacities.

P.L. Lanzi, W. Stolzmann, and S.W. Wilson (Eds.): LCS '99, LNAI 1813, pp.161-174, 2000.

In the agent classification of by P. Cariani, [5], the only forms of adaptive agents he considered are natural animals. These agents adapt their behavior according to their environment, but they also adapt their interactions modules, like sensors and effectors. If each type of classifier in a NHCS is in relation with a particular sensor of a creature, then with a macro-evolution process on the NHCS, this agent can be defined as adaptive in the Cariani sense.

This paper presents the structure of the NHCS of artificial creatures and the associated macro-evolution process. The first part shows an overview of the agent model ETIC, [20], used to develop these animats. The second section defines the concept of NHCS and the evolution methods that use the macro-mutation operator. The next chapter describes experimental results obtained on an artificial life application. The fifth part focuses on the idea of macro-evolution used in biology and adapted in artificial life for this paper. The conclusion then discusses results and presents future works about NHCSs.

2 ETIC Overview

A classification is necessary to implement a deep evolution of agents structure. The reactive/cognitive model [9], is not flexible enough for this kind of evolution. Agents built with the External Temporal Internal Classification, ETIC, can change from one class to another, and evolve *progressively* from simple to complex. Each of these classes gives particular functions to agents. The three base classes of ETIC are:

- E: External, an agent belonging to one or more subclasses of E is able to get information from its environment.
- T: Temporal, this class allows agents to have memory and planning functionalities.
- I: Internal, the subclasses of I determine if agents have a knowledge of some of their internal states and if agents can modelize other agents.

The application testing the NHCS method mainly uses five ETIC subclasses:

- Epn: this class represents a non-oriented local perception. An Epn agent perceives all items in its surrounding environment in a range of n[1].
- Epon: the perception of Epon agents is local and oriented. Epon agents perceive only the environment in front of them, according to their direction.
- Edon: agents belonging to this class have an oriented fuzzy perception. They cut the environment in front of them in four dials and they perceive only the number of each kind of objects in each dial. This perception can be wider than the first two.

[1] In a discreet environment, n is a distance of perception defined by the minimum number of elementary cells between two points.

- Eck: this class allows agents to communicate a part of their knowledge to other agents. A communication process can occur if an Eck agent perceives at least one other Eck agent. This communication is direct, point-to-point, between these two agents. The transmitted information is the best classifier of each other.
- Ipenergy: an agent with this class codes its energy level[2] in their classifiers. It thus has an internal representation of its energy and its behavior depends on it.

Fig. 1 shows the perception of three types of agents exploiting E subclasses: Ep1 agent, Epo3 agent facing east and Edo5 agent facing west.

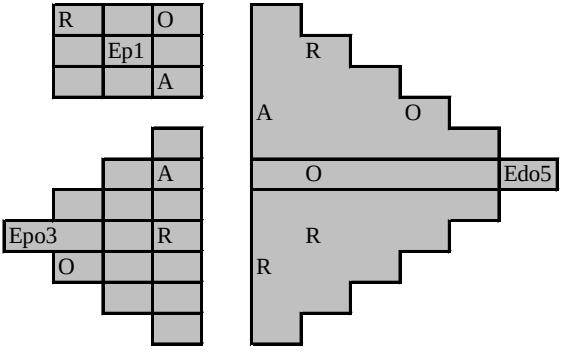


Fig. 1. Ep1, Epo3 and Edo5 agents.

These classes are cumulative, agents can be Ep1Epo3, Epo2Eck, Edo5Ipenergy, etc. The macro-evolution process uses NHCS by adding/removing classes or increasing/decreasing class parameter. This evolution allows a deep modification of agents structure, from a phenotypic and a genotypic point of view. Agents can adapt their morphology and their behavior to the environment.

3 Non-homogeneous Classifier System

A Non-Homogeneous Classifier System is an hybridization of the classical classifier system defined by John Holland. In a standard CS, all classifiers generally have the same structure and same properties. In a NHCS, several types of classifiers coexist and are merged. They are build with different structures and have different properties.

For example, an Ep2Edo7 agent can use either its local precise perception or its fuzzy dial perception to chose an action in a given situation. Such an agent possesses in its artificial organism, after some learning steps, Ep2 and Edo7 classifiers. How-

[2] In this artificial life problem, animats have an energy parameter defining their health level and their fertility for reproduction. If energy falls under 0, the creature dies.

ever, if the agent has undergone a macro-evolution, it could also combine other classifiers like Ep1, Edo1, Edo2, … , Edo6. These classifiers are totally different and must co-evolve to develop adapted behaviors.

The process of macro-evolution is strongly linked to the macro-mutation operator. This operator is added to the three classical operators of GAs: mutation, crossover and selection. It gets its own cycle, determining the frequency with which the macro-mutation can occur, and the following parameters:

- Pmac: the macro-mutation rate.
- Pevol: the evolution rate, representing the probability of evolution/devolution of the agent.
- Pclass: the class-evolution rate, defining if the evolution/devolution process acts on an entire ETIC class or only on the parameter of an existing class of the agent.

Fig. 2 shows the algorithm for the macro-mutation operator.

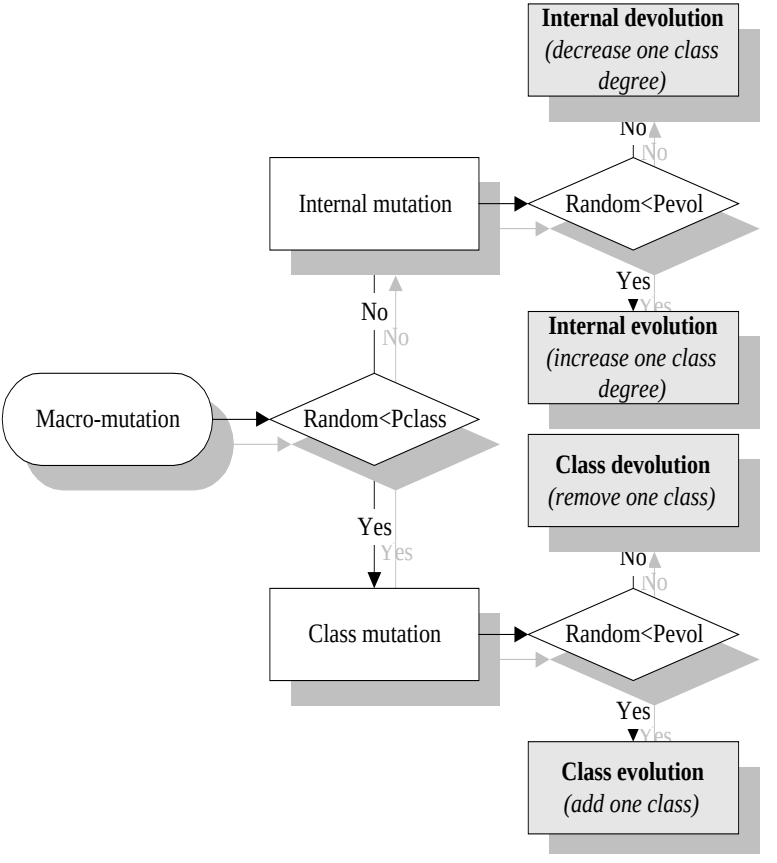


Fig. 2. Macro-mutation algorithm.

For example, if an agent is Ep1Epo3 and undergoes a macro-mutation, four cases can occur:

- Rand1<=Pevol and Rand2<=Pclass: an evolution occurs on an entire class. So, the agent adds a new class, it can become Ep1Epo3Edo1, Ep1Epo3Eck or Ep1Epo3Ipenergy.
- Rand1<=Pevol and Rand2>Pclass: in this case, the evolution takes place on an existing class, the agent evolves to Ep2Epo3 or Ep1Epo4.
- Rand1>Pevol and Rand2<=Pclass: the devolution is on a complete class. The agent loses a class and mutates to Ep1 or Epo3[3].
- Rand1>Pevol and Rand2>Pclass: this is a classless devolution, the agent decreases the parameter of one existing class, but never below 1, so the unique solution is Ep1Epo2.

The macro-evolution described previously is performed according to a cycle, but another way to proceed it exists: During an asexual reproduction. If the reproduction of agents is asexual, two fertile[4] agents are necessary to create a third one. But, if two agents with different classes can mate and give birth to a viable child, how the organism of this offspring is built? In this system, no constraint prevents this reproduction and a macro-evolution on the child NHCS occurs. This resulting NHCS is a combination of parents NHCSs with a potential macro-mutation, with Pmac probability. The extract of the organism shown in Fig. 3 belongs to a Ep1Epo3Edo5Ipenergy[5] agent after several steps of learning and evolution.

Transmission of classifiers from one generation to the next generation follows two simplified models of the genetic:

- Darwinist model: classifiers, acquired knowledge by individuals, isn't transmit to offspring. Only the NHCS, in relation with the morphology of the creature, is used to define the child capacities.
- Lamarckist model: all the knowledge of parents is transmitted to their offspring. Classifiers of parents are combined to obtain classifiers of the new born. In this model, two processes are used when an evolution is performed: a (λ,μ) or a $(\lambda+\mu)$ reproduction, [16].

The next part presents results concerning these reproductions and others about the co-evolution of differently structured classifiers in a NHCS.

[3] If the agent has only one class, it can't lose it.

[4] Age and energy conditions.

[5] The class Iasocial isn't studied in this paper. For information, this agent have the capacity to detect differences between altruists and egoists agents. Altruists can give a part of their energy to other agents, while egoists can't.

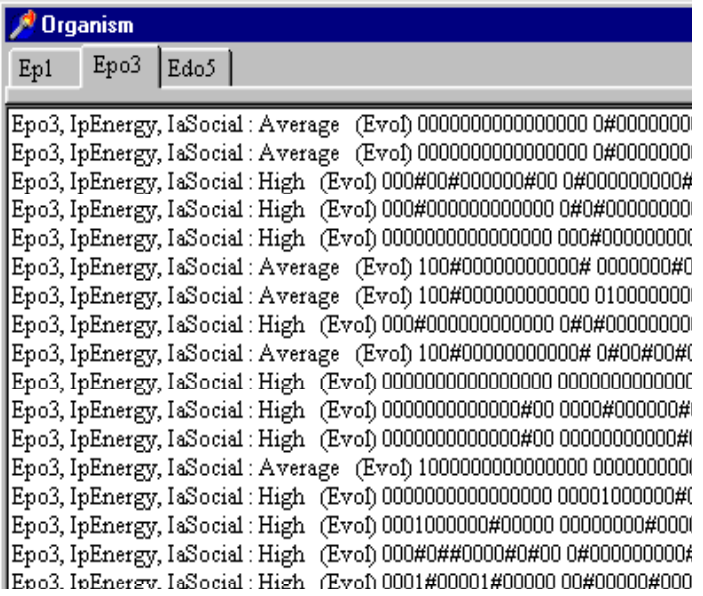


Fig. 3. Organism of a Ep1Epo3Edo5Ipenergy Agent.

4 Results

The aim of this A-Life application is the survival of populations of autonomous agents, [23], in a dynamic environment. They must adapt their behavior, by learning and evolution, and their structure, by macro-evolution on their NHCS. At the beginning of a simulation, agents have no knowledge, they follow the Aristotian principle of the *tabula rasa*. Then, they learn by reinforcement according to the results of their actions. Though this paper focuses only on parameters linked to macro-evolution and NHCS, several parameters are available to users. The GA rates are standard:

- Crossover rate = 0.20.
- Mutation rate = 0.05.
- Selection mode is roulette wheel with stochastic reminder and possibly elitist adjustments, [13].

The environment evolves by adding/deleting/moving obstacles and resources. Its size is fixed to 30*30, and it is toroidal[6]. Two statistical measures are used: Mean and standard deviation. Results are obtained by meaning ten trials of 4.000 time steps each. Simulations with a too high standard deviation are not considered in this paper.

[6] Agents going out by one side, return in the world by the opposite side.

4.1 Population Viability

The aim of the first experiment is to show the viability of different initial populations of agents according to their ETIC classes. At the beginning of a simulation, agents possess the class Ipenergy and one of the subclasses of Epn, Epon and Edon[7]. With these combinations, one value for n seems to be more efficient for each subclass: Ep2, Epo3 and Edo5. Fig. 4 shows the evolution of these populations during 4.000 time steps.

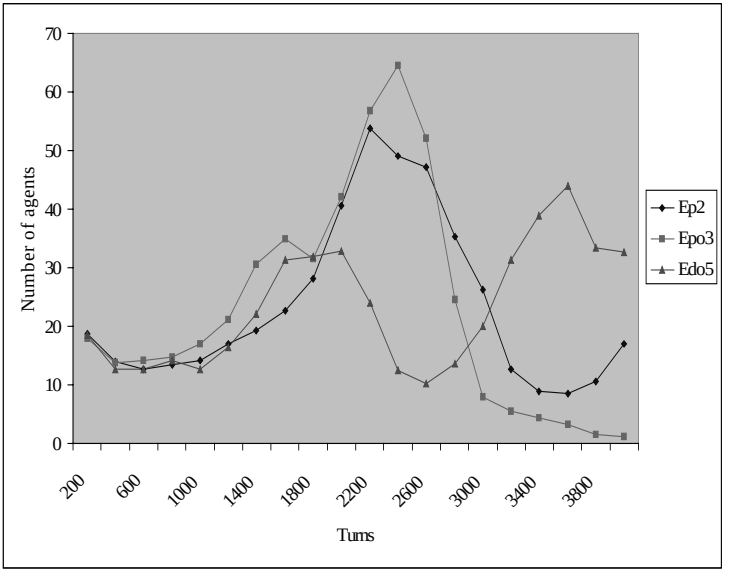


Fig. 4. Ep2, Epo3 and Edo5 agents populations.

The phenomenon appearing in this figure is the same described by Ginger Booth, [3]: the trophic cascade, the inversely proportional co-evolution between agents and resources. At the beginning of a simulation, agents take time to learn basic behaviors, like 'feed with a resource when it's on the same square' or 'approach a resource if possible', to maintain their energy level over 0. The resource quantity increases during this phase, then when resources are large and agents more efficient, they mate quickly and build a wide population. Resources disappear quicker, causing the fall of agents. Only the most adapted animats can survive this starvation, they are few but very efficient. Finally, resources invade again the environment and this process repeats cyclically in a trophic cascade.

The main difference between a Lamarckist and a Darwinist evolution is the quickness of the population growth. Agents have more advantages in the Lamarckist one,

[7] With n varying from 1 to 7.

they develop themselves quicker and contribute to an ampler trophic cascade. In a Darwinist evolution, agents have more difficulties due to the knowledge transmission that doesn't exist during the reproduction. But globally, results are more stable and an equilibrated cycle is reached.

4.2 Ecological Niches

The populations showed in Fig. 4 evolve and their agents also undergo macro-evolution on their NHCS. From generation to the next generation, they mutate to constitute more or less homogeneous populations. According to initial conditions, artificial species often emerge from the system and dominate the global population. The most significant case is the emergence of the communication in the NHCSs: 90% of the final populations include agents with the class Eck. These agents appear progressively along the evolution of species and finally they often invade the global population.

Other kinds of combinations emerge after several thousands steps. Generally, these agents have an efficient co-evolution in their NHCS: Ep2Epo4EckIpenergy, Epo3Edo7EckIpenergy, Ep1Edo7EckIpenergy. All of these have communication abilities and an internal representation of their energy level. The first also combines a precise short perception of its local environment and a medium precise oriented perception. The second has a medium precise oriented perception and a long range fuzzy perception. And, the third has a very short precise non-oriented perception and a large fuzzy perception. These combinations can be found in natural animals. Most of them uses two or more sensors to detect food, predators and other members of their specie, like eyes, ears, sonars, infrareds, vibration sensors, etc. However, these niches are strongly dependant of the initial parameters of the environment. If the conditions are extreme, very few resources and many obstacles, the emerging niche is Ep1Ipenergy. These agents can't lose time in communication and must take resources as soon as they detect them. As Dave Cliff [7] said: "In the ethology literature, an adaptive behavior is any behavior which, if exhibited by an animal, increase the chance that the animal will survive long enough in its ecological niche to produce viable offspring. Underlying this definition is the assumption that, if the animal does nothing, it will die before it has a chance to reproduce". Agents present in emerging niches in this application are precisely very active and they learn particular efficient behaviors, individual as well as collective.

4.3 Emerging Behaviors

Three main behaviors, cf. Fig. 5, that appear in this application are:

- Obstacle avoidance: the agent at the right of the figure shows how it can avoid obstacles. After learning, it avoids obstacles to reach resources in the quickest way.
- Obstacle following: the same agent follows obstacles to get a resource hide behind them. Agents Epn and Epon, which have a local precise visibility, develop generally this two kind of behaviors. Edo agents, with a fuzzy perception, can't avoid and follow obstacles, unless if they are combined with Epn or Epon subclasses.
- Regrouping: when few resources exist in the environment, surviving agents, those at the left of the figure, are generally more efficient. If they perceive a resource, they try to reach it rapidly. If many of them reach the resource in the same time and feed together, their energy level can be sufficient to become fertile. So, they reproduce and give birth to very efficient agents. This regrouping behavior, logic and coherent, is interesting due to its total emergence: nothing codes it in this application, neither code nor classifier reinforcement helps the appearance of this behavior.

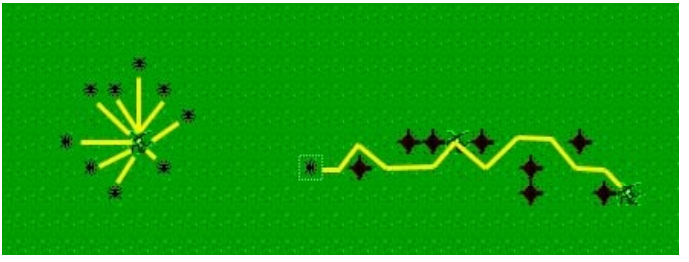


Fig. 5. Emerging behaviors.

All of these behaviors spontaneously emerge during most of simulations. Maybe other kinds of behaviors, less visible but also significantly, could emerge from this system.

4.4 Performance with Woods Model

To test the validity of the GA used in this application, simulations based on the Wilson's Woods environment, have been performed. The animat is modeled by a Ep1 agent, and only its learning and evolution capacities are kept. No macro-evolution occurs because it must stay Ep1 during all the time of a simulation. Fig. 6 shows the acquisition time of a resource according to the number of simulation turns. This result is based on the mean of ten trials.

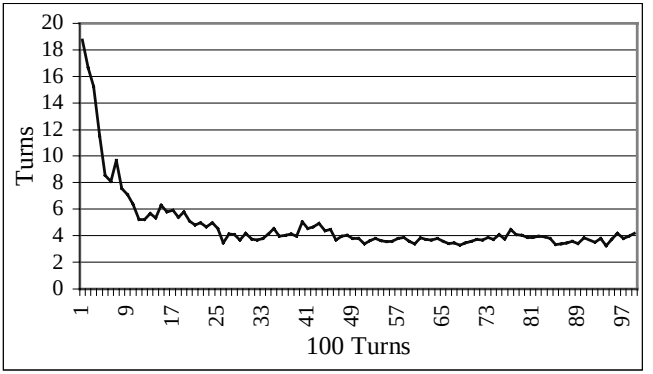


Fig. 6. Woods simulation.

In comparison to the results obtained by S. Wilson, the convergence to an efficient classifier base runs at the same speed. No gain is verified at short term. But, at medium and long term, performances of Ep1 agent are better than those found by the animat. After 8.000 turns, the animat moves during 4.5 steps to reach the resource, while the Ep1 agent reaches the resource in 3.6 steps after 5.000 turns.

If the macro-evolution is activated in these simulations, then the Ep1 agent can evolve to more complex classes. But, as in the extreme case described in 3.2, these niches are less adapted to this environment. In fact, the simple Ep1 agent is very adapted to the Woods world.

4.5 Hamilton's Kin Selection

W. Hamilton said in [15], that individuals can transmit copies of their own genotype not only by reproduction but also by helping related individuals like brothers, sisters and cousins. This *kin selection* theory explains the altruism phenomenon in eusocial species, particularly in the case of insects like ants, bees and wasps.

In this application, a simplified model of altruist/selfish behavior is developed: an altruist agent can give a part of its own energy[8] to another altruist with a weak energy level, while a selfish one never gives nothing to anyone. Fig. 7 shows the evolution of initial populations of Epo3Ipenergy agents, 50% of them are altruist and 50% are selfish. The reproduction is restricted to the agents of the same social type.

These results corroborate the idea that a cooperation and a mutual help improve the survival skills of a specie. From R. Dawkins point of view [8] the global goal of a creature being to transmit its genetic information to future generations, this method is particularly complementary to the classical reproduction. In most of simulations, altruist populations are more stable than selfish populations.

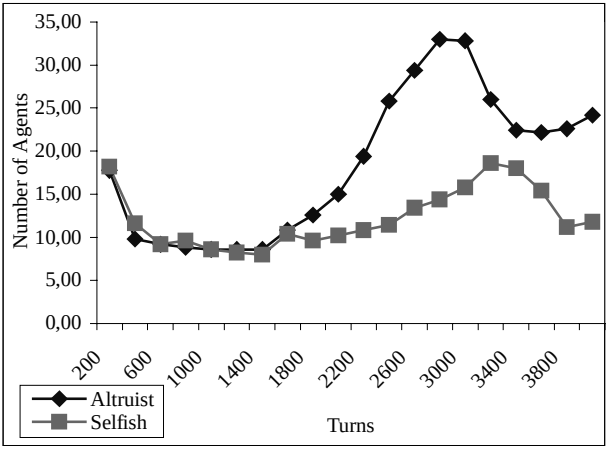


Fig. 7. Altruism vs. Selfishness.

5 Macro-evolution: From Natural to Artificial Life

Actually, biological theories can't explain some of the brutal changes occurring during the evolution of natural species. The theory of *punctuated equilibrium* [14] put forward the fact that jumps, caused by macro-mutations, would allow to species to modify deeply their genetic structure. This modification would appear after a long equilibrium of species to result in the creation of totally new ones without detectable transition. The work developed in [6] stipulates that these important overthrows can be determined and predicted by fractal equations [25]. Though these researches are not confirmed, they are particularly interesting because they mean that fractals and the evolution theory are closely linked. Effectively, according to the *self-organized criticality* theory, [1], some systems alter themselves spontaneously and violently, by crisis. So, many natural systems would maintain themselves at the edge of order and chaos, where fractal structures would emerge.

The study of phenomena existing at this edge is also specially studied in artificial life, [26]. Working on cellular automata [30], that are based on self-reproducing automata [28], C. Langton posed the *edge of chaos* hypothesis [19], which claims the existence of a phase transition [22], between ordered and chaotic behaviors, and locates complex dynamics in the vicinity of this transition. The classical *game of life*, originally designed by J.H. Conway [10], is a paradigmatic example of researches performed on the emergence of complex structures at this phase transition, [24].

[8] According to its energy level.

The huge modifications undergone by the phenotype during these jumps can also be the result of the alteration of particular genes, the homeogenes. These control genes have the property, under the effect of a mutation, to deeply modify the morphogenesis of a creature, as shown by the example of extra-eyes apparition in *Drosophila*, [11].

If the ETIC classes of agents are considered as control genes, then the macro-mutation operator can be seen as a kind of simplified model of the punctuated equilibrium theory and the homeogenes. The results found in the part 4.2 show that artificial species evolve brutally by successive class jumps to create relatively stable ecological niches according to environmental conditions. So, with an improved model, would a rule allowing to predict these evolution in an artificial system exist ? This question should be studied in further experiments.

6 Conclusion and Future Works

The aim of this work was to prove the adaptation of agents population using a Non-Homogeneous Classifier System combined with a macro-evolution process. Generally, agents adapt their behavior and their structure, in ETIC classes terms, to initial conditions of environment. In few cases, with the most extreme values, agents can't adapt and populations die. But with a large number of initial values, agents find the most efficient classifiers and the most adequate morphology. Tests show that creatures survive better with combination of sensors giving them several perception properties. Moreover, agents mutating and discovering communication, with a knowledge transmission, often invade populations. Finally, several behaviors emerge totally from the system without the intervention of a particular coding constraining the reinforcement of specific classifiers.

A NHCS is necessary to implement an efficient macro-evolution process. Indeed, previously, simulations have been performed without NHCS and classifiers contained all information due to ETIC classes. So an Ep1Epo4Edo7 agent possesses huge classifiers with non-separate parts Ep1 and Epo4 and Edo7. If a classifier has an inefficient part, then this classifier isn't efficient if either of the two other parts are correct. A NHCS allows to share the knowledge from one classifier to several classifiers. The inefficient part, transformed in a complete classifier, can be removed from the base. Moreover, the NHCS method allows a modular development of A-Life applications where each agent class can be implemented progressively.

Though several A-Life systems and Multi-Agents systems obtain similar results, NHCS is a new approach allowing co-evolution between classifiers.

Several directions can be followed for future works. First, with the large number of parameters included in this system, other simulations could de performed and new behaviors or new niches could appear. Secondly, this application also can be enriched with another kind of object: Tools. Agent using tools would improve their consump-

tion of resources and these tools would lead to the concept described in [31], cultural learning. But, the main goal is to develop an A-Life online system integrating NHCS in artificial creatures. This virtual laboratory would allow agents, like the critters of K. Sims, [27], to grow and evolve in real time 3D with strong interactions with humans. Agent behaviors could be learned either by imitation of humans or by a reinforcement/macro-evolution process.

The final step of this work, in a very long term, is its implementation on real artificial creatures like robots. But, as underlined by R. Brooks, [4], most of efficient algorithms used in virtual creatures doesn't run correctly for real robots. Several other parameters must be considered: Fuzzy information given by sensors, unpredictable effects performed by effectors, etc. The adaptation of NHCS should be studied deeply to stay efficient in real condition!

References

1. Bak, P. : How the Nature works : the science of self-organized criticality. Oxford University Press, Oxford (1997).
2. Bäck, T. and Schwefel, H.P. : An Overview of Evolutionnary Algorithms for Parameter Optimization. Evolutionnary Computation, Vol. 1. MIT Press (1993) 1-23.
3. Booth, G. : Gecko : A Continuous 2D World for Ecological Modelling. Artificial Life, Vol. 3:3. MIT Press (1997) 147-164.
4. Brooks, R. : Artificial Life and Real Robots. In : Varela, F. and Bourgine, P. (eds.) : Towards a practice of Autonomous Systems, Proceedings of the 1st European Conference on Artificial Life. MIT Press, Paris (1991).
5. Cariani, P. : Emergence and Artificial Life. In : Langton, C., Taylor, C., Farmer, J.D. and Rasmussen, S. (eds.) : Proceedings of Artificial Life II, SFI Studies in the Sciences of Complexity, Vol. 10. Addison-Wesley (1991) 775-797.
6. Chaline, J., Nottale, L. and Grou, P. : L'arbre de la vie a-t-il une structure fractale ? In : Comptes rendus de l'Académie des Sciences, Série IIa Sciences de la terre et des planètes, Vol. 328:11 (1999) 717-726.
7. Cliff, D. : AI and A-Life : Never Mind the Blocksworld. In : Cohn, A.G. (ed.) : Proceedings of the 11th European Conference on Artificial Intelligence. Wiley (1994) 799-804.
8. Dawkins, R. : Le Gène Egoïste. Armand Colin (1990).
9. Ferber, J. : Les Systèmes Multi-Agents : vers une Intelligence Collective. Inter-Editions (1995).
10. Gardner, M. : The fantastic combinations of John Conway's new Solitaire Game Life. Scientific American (1970).
11. Gehring, W.J. : The master control gene for morphogenesis and evolution of the eye. Genes to Cells, Vol. 1 (1996) 11-15.
12. Goldberg, D. : Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley (1989).

13. Goldberg, D. and Deb, K. : A Comparative Analysis of Selection Schemes used in Genetic Algorithms. In : Rawlins, G.J.E. (ed.) : Proceedings of the 1st Workshop on Foundations of Genetic Algorithms. Morgan Kaufmann (1991) 69-93.
14. Gould, S.G. : La vie est belle, Paris. Seuil (1991).
15. Hamilton, W.D. : The Genetical Evolution of Social Behavior : I and II. In : Williams, G. (ed.) : Group Selection. Aldine (1971) 23-43 and 44-89.
16. Hoffmeister, F. and Bäck, T. : Genetic Algorithms and Evolution Strategies : Similarities and differences. In : Schwefel, H.P. and Männer, R. (eds.) : Proceedings of the 1st Workshop on Parallel Problem Solving from Nature. Springer-Verlag (1991) 455-469.
17. Holland, J. : Adaptation in Natural and Artificial Systems. MIT Press (1975).
18. Langton, C. (ed.) : Artificial Life : Proceedings of the 1st Workshop on the Synthesis and the Simulation of Living Systems '1987. Addison-Wesley (1989).
19. Langton, C. : Life at the edge of chaos. In : Langton, C., Farmer, J.D. and Rasmussen, S. (eds.) : Artificial Life II, SFI studies in the Sciences of Complexity, Vol. 10. Addison-Wesley (1991).
20. Lattaud, C. : A Classification for the control of the evolution of adaptive agents. In : Proceedings of the 10th International Florida Artificial Intelligence Research Symposium. (1997) 449-454.
21. Lattaud, C. : A Macro-Mutation Operator in Genetic Algorithms. In : Prade, H. (ed.) : Proceedings of the 13th European Conference on Artificial Intelligence. Wiley (1998) 323-324.
22. Li, W., Packard, N. and Langton, C. : Transition phenomena in cellular automata rule space. Physica D, Vol. 45 (1990) 77-94.
23. Maes, P. : Modelling Adaptive Autonomous Agents. In Langton, C. (ed.) : Artificial Life : An Overview. MIT Press / Bradford Books (1995) 135-162.
24. Magnier, M., Lattaud, C. and Heudin, J.C. : Complexity classes in the two-dimensional life cellular automata subspace. Accepted in Complex Systems (1999).
25. Mandelbroot, B. : The fractal geometry of Nature. Freeman N.Y (1982).
26. Packard, N. : Adaptation at the edge of chaos. In : Kelso, S. and Shlesinger, M. (eds.) : Complexity in Biological Modeling (1998).
27. Sims, K. : Evolving 3D Morphology and Behavior by Competition. In : Brooks, R. and Maes, P. (eds.) : Artificial Life IV : Proceedings of the 4th International Conference on Artificial Life. MIT Press (1994) 28-39.
28. Von Neumann, J. : Theory of self-reproducing automata. In : Burks, A. (ed.), Princeton University Press (1966).
29. Wilson, S. : Knowledge Growth in an Artificial Animal. In : Grefenstette, J.J. (ed.) : Proceedings of the 1st International Conference on Genetic Algorithms and their Applications. (1985) 16-23.
30. Wolfram, S. : Universality and complexity in cellular automata, Physica D, Vol. 10 (1984) 1-35.
31. Zannoni, E. and Reynolds, R. : Learning to Control the Program Evolution Process with Cultural Algorithms. Evolutionnary Computation, Vol. 5:2 (1997) 181-211.

An Introduction to Anticipatory Classifier Systems

Wolfgang Stolzmann

Institute for Psychology, University of Wuerzburg, Germany
stolzmann@psychologie.uni-wuerzburg.de

Abstract. Anticipatory Classifier Systems (ACS) are classifier systems that learn by using the cognitive mechanism of anticipatory behavioral control which was introduced in cognitive psychology by Hoffmann [4]. They can learn in deterministic multi-step environments.[1] A stepwise introduction to ACS is given. We start with the basic algorithm and apply it in simple "woods" environments. It will be shown that this algorithm can only learn in a special kind of deterministic multi-step environments. Two extensions are discussed. The first one enables an ACS to learn in any deterministic multi-step environment. The second one allows an ACS to deal with a special kind of non-Markov state.

1 Introduction

An Anticipatory Classifier System (ACS) is a learning algorithm based on learning classifier systems and the psychological learning mechanism of "Anticipatory Behavioral Control". The knowledge of an ACS is represented by Condition-Action-Expectation rules (*C-A-E* classifiers). An action *A* is always accompanied by expectations *E* of its result *R*. By comparing the differences of the perceived result *R* after acting on an environment and its expectations *E*, an ACS builds such *C-A-E* classifiers. It always starts with the most general knowledge and learns by continuously specifying that knowledge. An ACS is capable of *latent learning*, i.e. learning in the absence of environmental reward and the *C-A-E* classifiers can be used to do *action-planning*. Latent learning and action-planning in ACS is discussed in Stolzmann & Butz [this volume, pp. 303-320].

This chapter gives a stepwise introduction to ACS. First an ACS and its basic learning algorithm will be introduced and applied in simple environments. Two types of environments are frequently used to study learning classifier systems, state environments [1,10] and "woods" environments [3,7,9,14,15]. A series of "woods" environments: MazeF1, MazeF2, MazeF3 and MazeF4 will be introduced to study ACS. Two extensions of the basic learning algorithm are necessary to enable an ACS to deal with all these "woods" environments.

[1] Butz, Goldberg & Stolzmann [2] show that ACS can also learn in deterministic single-step environments with a perceptual causality in its successive states.

P.L. Lanzi, W. Stolzmann, and S.W. Wilson (Eds.): LCS '99, LNAI 1813, pp.175-194, 2000.

2 Anticipatory Classifier Systems (ACS)

In this section a detailed description of ACS is given. Firstly, anticipatory behavioral control is briefly introduced. Secondly, the functionality of an ACS is explained. Finally, the basic learning mechanism of an ACS is presented.

2.1 Anticipatory Behavioral Control

The psychologist Tolman [12] first proposed the formation of SRE-units he called "field-expectancies" while forming a learning theory. The SRE-units represent the knowledge of an organism that in situation S the reaction R leads to the effect E. Nonetheless, Tolman did not state in detail how such units could be formed. Hoffmann [4] formulated a learning mechanism called anticipatory behavioral control (cf. figure 1) that explains how SRE-units could be formed:

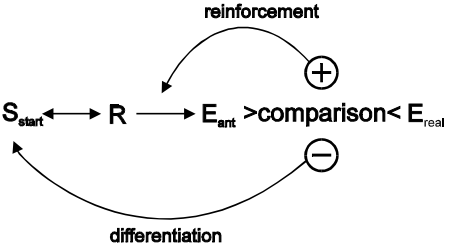


Fig. 1. Anticipatory behavioral control (according to Hoffmann [4, p. 44])

Four assumptions are made:
1. Each action R is combined with an anticipation of its (desired) effects E_{ant}.
2. The bonds between actions and their anticipated effects are reinforced.
3. If the reliability of action-effect relations is dependent on situational conditions S_{start}, then situational properties are differentiated.
4. Reinforcement and differentiation depend on a comparison of anticipated effects E_{ant} and actually occurring effects E_{real}.

It is not necessary to assume that environmental reward is given, i.e. anticipatory behavioral control is able to explain how an organism can learn latently.

2.2 Basic Elements of an ACS

An ACS is an agent that learns in a multi-step environment. A characteristic of a multi-step environment is that an action executed by the agent leads to a new environmental state which the agent has to deal with next. The time *t* is discrete.

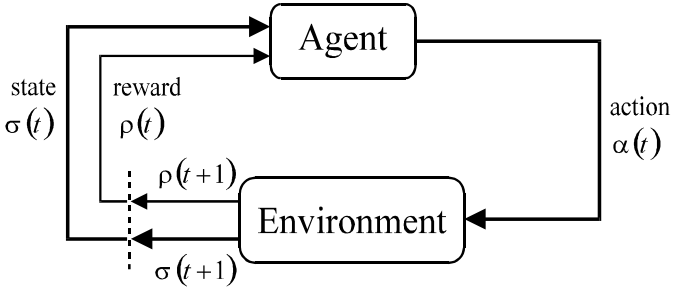


Fig. 2. The agent-environment interaction

Figure 2 shows the agent-environment interaction. At time *t* the agent perceives the state $\sigma(t)$ and gets an environmental reward of $\rho(t)$ ($\rho(t)$ can be zero). On the basis of $\sigma(t)$ and $\rho(t)$ the agent selects an action $\alpha(t)$. The execution of $\alpha(t)$ changes the environmental state and the environmental reward. At time *t*+1 $\sigma(t+1)$ and $\rho(t+1)$ are perceived by the agent and so on.

An ACS consists of the following four basic components:
1. an *input interface* with detectors to perceive environmental states σ ,
2. an *output interface* with effectors to execute actions α ,
3. a *classifier list* which contains simple production rules called *C*-*A*-*E* classifiers and
4. a *message list* which contains messages sent by the input interface and the classifier list.

Each detector of the input interface contains information about one attribute of the environmental state and usually delivers values 0 and 1. So a message, that was sent by the detectors, is the internal representation of the current environmental state. If the input interface consists of *n* detectors, then a message is usually defined as an element of $\{0,1\}^n$. Note that it is also possible to use other sets of attribute values than $\{0,1\}$.

A classifier consists of three basic parts:
1. a *condition part C* with conditions that contain information about the attributes of environmental states,
2. an *action part A* with instructions for the effectors and
3. an *expectation part E* which specifies the next anticipated state of the environment.

Section 4 introduces a fourth part (a *mark M*) that remembers the states in which the classifier did not work correctly.

A classifier is represented as a 3-tuple *C*-*A*-*E* with $C, E \in \{0,1,\#\}^n$ and *A* is a finite chain of simple actions. A '#'-symbol in the condition part *C*, which is called the "DON'T CARE" symbol, means that the ACS ignores the i_{th} detector. A '#'-symbol in the expectation part *E*, which is called the "PASS-THROUGH" symbol, means that the ACS believes that the i_{th} detector stays the same when *A* causes the output interface to act in the environment. A classifier processes two strength values *q* and *r*.

The *quality q* predicts the accuracy of its anticipation. The *reward-prediction r* predicts the amount of reward expected from the environment.

2.3 Behavioral Acts in an ACS

Figure 3 shows a basic schema of a behavioral act in an ACS.

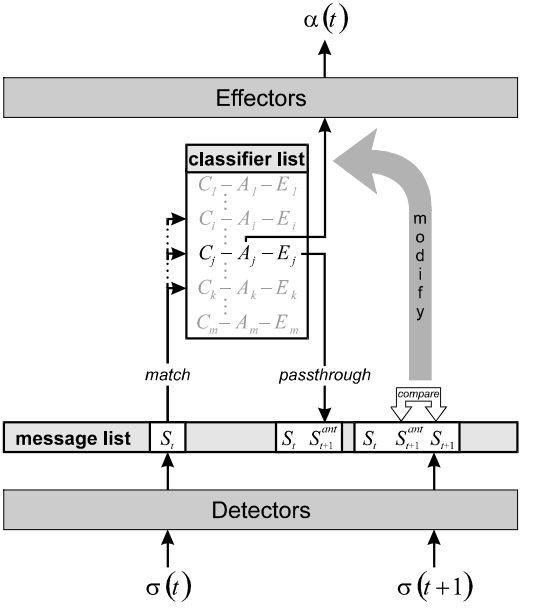


Fig. 3. A behavioral act at time *t* in an ACS

1. At time *t* the detectors perceive the environmental state $\sigma(t)$ and send the perception of $\sigma(t)$ called state S_t to the message list.
2. S_t is compared with the condition parts *C* of all classifiers. The matching classifiers form a match set.
3. One classifier of the match set is selected by roulette-wheel [6] or with an exploration probability p_x by random selection. During the roulette-wheel selection the probability for a classifier c_i to become active is $f_{c_i} \Big/ \sum_{c \in \text{match set}} f_c$. Here $f_c(t) = q_c(t) \cdot r_c(t)$ is the fitness of a classifier *c*. In figure 3 the classifier $c_j = C_j - A_j - E_j$ is selected.
4. The selected classifier c_j becomes active. With regard to S_t its expectation part E_j adds a new message $S_{t+1}^{ant} = passthrough(S_t, E_j)$ to the message list and its action part A_j causes the detectors to execute the external action $\alpha(t)$.

The function *passthrough* replaces the '#'-symbols of E_j by the corresponding components of S_t . As mentioned above a '#'-symbol in E_j means that the ACS believes that the i_{th} detector stays the same when $\alpha(t)$ is executed in the environment. So S_{t+1}^{ant} is the anticipated environmental state immediately after $\alpha(t)$ is executed.

5. After the execution of $\alpha(t)$ the detectors perceive $\sigma(t+1)$ and send S_{t+1} to the message list.
6. S_{t+1}^{ant} and S_{t+1} are compared. Depending on the result of this comparison, the classifier list is modified and anticipation-learning takes place (cf. section 2.4.1). The quality q_{c_j} of the classifier c_j is increased or decreased, or a new classifier is added to the classifier list.
7. S_t and S_{t+1}^{ant} are deleted and S_{t+1} starts a new behavioral act.

2.4 The Basic Learning Algorithm

There are two different concepts of learning in ACS: anticipation-learning and reward-learning. Anticipation-learning is the core of the implementation of anticipatory behavioral control. It is independent of reward-learning. Reward-learning can be any reinforcement learning technique. Here we use the bucket brigade algorithm [5] where the currently active classifier makes a payment to the previously active classifier. This method uses the Widrow-Hoff delta rule [13].

2.4.1 Anticipation-Learning

The basic idea of anticipation-learning is easy: If S_{t+1} was anticipated correctly, i.e. if S_{t+1} matches S_{t+1}^{ant}, then the quality q of the active classifier should be increased. If the anticipation was wrong, then a new classifier should be generated that anticipates S_{t+1} correctly. If it is not possible to generate such a classifier, then the quality q of the active classifier should be decreased. Analogous to the bucket brigade algorithm, a bid ratio $b_q \in [0,1]$ is used to update the quality q.

Four cases are distinguished in detail:

useless case:
If the behavioral act t does not change anything in the environment, then the quality of the active classifier c becomes smaller: $q_c(t+1) = (1-b_q) \cdot q_c(t)$

expected case:
If S_{t+1} matches S_{t+1}^{ant}, and if the behavioral act t changes the environment, then the quality of the active classifier c becomes larger: $q_c(t+1) = (1-b_q) \cdot q_c(t) + b_q$.

correctable case:

If S_{t+1} does not match S_{t+1}^{ant}, and if the behavioral act *t* changes the environment, and if all components of *C* and *E* (of the active classifier *c* = *C*-*A*-*E*) for which S_{t+1} does not match S_{t+1}^{ant} are '#'-symbols, then a new classifier $c_{new} = C_{new} - A_{new} - E_{new}$ with $A_{new} = A$ is produced. C_{new} and E_{new} differ from *C* and *E* in the non-matching components of S_{t+1}^{ant} and S_{t+1}. In these components E_{new} is equal to S_{t+1} and C_{new} is equal to S_t. Thus, C_{new} still matches S_t and E_{new} now anticipates S_{t+1} correctly.

not correctable case:

If S_{t+1} does not match S_{t+1}^{ant}, and if it is not possible to correct the classifier, then the quality of the active classifier becomes smaller just as in the useless case.

It can be shown that $q_c(t) \in [0,1]$ for all *t*, if $q_c(0) \in [0,1]$. $q_c(t)$ can be regarded as the probability that S_{t+1}^{ant} matches S_{t+1}, i.e. $q_c(t)$ describes the quality of the correctness of the anticipated consequences.

Till now we have an algorithm that is able to produce new classifiers but it is not possible to delete classifiers. That is why we define a threshold $\theta_i \in [0,1]$. If the quality q_c of a classifier *c* becomes lower than θ_i, then *c* is *inadequate* and is deleted. A typical value for θ_i is 0.1.

Another threshold is $\theta_r \in [0,1]$. A typical value for θ_r is 0.9. If $q_c \geq \theta_r$ then *c* is called *reliable*, i.e. the probability that *c* anticipates the next state correctly is very high.

2.4.2 Reward-Learning

Anticipation-learning enables an ACS to learn without environmental rewards. However, if environmental rewards are given, then an ACS can be treated like a usual classifier system that uses a reinforcement learning technique. In this paper the bucket brigade algorithm [5] is used to update the reward-predictions *r* of the classifiers. Analogous to anticipation-learning, a bid ratio $b_r \in [0,1]$ is used.

Let c_t be the active classifier at time *t* and c_{t+1} the active classifier at time $t+1$.

If $\rho(t+1) \neq 0$, then $r_{c_t}(t+1) = (1-b_r) \cdot r_{c_t}(t) + b_r \cdot \rho(t+1)$.

If $\rho(t+1) = 0$, then $r_{c_t}(t+1) = (1-b_r) \cdot r_{c_t}(t) + b_r \cdot r_{c_{t+1}}(t)$.

That means, if there is no environmental reward at time $t+1$, then the currently active classifier c_{t+1} gives a payment of $b_r \cdot r_{c_{t+1}}(t)$ to the previous active classifier c_t. If there is environmental reward $\rho(t+1)$, then $b_r \cdot \rho(t+1)$ is given to the previous active classifier c_t.

3 Simple Applications of ACS

In this section an ACS is applied in two simple "woods" environments, on the one hand to show how it works and on the other hand to show its limitations. First, the "woods" environments are introduced.

3.1 Woods Environments

"Woods" environments are 2-dimentional grids. Each cell can contain an obstacle T (tree), a goal F (food), or can be empty. The agent's perception is restricted to the eight nearest cells. The agent detects these cells starting north and coding clockwise. The detector values are "t" (tree), "f" (food) and "b" (blank) for an empty cell. The agent can use the actions N, NE, E, SE, S, SW, W and NW to reach one of these cells. If the agent tries to reach an obstacle T, then the action fails and the agent does not move. In this paper the actions are encoded by ↑, ↗, →, ↘, ↓, ↙, ← and ↖. Usually the agent's task is to learn the shortest path to goal states. In this paper a further task is to learn an internal model of the environment. A learning experiment in a "woods" environment consists of several trials. A trial starts by randomly placing the agent into an empty cell of the environment. If the agent reaches a goal state or if a maximum number of steps (MaxSteps) is reached, then a new trial starts. MaxSteps is used to avoid infinite loops. Figure 4 shows a series of four new "woods" environments that will be used in this paper.

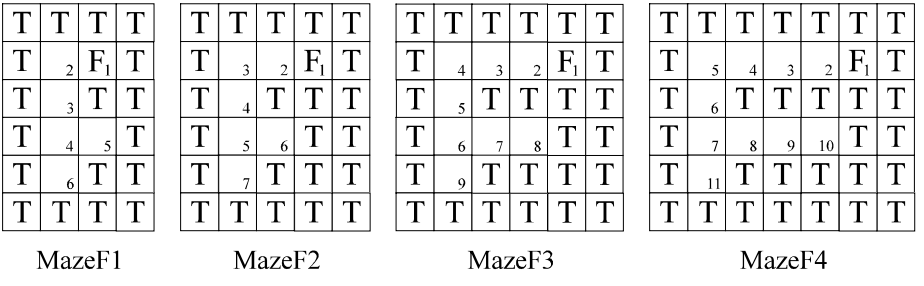


Fig. 4. "Woods" environments

The small numbers indicate the environmental states, for example in MazeF1 $\sigma(t) \in \{1,\ldots,6\}$.

If the agent's task is to reach the goal state F, then an optimal performance can be defined. For each environment the optimal performance is the average minimal number of steps to food. In MazeF1 this is (1+1+2+2+3)/5=1.8.

Table 1. Optimal performances

environment	MazeF1	MazeF2	MazeF3	MazeF4
optimal performance	9/5=1.8	15/6=2.5	27/8=3.375	45/10=4.5

All these environments are deterministic, i.e. $\sigma(t+1)$ is determined by $\sigma(t)$ and $\alpha(t)$, but from the agent's perspective MazeF4 is not deterministic because the environmental states 3 and 9 are not distinguishable for the agent. So, sometimes the action → leads to state 2 and sometimes to state 10. This perceptual aliasing problem is discussed in section 5.

In each environment an ACS has two tasks: first, to learn the shortest path to the goal state F and second, to learn an internal model of the environment. The knowledge of an ACS is represented by the classifiers, but only the reliable classifiers form the internal model. At the end of each trial the "achieved knowledge" and the "number of steps to food" are calculated.

To calculate the "achieved knowledge" it is tested whether or not there is a reliable classifier for each transition $(\sigma_1, \alpha, \sigma_2)$ with $\sigma_1 \neq \sigma_2$ that can be applied in σ_1 and anticipates σ_2 correctly.

As usual, the "number of steps to food" is the average number of steps to the goal state F during the past 50 trials (e.g. Lanzi & Wilson [9]). Therefore the curves for the "number of steps to food" contain no values for "Number of trials" < 50.

Following Lanzi & Wilson [9] a learning experiment is divided in two parts. After a certain number of trials exploration stops. A parameter end_explor is used.

If "Number of trials" ≤ end_explor, then roulette-wheel selection and random selection is applied (cf. section 2.3)

If "Number of trials" > end_explor, then the fittest classifier of the match set always becomes active.

3.2 An ACS in MazeF1 and MazeF2

First, an ACS is applied in MazeF1. Figures 5 and 6 show the results of several learning experiments with a reward $\rho > 0$ in the goal state F. Figure 5 shows the results for different rewards and figure 6 the results for different bid ratios b_r. The left curves show that in each learning experiment about 80% of the knowledge is learned. The ACS does not learn 100% because transitions that lead away from the goal state F are executed seldom as the reward-prediction *r* overrules the quality *q* of a classifier when doing roulette-wheel selection. The right curves show that in each learning experiment the number of steps converge to a value that is smaller than 3, but only the solid line in figure 6 (right) reaches the optimal performance.

If the Number of Steps to Food converges to a value ≥ 2, then the ACS's policy is not optimal: An ACS with a policy that is defined by the transitions 2→1, 3↗1, 4→5, 5↖3, 6↗5 needs an average Number of Steps to Food of 2 and in state 4 the policy is not optimal.

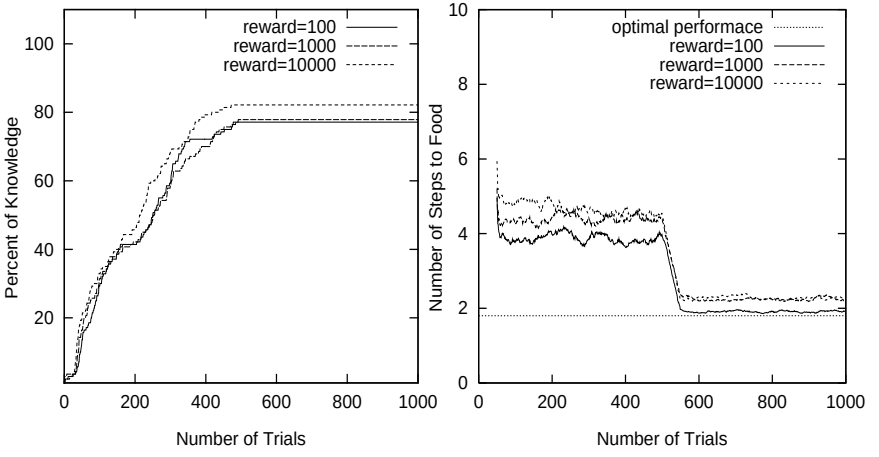


Fig. 5. Average of 10 learning experiments in MazeF1: $b_q = 0.05$, $b_r = 0.1$, $p_x = 0.5$, $\theta_r = 0.9$, $\theta_i = 0.1$, MaxSteps = 100, end_explor = 500

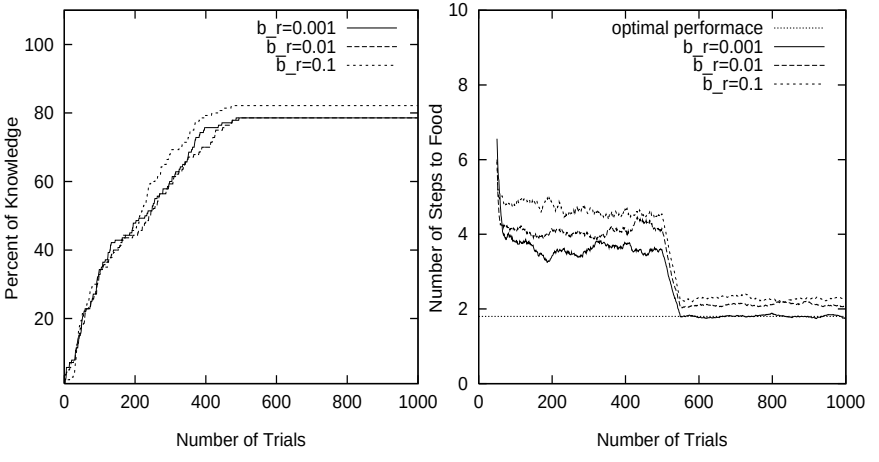


Fig. 6. Average of 10 learning experiments in MazeF1: $\rho = 10000$ in state F, $b_q = 0.05$, $p_x = 0.5$, $\theta_r = 0.9$, $\theta_i = 0.1$, MaxSteps = 100, end_explor = 500

Figures 7 and 8 show the results of several learning experiments with a reward $\rho = 0$ in the goal state F. The results depend only on anticipation-learning because $b_r = 0.0$, i.e. the reward-predictions *r* are not changed for any classifiers.

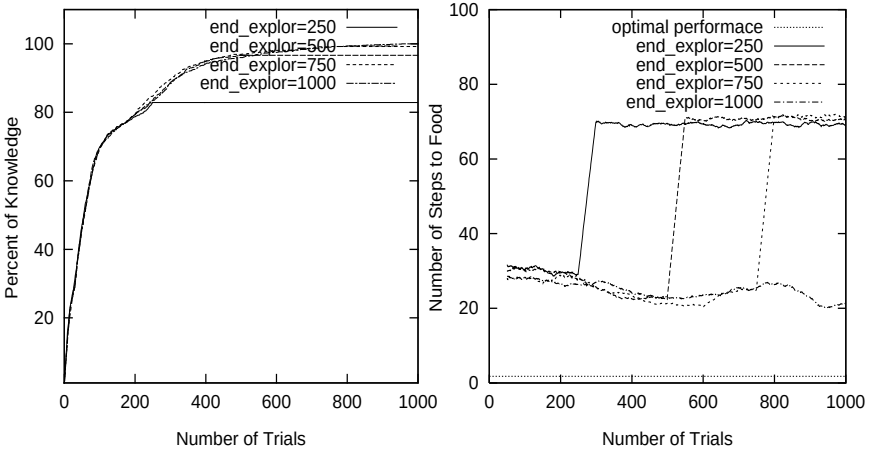


Fig. 7. Average of 100 learning experiments in MazeF1: $\rho = 0$ in state F, $b_q = 0.05$, $b_r = 0.0$, $p_x = 0.5$, $\theta_r = 0.9$, $\theta_i = 0.1$, MaxSteps = 100

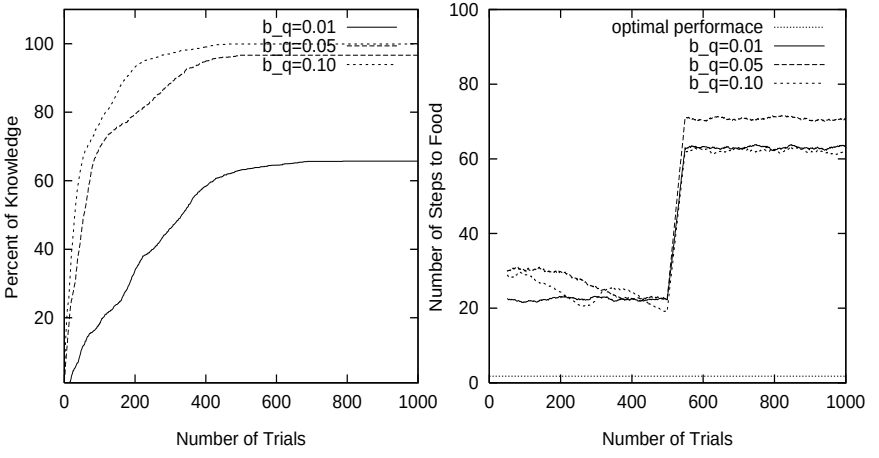


Fig. 8. Average of 100 learning experiments in MazeF1: $\rho = 0$ in state F, $b_r = 0.0$, $p_x = 0.5$, $\theta_r = 0.9$, $\theta_i = 0.1$, MaxSteps = 100, end_explor = 500

The left curves show that 100% of knowledge can be learned. On the one hand the results depend on end_explor (cf. figure 7 and table 2) and on the other hand they depend on b_q (cf. figure 8 and table 3). In each case the Number of Steps to Food does not converge because there is no reward in state F.

Table 2. Values for Number of Trials = 1000 of the curves shown in figure 7 (left)

end_explor	250	500	750	1000
knowledge(1000)	82.86%	96.64%	99.21%	100.00%

Table 3. Values for Number of Trials = 1000 of the curves shown in figure 8 (left)

b_q	0.01	0.05	0.10
knowledge(1000)	65.71%	96.64%	99.93%

Table 4. A typical final classifier list: $\rho = 10000$ in state F, $b_q = 0.05$, $b_r = 0.001$, $p_x = 0.5$, $\theta_r = 0.9$, $\theta_i = 0.1$, MaxSteps = 100, end_explor = 500

C	*A*	*E*	*r*	*q*	applicable in state
ttft####	↓	bftb####	24.674212	0.768354	2
bf#bttt	↘	tt#ttbbb	2.035896	0.942009	3
t#t#tbbb	←	b#b#bttt	1.991718	0.874828	5
#ftb####	↓	#tbt####	2.996855	0.947663	3
tt#ttbbb	↖	bf#bttt	597.179251	0.999998	5
bb###ttt	↗	tt###bbb	1.184156	0.932362	6
tt###bbb	↙	bb###ttt	0.529207	0.979263	5
b#b#bttt	→	t#t#tbbb	1.287998	0.898047	4
#tbt####	↑	#ftb####	961.940457	1.000000	4
#bt#t###	↑	#tb#b###	57.844570	1.000000	6
#tb#b###	↓	#bt#t###	0.541348	0.991742	4
bf#bbtt#	↗	tt#ttbb#	5376.512296	1.000000	3
##f#btt#	→	##t#tbb#	2048.023490	0.999996	2
bftb####	↑	ttft####	21.549042	0.961528	3
########	↑	########	0.500000	0.100000	2,3,4,5,6
########	→	########	0.500000	0.100000	2,3,4,5,6
########	↓	########	0.500000	0.100000	2,3,4,5,6
########	←	########	0.500000	0.100000	2,3,4,5,6
########	↗	########	0.500000	0.100000	2,3,4,5,6
########	↘	########	0.500000	0.100000	2,3,4,5,6
########	↙	########	0.500000	0.100000	2,3,4,5,6
########	↖	########	0.500000	0.100000	2,3,4,5,6

Table 4 shows a typical classifier list of an ACS that finished the learning experiment shown in figure 6, solid line. The last eight classifiers are the most general classifiers the ACS started with. The most general classifiers are never deleted although $q \le \theta_i$, so that the match set is never empty. An ACS with the classifier list shown in table 4 has an optimal policy to reach the goal state F in MazeF1.

An ACS with the same parameters as in table 4 was applied in MazeF2. At the end of the learning experiment one of 25 classifiers was

c = ##b#btt# - → - ##f#tbb#, r_c = 7782.220612, q_c = 0.446805.

This classifier *c* is not reliable because it can be applied in state 5 as well as in state 3 (cf. figure 4). If it is applied in state 3, then it reaches the expected case and q_c is increased. However, if it is applied in state 5, then it reaches the not correctable case and q_c is decreased. This is why the dashed curves in figure 9 (section 4.2) do not reach the optimum. In section 4 this problem is analyzed and solved.

4 A First Extension of ACS

In the so far introduced version of ACS, new classifiers are produced only if the *correctable case* is applied during a behavioral act. A specification of changing components takes place. This means that only those components are specified whose corresponding sensors of the input-interface change their values during the behavioral act. Therefore, it is not possible for an ACS to solve a task where at least one behavioral act, whose behavioral consequences depend on an environmental attribute that is not changed by the action, plays an important role.

Now an extension of ACS is introduced that enables ACS to solve such tasks. This extension is called *Specification of Unchanging Components.*

4.1 Specification of Unchanging Components

First, the algorithm is introduced and second, an example is given.

Let S_1 be the current state and $c = C - A - E$ the active classifier. If the application of *c* does not lead to the *expected case*, then *c* remembers S_1. Let S_2 be a later state that leads to a behavioral act where *c* is applied and leads to the *expected case*. Then, a component is randomly selected out of all components with the following property: S_1 is different from S_2 in this component and *C* and *E* consist of a #-symbol in this component. If such a component *i* exists then the *i*-th component of *C* and *E* is respectively replaced by the *i*-th component of S_2.

A parameter that defines a maximum number of components with specifications of unchanging components can be used. This parameter $u_{\max}$ is called *maximum number of unchangeable components*. It is also possible to use a parameter $c_{\max}$ that defines a maximum number of changeable components and controls the specifications during the *correctable case*. $c_{\max}$ is not used in this paper.

The following example will show how *specification of unchanging components* can be used. Let's consider the classifier *c* = ##b#btt# - → - ###f#tbb# once again that is produced by an ACS in MazeF2. If *c* is active in $\sigma = 5$, i.e. *S* = btbtbttt, then the *not correctable case* is applied and *c* gets a so called mark. This mark remembers *S*, i.e. *c* = ##b#btt# - → - ###f#tbb# -> btbtbttt now. If *c* becomes active in $\sigma = 3$ later on, i.e. *S* = ttbtbttt, then the *expected case* is applied and *S* is compared with the mark. The only different component is the first one. Therefore, a new classifier c_{new} is produced that is specified in the first component: c_{new} = t#b#btt# - → - t#f#tbb#.

c_{new} always reaches the expected case and will become reliable because it is applicable only in state $\sigma = 3$.

4.2 An ACS in MazeF2

Figure 9 shows the results of two learning experiments of an ACS in MazeF2. Environmental reward is given in the goal state F. The dashed curves show the results for an ACS that does not use the specification of unchanging components ($u_{\max} = 0$). The optimal performance is not reached. The solid curves show the results for an ACS that uses the specification of unchanging components. $u_{\max} \geq n$ causes no limitations at all (*n* is the number of detectors). In this case the optimal performance is reached.

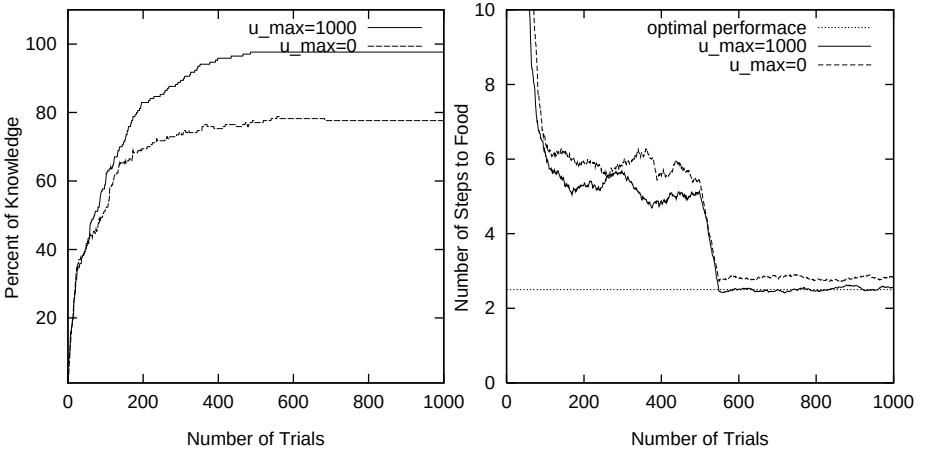


Fig. 9. Average of 10 learning experiments in MazeF2: $\rho = 10000$ in state F, $b_q = 0.05$, $b_r = 0.001$, $p_x = 0.5$, $\theta_r = 0.9$, $\theta_i = 0.1$, MaxSteps = 100, end_explor = 500

Figure 10 shows the results of two analogous learning experiments without environmental reward in the goal state F. As expected, the number of steps to food do not become smaller but an internal model is learned.

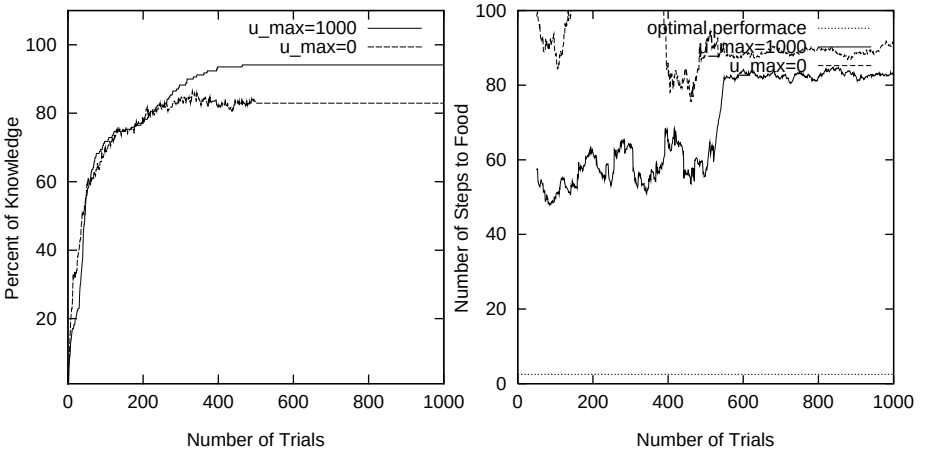


Fig. 10. Average of 10 learning experiments in MazeF2: $\rho = 0$ in state F, $b_q = 0.05$, $b_r = 0.0$, $p_x = 0.5$, $\theta_r = 0.9$, $\theta_i = 0.1$, MaxSteps = 100, end_explor = 500

4.3 Generalized Marks

Let us consider an ACS in MazeF3 to show that the marks of a classifier can be generalized to improve the *specification of unchanging components*. An ACS that did not use specification of unchanging components (same parameters as in table 4) produced c = ####btt# - → - ####tbb#, r_c = 3.524158, q_c = 0.251021. c can be applied in the environmental states $\sigma_1 = 4$, S_1 = ttbtbttt, $\sigma_2 = 5$, S_2 = bbtbbttt and $\sigma_3 = 6$, S_3 = btbtbttt, but only in S_1 the expected case is reached. Let us assume that the classifier c was applied in σ_2, i.e. it contains the mark S_2. When c is applied in σ_3, then it cannot get the mark S_3 because it is already marked. Therefore generalized marks are introduced now.

Each component of a mark can contain a set of detector values. If a classifier already contains a mark $M = (m_1,\ldots,m_m)$ and shall be marked with a further state $S = (s_1,\ldots,s_m)$, then each component of M is updated by $m_i \leftarrow m_i \cup \{s_i\}$.

In our example the classifier c first is marked with S_2 and gets the mark $M = (\{b\},\{b\},\{t\},\{b\},\{b\},\{t\},\{t\},\{t\})$. Then it is marked with S_3 and the mark becomes $M = (\{b\},\{b,t\},\{t,b\},\{b,t\},\{b\},\{t\},\{t\},\{t\})$. When c is applied in S_1 now it reaches the *expected case* and specification of unchanging components takes place.

Therefore, it is necessary to define what it means that a component of S_1 is different from the mark M:

A component s of a state S is different from a component m of a mark M, if $s \notin m$.

In our example only the first component of S_1 is different from M and the classifier is specified in the first component of its condition- and expectation-part.

4.4 An ACS in MazeF3

Figures 11 and 12 show the results of two learning experiments of an ACS in MazeF3. On the one hand specification of unchanging components with generalized marks is used (solid lines) and on the other hand it is not used (dashed lines).

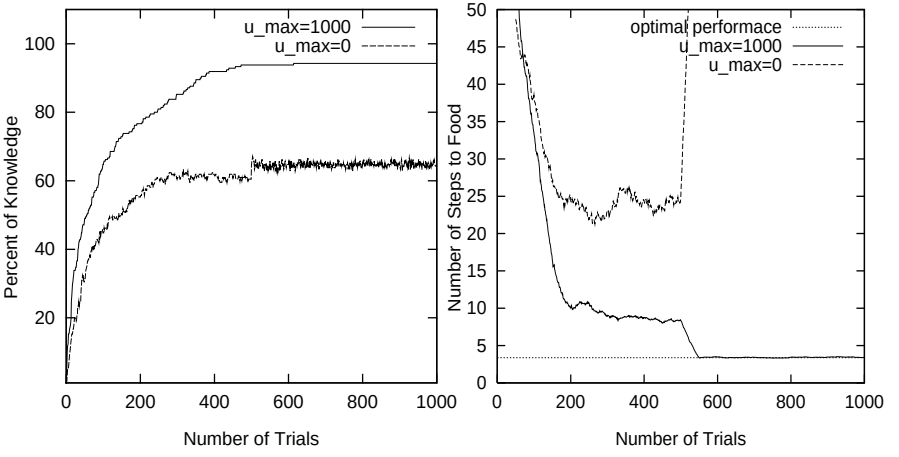


Fig. 11. Average of 10 learning experiments in MazeF3: parameters as in figure 9

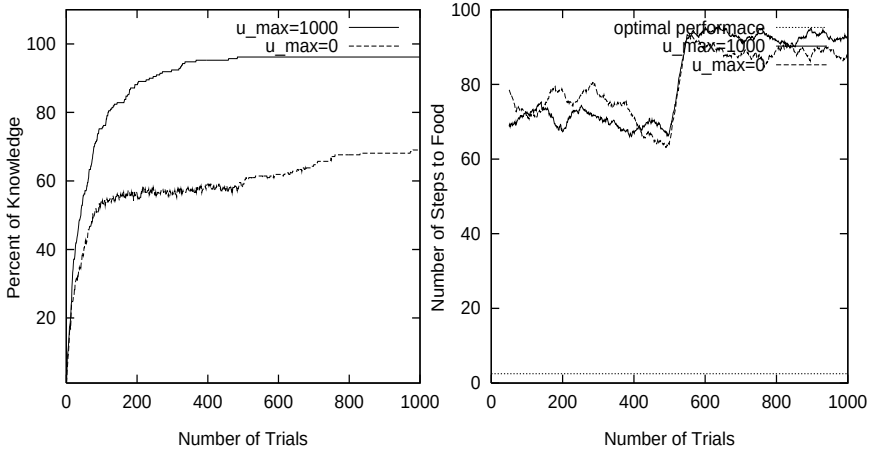


Fig. 12. Average of 10 learning experiments in MazeF3: parameters as in figure 10

5 A Second Extension of ACS

In this section an extension of ACS is discussed that enables an ACS to deal with a special kind of non-Markov states, i.e. a very special kind of non-deterministic environment.

5.1 The Perceptual Aliasing Problem in Woods Environments

As mentioned in section 3.1, MazeF4 is not deterministic from the agent's perspective because the environmental states 3 and 9 are not distinguishable. This problem is called the *perceptual aliasing problem* and non-distinguishable but different states are called *aliasing states.*

A state S_t is an aliasing state, if there is a sequence $S_0,\alpha(0),\ldots,S_t,\alpha(t)$ with $p\{S_{t+1}|\alpha(t),S_t,\ldots,\alpha(0),S_0\}=1$ and $p\{S_{t+1}|\alpha(t),S_t\}\neq p\{S_{t+1}|\alpha(t),S_t,\ldots,\alpha(0),S_0\}$.[2]

An aliasing state is a special kind of non-Markov state because in general $p\{S_{t+1}|\alpha(t),S_t,\ldots,\alpha(0),S_0\}$ can be different from 1 in non-Markov environments. In Woods environments there is always a sequence $S_0,\alpha(0),\ldots,S_t,\alpha(t)$ with $p\{S_{t+1}|\alpha(t),S_t,\ldots,\alpha(0),S_0\}=1$, except for a new approach by Lanzi & Colombetti [8].

Cliff & Ross [3] introduced memory to ZCS and Lanzi & Wilson [9] introduced memory to XCS in order to deal with aliasing states.[3] In this section another approach called action chunking is introduced to avoid aliasing states.

Before an ACS can deal with aliasing states, it needs a criterion to recognize an aliasing state:

Let S_2 be the current state and the classifier $c_2 = C_2 - A_2 - E_2$ be the winner of the competition and let c_2 reach the *expected case*. If c_2 is marked and if the *specification of unchanging components* fails then the ACS assumes that S_2 is an aliasing state.

5.2 Action Chunking in ACS

An ACS builds *action chunks* to avoid aliasing states. This is done as follows:

Let $c_1 = C_1 - A_1 - E_1$ be the classifier that was active just before c_2 and led to the expected case, i.e. $S_2^{ant} = S_2$. (c_1 can also be a classifier that was produced by *specification of unchanging components* or in the *correctable case* just before c_2 became active.) The classifiers c_1 and c_2 can be used to create a new classifier which avoids the aliasing state S_2. This new classifier $c_{new} = C_{new} - A_{new} - E_{new}$ consists of

$C_{new} = passthrough(C_2, C_1)$, $E_{new} = passthrough(E_1, E_2)$ and $A_{new} = A_1 A_2$.

[2] *p{A|B}* is the conditional probability of *A* given *B*.

[3] ZCS and XCS are well known learning classifier systems.

A_1A_2 means that the output-interface first executes A_1 and then A_2 and is called action chunk of A_1 and A_2.

The following transformation explains the definition of E_{new}:

$$\begin{aligned} S_3^{ant} &= passthrough\left(S_2^{ant}, E_2\right) \\ &= passthrough\left(passthrough\left(S_1, E_1\right), E_2\right) \\ &= passthrough\left(S_1, passthrough\left(E_1, E_2\right)\right) \\ &= passthrough\left(S_1, E_{new}\right) \end{aligned}$$

because *passthrough* is associative. Both components of *passthrough* can contain '#'-symbols.

Note: The definition of C_{new} is correct. If you define $C_{new} = passthrough\left(C_1, C_2\right)$, then there is no guarantee that the new classifier matches S_1 (S_1 is the state just before S_2). An alternative is to define $C_{new} = C_1$, but it is not used here.

It is possible that an action chunk consists of more than two actions. So we define a maximum number of simple actions in an action part $a_{\max}$.

5.3 Controlled Action Chunking

If an ACS uses action chunks, then it can produce classifiers with useless action chunks, for example a classifier with an action chunk ↑↓→ first moves north and then moves south with the result of being in the same state as before. A further algorithm is introduced that solves this problem.

Controlled execution of a classifier *c* with an action chunk $a_1,\ldots,a_k$ using a message list *M*:

$M := \{m_0\}$ with $m_0 := S_t$

For $i = 1,\ldots,k$:

  Execute a_i and perceive its result m_i.

  If $m_i \in M$ then

    decrease the quality q_c of *c* and stop

  else

    $M := M \cup \{m_i\}$.

Compare $m_k = S_{t+1}$ with S_{t+1}^{ant} and calculate the learning algorithm for behavioral acts.

5.4 An ACS in MazeF4

Figure 13 shows the results of an application of an ACS in MazeF4.

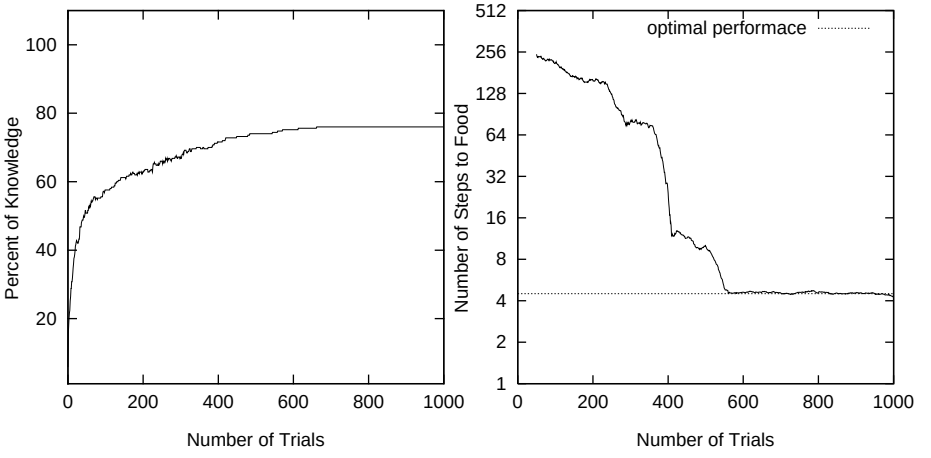


Fig. 13. Average of 10 learning experiments in MazeF4: $\rho = 10000$ in state F, $b_q = 0.05$, $b_r = 0.001$, $p_x = 0.5$, $\theta_r = 0.9$, $\theta_i = 0.1$, $u_{\max} = 1000$, MaxSteps = 1000, $a_{\max} = 2$, end_explor = 500

If action chunking is not used and an ACS is applied in MazeF4, then the ACS is not able to learn a good policy to reach the goal state F. It always learns a policy that contains an infinite loop. Let us assume that the policy is optimal for all states except of the aliasing states 3 and 9. The policy in the aliasing states can either be → or ←. If it is → and a trial starts in state 9 or 10, then the ACS alternates between state 9 and state 10. If it is ← and a trial starts in one of the states 3, 4, 5, 6, 7, 8 or 9, then it alternates between states 3 and 4.

6 Summary and Conclusions

Learning in an ACS can be summarized as follows: First, an ACS learns action-effect relations, i.e. it learns which environmental changes occur reliably when an action is executed. This is done in the *correctable case* (cf. section 2.4.1). An ACS with this basic learning algorithm only learns in MazeF1. Second, if action-effect relations are not sufficient for successful behavior like in MazeF2 and MazeF3, then additional situational properties are taken into account. Situation dependent action-effect relations are being learned. This is done by *specification of unchanging components* (cf. section 4.1). Third, an ACS can form action chunks to avoid aliasing states. In section 5 is shown that an ACS that uses action chunking can learn in MazeF4. Figure 13 shows that an ACS can learn very quickly in MazeF4. Future research will have to show whether this result is specific to MazeF4 or whether an ACS is able to learn quickly in any woods environment with aliasing states, for example in Woods101, Maze7, Woods$101\frac{1}{2}$, Woods102 [9] or Maze10 [8].

Up to now a very important assumption for an ACS is that its environment is almost deterministic. What happens if an ACS is applied in several kinds of probabilistic environments is discussed in [2].

In this chapter ACS were applied in simple "woods" environments. Applications of ACS in learning robots can be found in Stolzmann & Butz [this volume, pp. 303-320]

Notations

Anticipatory behavioral control:

S	Situation, Stimulus
R	(Re)action, Response
E	Effect

Environment:

t	time
$\sigma(t)$	state (external)
$\alpha(t)$	action (external)
$\rho(t)$	reward (external)

Anticipatory Classifier System:

n	Number of detectors
S	state (intern)
C-A-E	Classifier
C	Condition part of a classifier
A	Action part of a classifier
E	Expectation part of a classifier
M	Mark
q	quality
r	reward-prediction
b_q, b_r	bid ratios for updating q and r
p_x	exploration probability
f	fitness ($f = q \cdot r$)
$u_{\max}$	max. no. of unchangeable components
$c_{\max}$	max. no. of changeable components
$a_{\max}$	max. no. of simple actions in an A-part
$s_{\max}$	max. no. of steps (=behavioral acts) during action-planning (used in Stolzmann & Butz [this volume, pp. 303-320
θ_r	threshold for the reliability of a classifier c ($q_c \geq \theta_r \Rightarrow c$ is reliable)
θ_i	threshold for the inadequacy of a classifier c ($q_c \leq \theta_i \Rightarrow c$ is inadequate)

Woods environment:

MaxSteps	maximum number of steps to reach the goal state
end_explor	the end of exploration during a learning experiment

References

[1] Barry, Alwyn (1999). Aliasing in XCS and the Consequtive State Problem. In Banzaf, W. et al. (editors). *Proceedings of the Genetic and Evolutionary Computation Conference GECCO 99, July 13-17,1999 Orlando, Florida. Volume 1.* San Francisco, CA: Morgan Kaufmann. 19-34.

[2] Butz, M., Goldberg, D., & Stolzmann, W. (1999). New Challenges for an Anticipatory Classifier System: Some Hard Problems and Possible Solutions. IlliGAL Report No. 99019. Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign

[3] Cliff, Dave and Ross, Susi (1995). Adding Temporary Memory to ZCS. *Adaptive Behavior* Vol.3, No. 2, 101-150.

[4] Hoffmann, Joachim (1993). *Vorhersage und Erkenntnis*. Göttingen: Hogrefe.

[5] Holland, J. H. (1985). Properties of the bucket brigade algorithm. In John J. Grefenstette, editor. *Proceedings of the 1st International Conference on Genetic Algorithms and their Applications (ICGA85)*. Lawrence Erlbaum Associates: Pittsburgh, PA, July 1985. 1-7.

[6] Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*. Reading, Massachusetts: Addison-Wesley.

[7] Lanzi, Pier Luca (1998). An Analysis of the Memory Mechanism of XCSM. In Koza, John R. et al. (editors). Genetic Programming 1998: *Proceedings of the Third Annual Conference, July 22-25, 1998, University of Wisconsin, Madison, Wisconsin.* San Francisco, CA: Morgan Kaufmann. 643-651.

[8] Lanzi, P. L., & Colombetti, M. (1999). An Extension to the XCS Classifier System for Stochastic Environments. In Banzaf, W. et al. (editors). *Proceedings of the Genetic and Evolutionary Computation Conference GECCO 99, July 13-17,1999 Orlando, Florida. Volume 1.* San Francisco, CA: Morgan Kaufmann. 353-360.

[9] Lanzi, P. L., & Wilson, S. W. (1999). Optimal Classifier System Performance in Non-Markov Environments. Technical Report N. 99.36, Politecnico di Milano (submitted to Evolutionary Computation).

[10] Smith, R. E. (1994). Memory exploitation in learning classifier systems. *Evolutionary Computation* 2(3). 199-220.

[11] Stolzmann, Wolfgang (1998). Anticipatory Classifier Systems. In Koza, John R. et al. (editors). Genetic Programming 1998: *Proceedings of the Third Annual Conference, July 22-25, 1998, University of Wisconsin, Madison, Wisconsin.* San Francisco, CA: Morgan Kaufmann. 658-664.

[12] Tolman, Edward C. (1932). *Purposive behavior in animals and men*. New York: Appleton.

[13] Widrow, B., & Hoff, M. (1960). Adaptive switching circuits. *Western Electronic Show and Convention*, 4. 96-104.

[14] Wilson, S. W. (1985). Knowledge growth in an artificial animat. In L.E. Associates (Ed.). *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*. 16-23

[15] Wilson, S. W. (1994). ZCS: a zeroth level classifier system. *Evolutionary Computation* 1(2). 1-18

A Corporate XCS

Andy Tomlinson and Larry Bull

Intelligent Computer Systems Centre
University of the West of England
Bristol BS16 1QY, U.K.
{Andy.Tomlinson, Larry Bull}@uwe.ac.uk

Abstract. Previously we have applied rule linkage to ZCS and shown that the resultant system demonstrates performance improvements over ZCS in a series of sequential tasks, particularly tasks which present ambiguous stimuli to the system. In this paper we show that similar benefits can be gained by applying rule linkage to the more complex XCS. We then show that the benefits of rule-linkage can be increased by further XCS specific modifications to the system's rule-linkage mechanisms.

1 Introduction

Wilson and Goldberg [8] originally proposed the theoretical possibility of forming rule-clusters or "corporations" within the rule-base of a Michigan-style classifier system [3]; a theory that was further developed by Smith [4].

In our previous work we have implemented a corporate classifier system (CCS) [5,6,7] based on the ideas of Smith [4] which demonstrates that rule-linkage can, for a certain class of problems, offer benefits to a system based on the zeroth-level classifier system (ZCS) [10]. Here we show that similar benefits can be gained when similar rule-linkage mechanisms are applied to XCS [11].

The paper is arranged as follows, a brief description of XCS is given in the next section, followed by an overview of the rule-linkage mechanisms implemented in CCS. The application of rule-linkage in XCS is then discussed and performances of XCS and CXCS are compared in two contrasting classes of environments. The first, Woods2 [11] tests the systems' abilities to form external associations, and in particular the capability to form accurately general hypotheses. The second class of environments is comprised of Delayed Reward Tasks (DRTs) [5] and tests the systems' abilities to form predominantly internal associations. A number of modifications to the rule-linkage mechanisms are then proposed which are shown to improve performance of CXCS in this second class of environments.

2 XCS

The most significant differences between XCS and traditional Michigan-style systems are that XCS dispenses with the internal message list and perhaps more importantly

P.L. Lanzi, W. Stolzmann, and S.W. Wilson (Eds.): LCS '99, LNAI 1813, pp.195-208, 2000.

that in XCS, rule fitness for the genetic algorithm (GA) [3] is based not on rule predictions (or strengths) but on the accuracy of these predictions (see also Frey and Slate [2]). The intention is to steer the population to form a complete and accurate mapping of the search space (consisting of optimally general classifiers) rather than to simply focus on the higher payoff niches in the environment.

A further difference is that rather than executing the GA panmicticly, XCS restricts GA activity to within niches defined by the match sets [1]. XCS has been shown to evolve rules that are maximally general, subject to an accuracy criterion. This encourages efficiency in knowledge representation within the rule-base. A brief overview of XCS functionality as implemented here is now given.

On each time-step, within the performance component, the system receives some binary-encoded sensory input and forms a match-set [M] consisting of all stimulus matching rules. A system prediction is then formed for each action represented in [M] according to a fitness-weighted average of the predictions of rules in [M] that advocate that action. The system action is selected either deterministically, probabilistically (roulette-wheel selection) or randomly from actions with non-zero predictions. Rules in [M] that advocate the selected action form an action-set [A]. The action is sent to the system effectors and a reward may or may not be received from the environment. If [M] is empty a covering operator is employed to create a new matching rule.

Reinforcement in XCS consists of updating three parameters, p, E and F for each qualifying rule.A rules fitness (F) is updated every time it belongs to $[A]_{-1}$ (or [A] in a single-step problem). The fitness is updated according to the relative accuracy of the rule within the set. There are three steps to the calculation:

1. Each rule's accuracy K_j is determined as follows:

$$K_j = \exp[(\ln \alpha)(E_j - E_0)/E_0)] * 0.1 \quad \text{for} \quad E_j > E_0 \text{ otherwise } 1.$$

2. A relative accuracy K'_j is determined for each rule by dividing its accuracy by the total of the accuracies in the set.
3. The relative accuracy is used to adjust the classifier's fitness F_j using the moyenne adaptive modifee (MAM) [11] procedure: If the fitness has been adjusted $1/\beta$ times, $F_j \leftarrow F_j + \beta(K'_j - F_j)$. Otherwise F_j is set to the average of the current and previous values of K'_j .

Next E_j is adjusted using P (see below) and the current value of p_j. The Widrow-Hoff technique is used as follows:

$$E_j = E_j + \beta(|P - p_j| - E_j).$$

Finally p_j is adjusted. The maximum $P(a_i)$ of the system's prediction array is discounted by a factor γ ($0 < \gamma <= 1$) and added to any external reward from the previous time-step. This value is called P and is used to adjust the predictions of the rules in $[A]_{-1}$ using the Widrow-Hoff delta rule [11] with learning rate β ($0 < \beta <= 1$).

$$p_j = p_j + \beta(P - p_j).$$

The GA acts on the match-set [M] . Two rules from the niche are selected for reproduction stochastically based on rule-fitness. The XCS population, [P] may be of fixed size or of variable size (and initialized to 0). A maximum size, N_p is defined. If [P] contains less than N_p members, the copies are inserted into the population and no compensating deletion occurs. Otherwise two rules are selected from [P] stochastically. Each rule keeps an estimate of the size of the match-sets in which it occurs. A rule's deletion probability is set proportional to this match-set size estimate; this tends to balance system resources across all presented niches.

The GA is activated within a match set if the number of time-steps since the last GA in that match-set exceeds a specified threshold. Each rule is time-stamped at birth with the value of a counter incremented each time-step. When a match-set is formed XCS computes the average time-stamp of its rules and executes the GA if the difference between the average and the current counter value exceed the threshold value (typically this threshold is set to 25).

An action selection strategy is randomly determined at the beginning of each task. There is a 50% likelihood that on each step the action will be selected randomly from those advocated within [M] by rules with non-zero predictions. Such trials are termed exploration trials. On the other trials, termed exploitation trials, a deterministic policy is adopted. During testing exploration is turned off for the last 1000 trials. The XCS GA is only active on exploratory trials, and as such is also turned off at the end of runs. The aim is to facilitate evaluation of the resultant rule-base's utility at the end of the learning period.

In this work, the system is equipped with a fixed size population (randomly initialised), and the concept of macro-classifiers [11] is not employed. Wilson states that this is simply a coding issue and as such should not effect results.

3 Corporate Classifier Systems

Our previous CCS, based on Wilson's ZCS model [10], employs linkage between rules in the population to form rule-chains or "corporations". Each rule in the population is equipped with two, initially inactive *link* parameters, a "link forward" and a "link back". When activated, either, or both of these links may reference another rule in the population. The result of such associations is a population of arbitrarily long rule-chains or corporations, whose members are treated as collective units, both by the discovery component and the performance component of the system.

Activities of the discovery component are based on a measure of *corporate fitness*. For a single rule, this value is the same as its strength parameter as determined by the performance component. For linked rules, corporate fitness is set to the mean strength of all rules in that particular corporation.

If a corporate rule is selected for deletion then the corporation is first disbanded (the linked-list is separated) and then the selected rule is deleted from the rule-base. If a corporate rule is selected for reproduction then the whole corporation is reproduced. The crossover mechanism is expanded to facilitate a form of *corporate crossover* which produces as offspring, a single hybrid corporation which inherits sections of both parent corporations (see figure 1).

Corporations are encouraged to encapsulate temporal chains of inference and so offer improved performance in tasks, such as the previously mentioned delayed reward tasks, which present arbitrarily ambiguous sensory stimuli. Rule-linkage, initially triggered with a fixed probability of 0.1 per time-step, acts across subsequent match sets ([M]) choosing a candidate to join probablistically (based on strength), from each niche. Unlike the original ZCS design, the GA is also restricted to acting in match set niches. These precautions ensure that corporations tend to represent viable sequences of previously presented stimuli, along with associated system responses.

In contrast to the original proposals of Wilson and Goldberg the performance component of CCS is made responsive to the presence of corporations. As in ZCS, on each time-step the system action is selected according to a roulette-wheel selection policy based on the strengths of rules within [M]. This rule is examined for corporate status (i.e. is it linked to another rule?). If a corporate rule has been selected then that corporation has absolute control over the system until either a reward is received (at which point a new task begins) or, the appropriate rule in the chain does not match the currently presented stimulus. In either event performance system functionality returns to standard ZCS behavior.

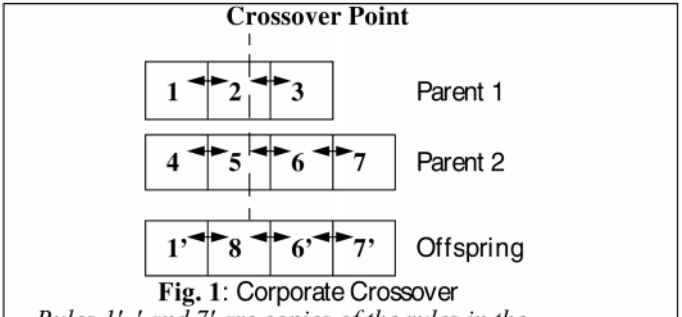


Fig. 1: Corporate Crossover

Rules 1', ' and 7' are copies of the rules in the original corporations. Rule 8 is the product of crossover between rule 2 and rule 5.

So, if on some time-step *t*, a corporate rule takes control of the system, then on the next step, *t+1*, if the next rule in the corporation matches the new stimulus, control is held and the action of this rule automatically becomes the system action at time *t+1*. This characteristic of the performance component is referred to as persistence and has been shown to enable CCS to overcome sensory ambiguity during multiple-step tasks [5].

Further encapsulation of corporations is achieved by preventing any rule which has an active “link back” (i.e. a follower) from entering [M], unless it links back to the rule that is currently in control of the system.

4 CXCS: Corporate XCS

The linkage mechanisms of CCS are now implemented in a version of Wilson’s XCS. This version differs from the original design in that on each invocation, the GA

(which acts in [M]) produces only a single offspring. As the XCS GA activation policy is based on the mean age of classifiers within the niche, it is anticipated that this difference will result in only minor performance variations. As the mean age within the niche will be less reduced by each GA action due to the addition of only one new rule, the GA should simply occur slightly more frequently within each niche. During comparisons, the GAs of both XCS and the corporate system, CXCS, produce a single offspring rule or corporation on each invocation.

Rules within the system are given the same linkage components as those in CCS and linkage occurs again between rules from subsequent match sets at a fixed rate (typically a 10% probability on each step). Like the GA, linkage occurs only on exploratory cycles and so is also turned off for the last 1000 trials of testing. In CCS, rule selection for linkage could be either random or probablistic, or deterministic, based on the relative strengths of rules within the niches. The equivalent parameter to ZCS/CCS rule strength in XCS is the prediction parameter. In XCS it is the accuracy of the prediction that is used to evaluate rules, and it is not in keeping with XCS philosophy to base discovery decisions on the prediction parameter alone. Accuracy and fitness are also discounted as possible weightings for rule selection for linkage. In CXCS it is possible that rules that appear to be inaccurate when evaluated alone are precisely the rules that could benefit from rule-linkage. If the inaccuracy is due to some sensory deception then the context of a corporate rule-chain may limit a rule's activation to instances in which it's action results in a more predictable consequence. In context the rule becomes more accurate. This is the main motivation for developing CXCS. With this in mind, selection for linkage in CXCS is determined randomly from rules (whose appropriate link is unattached) within the niche, imposing no bias based on the system's current perception of rule utilities.

Corporations are reproduced and evaluated collectively. As such rules within a corporation should share certain parameters used by the discovery component. These are fitness, which determines a rule's chance of selection for reproduction, and the estimate of mean match set size which determines a rule's chance of being selected for replacement; two parameters are introduced, "corporate fitness" and "corporate niche size estimate". For single rules these parameters are identical to their existing fitness and match set size estimates. For linked rules, these values can be determined in a number of ways. Each rule could be given the average fitness and match set size estimate of all rules within the corporation. Alternatively, corporate fitness could be based on the lowest exhibited fitness within the corporation. In this way, a corporation is considered only as accurate or fit as its weakest link. This approach certainly offers the theoretical advantage of a bias against unwanted parasites within corporations.In the initial design corporate fitness for each rule in a corporation will be set to the lowest exhibited fitness within the corporation. Corporate niche size estimates will be determined as the mean match set size estimate within that corporate unit.

As in CCS, corporations can, while they continue to match presented stimuli, maintain persistent control of the performance component. In CXCS corporations can only take control during exploitation cycles. To allow such behavior during exploration trials would represent a significant degradation of the system's discovery abilities. Again, as in CCS, *followers* (rules with an active "link-back" component) are given only limited access to [M].

When comparing systems that introduce different numbers of offspring per invocation of the GA it is important to consider the differences in relative rule replacement rates. Without such consideration it is possible to generate quite

misleading comparisons of systems as rule replacement concerns tend to be amongst the more fragile aspects of classifier system design. To counteract this, a variable element is introduced into the CXCS GA activation. The system records the number of rules reproduced on each invocation of the GA (i.e. the size of offspring corporation, S_c). When the existent activation policy indicates that the GA should fire, a further mechanism will only allow the GA to fire with probability set according to the reciprocal of the mean offspring size parameter, S_m (initialized to 1). This estimate is adjusted on each invocation of the GA according to the standard Widrow-Hoff delta rule [11] with the learning rate parameter β (typically 0.2), i.e. $S_m <- S_m + \beta(S_c - S_m)$. This modification to the GA activation mechanism ensures at least a more consistent rate of rule replacement throughout testing, however the drawback is that a corporate system, compared to a standard system will incur a relative reduction in crossover events. The more significant factor is perhaps the rule replacement rate and its effect on convergence within the rule-base, and so here, the variable GA activation policy is adopted for all tests.

An implementation of CXCS, as described above, is now compared to XCS initially in Wilson's Woods2, an environment that does not require the presence of internal associations within the system's rule-base, and then in a series of delayed reward tasks which can only be solved by the formation of internal associations between rules.

5 Analysis of Performance

5.1 WOODS2

Woods2 [11] is a two-dimensional, toroidal grid-world comprised of 30 x 15 cells. Each cell in the grid may be blank or occupied by one of four types of object, two of which are "food" and two are "rocks". The system is considered to be an artificial animal which traverses the rectilinear grid-world seeking food. It is capable of detecting the sensor codes of objects occupying its surrounding eight cells. These codes (each of which is three-bits long) comprise the system's stimulus at any time-step. 000 represents a blank cell, 110 and 111 represent food type objects, and 010 and 011 represent rocks. As such the stimulus length is 24-bits, with the left-hand three bits representing the cell due north of the current location, and the remainder corresponding to cells proceeding clockwise around it. On receipt of such a stimulus the system decides upon one of eight actions, which represent an attempt to move into one of the surrounding squares. If the cell is blank, the system moves into it, if the cell is occupied by a rock then the system is not able to make the move and if the cell contains food then the move is allowed and the system receives a reward (r_{imm}=1000), this is considered to be the termination of an individual trial. During testing, on the receipt of such a reward, the artificial animal, or animat [9] is randomly relocated in some blank cell, again ready to seek some food object. A record is kept of the mean number of time-steps taken to reach food over a period of 4,000 successive trials and this is used as a measure of system performance. For further details of Woods2 see [11].

5.2 Performance in WOODS2

The XCS and CXCS models described above are now tested in Woods2. System parameters for these tests are as in [11]:

Rulebase Size: N_p = 800,
Probability of # at an allele position in the initial population: P# = 0.5,
Initial rule prediction = 10.0,
Learning Rate: β = 0.2,
Discount Factor: γ = 0.71,
Probability of crossover per invocation of the GA: χ = 0.8,
Probability of mutation per allele in the offspring: μ = 0.01,
If the total prediction of [M] is less than ϕ times the mean of the population ([P]), covering occurs: ϕ = 0.5.
GA activation threshold parameter = 25,
Number of single-rules or corporations produced by GA as offspring per invocation, 1.
Initial rule error: = 0.0
Initial rule fitness: = 10.0
accuracy function parameter: $e_{0:}$ = 0.01
accuracy function parameter:α = 0.1
Linkage Rate: = 0.1,

Plots of performance are presented (figure 2) which show that both designs reach near optimal performance. Performance plots here represent the average steps to food in the last 50 exploit problems, and the curves are averages of ten runs.

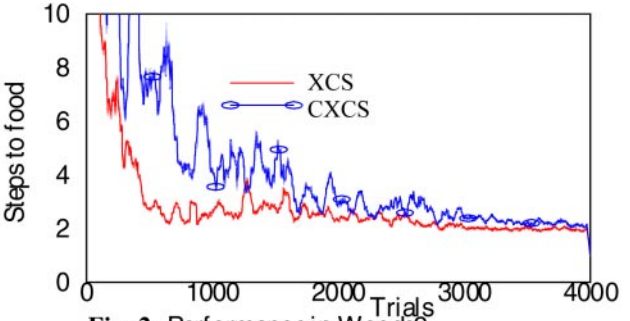


Fig. 2: Performance in Woods2

XCS achieves near optimal performance in Woods2 and so it was unlikely that CXCS would offer performance improvements in this Markovian environment which presents no sensory ambiguities to the system. The aim was simply to assess the extent of incurred overheads due to the more complex structures present in the new system. About 130 corporations of various sizes are present in the resultant rule-base and this figure breaks down approximately as follows:

This accounts for almost half of the rule-base, and so their presence can be considered significant. Their effect on performance however is less so. In this instance it appears that is an acceptable scenario. It should be stated that as the

linkage rate is increased from 0.1 the presence of corporations does start to degrade CXCS performance in Woods2 more significantly (not shown).

Table 1: Corporations present after testing in Woods2

Size	2	3	4	5+
No. Of Corp.	80	30	10	10

The systems are now tested in a series of environments in which it is hoped that corporations will in fact be able to improve system performance.

5.3 Simple Delayed Reward Tasks (DRTs)

On each of *N* timesteps the system is presented with a stimulus and must select one of *A* actions, where *A* is a variable integer value which defines the breadth of a maze. *N* is the number of states or nodes to a reward and thus defines the maze depth. After *N* steps, the system receives a reward from the environment and a new task then begins. The size of the reward depends on which route the system chooses and so over time the system learns the optimum reward yielding route through the maze.

There is however more than one maze. There can be up to *Mz* different mazes. The system is informed which particular maze it is being presented with only on the first time-step of each trial. On all subsequent steps the stimulus is representative only of the current time-step in the trial. The maze is selected randomly at the start of each trial.

Figure 3 illustrates a simple task of this type with *A* set to 2, *N* set to 2 and *Mz* set to 2. The environmental stimulus at each time-step is also included. In this example, a reward of 1000 is awarded for one route on each map. All other routes receive a reward of 0.

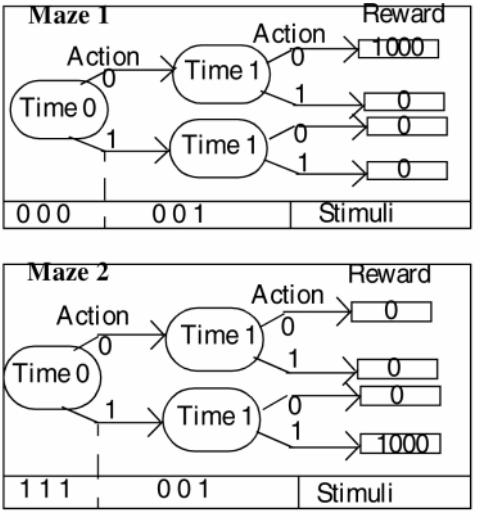


Fig. 3: Simple Delayed Reward Task - DRT 2:2

Throughout this work, as in figure 3, the message length L is set to 3. With L set to 3 there are 8 possible stimuli and so for a two map problem the maximum depth will be 7, as the first time-step (ts_0) takes two stimuli. Similarly with L set to 3 a four maze problem may have a maximum depth of 5.

In the task depicted in figure 3 rule-linkage will enable the system to determine the appropriate move on the second timestep (ts_1). In maze one the correct action is 0 and in maze two it is 1 (the stimulus however is the same - 001).

5.4 Performance in DRTs

Tests are now conducted in a series of four such delayed reward tasks. On each time-step the system must choose one of two actions (0 or 1). The first task, 2:2 consists of two mazes of length two the second task, 2:3 consists of two mazes of length three, etc. One path in each maze will yield a reward of 1000, all others return a reward of 0. The mazes are set up so that if the system selects the same action on each step through the maze it will be guaranteed a reward of 0. This precaution ensures that the successful solution of the mazes is not achievable by a single completely general rule, and will in fact require some form of cooperative behavior within the rule-base. Tests consist of a series of 10,000 trials and all curves are again averages of ten runs. The plots (figures 4, 5, 6 and 7) represent the average score over the last fifty exploitation trials. All parameters are as in the previous tests in Woods2 with the exception of the population size, (400), and the probability of a # at an allele position in the condition of a rule (0.33). These are more standard parameter settings, as used by Wilson when testing ZCS [10], and also when testing XCS in the Boolean Multiplexor Problems [11]. Figures 4 to 7 also include plots of CXCS with linkage rate increased from 0.1 to 0.25. It can be seen that in these environments an increase in rule-linkage does provide some benefit.

According to the XCS action selection strategy, on a fixed proportion of trials selection will be random from all advocated actions. On such an exploratory trial all rules are likely to receive variations in profit according to the different contexts in which they fire (even if all rules are 100% specific). This will clearly result in low perceived accuracy for all firing rules. XCS accuracy is determined by a quite severe function with a sharp cutoff beyond the acceptable error margin. In such mazes it is possible that all firing rules on any time-step will exhibit a perceived accuracy of 0, leading to each rule having a resultant relative accuracy, and thus a fitness based on the reciprocal of the niche size. Although, during an exploitation cycle, rules representing the optimal system action may be present in the niche, with the bid based on the prediction scaled according to fitness, there will be no discernible bias towards the higher reward yielding action and so the system has certain difficulties mapping the maze which imposes rule co-dependencies that XCS is unable to facilitate.

The effects of the above problem are illustrated by the state of the rule-base at the end of testing and also by the continual activation of the cover operator, especially during the last 1000 exploitation trials. On a 2:2 task for instance, the rule-base will contain many rules of varying specificity that match the “second step stimulus” (001). All of these with the exception of the fully general ### rules have a fitness value of 0. The prevalent ### rules tend to increasingly occupy both niches and thus their prediction values will slowly fall to 0 (due to the previously mentioned precautions

taken with the reward scheme designs), and over this period their fitness values will gradually rise to maximum fitness. Eventually [M] prediction on the second step falls below the threshold parameter value and the cover operator is invoked. This process continues throughout the test period.

CXCS does exhibit some ability to map these "internal association" tasks, however results are somewhat disappointing and certainly do not compare to the equivalent results produced by running the CCS model in the same tasks [5,6]. In a 2:2 task, for example, the resultant rule-base at the end of testing will typically contain about 100 corporations. These will be of indeterminate length and there will be a number of excessively long corporations. In the equivalent CCS test, the resultant rule-base would typically contain approximately 200 corporations, of which, just about all were of length two. In other words the entire population had linked to form corporations of the appropriate length to tackle the presented task.

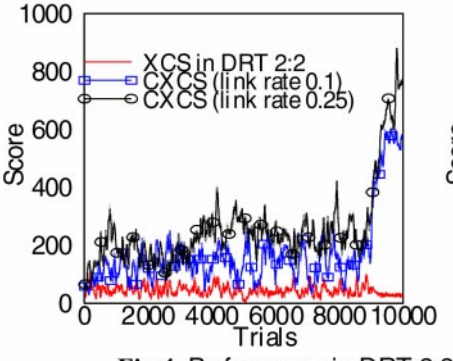


Fig 4. Performance in DRT 2:2

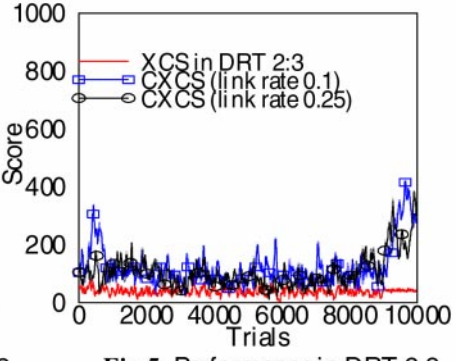


Fig 5. Performance in DRT 2:3

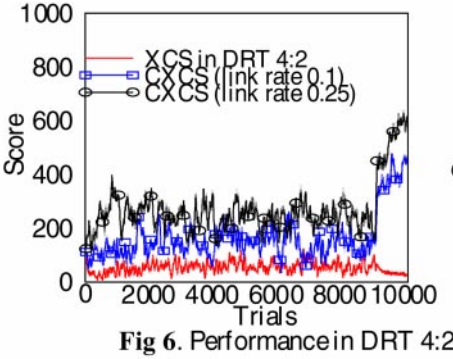


Fig 6. Performance in DRT 4:2

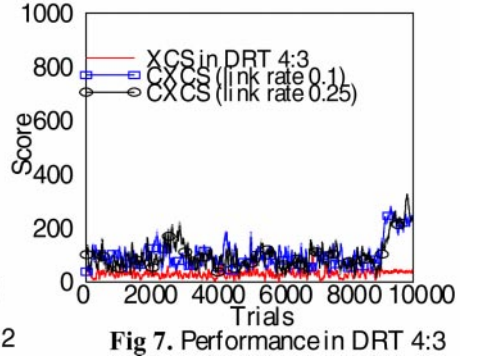


Fig 7. Performance in DRT 4:3

Finally, in the next section, we show that the CXCS mechanisms can be modified to provide significant performance improvements in the more complex delayed reward tasks.

6 Modifications to CXCS

6.1 A Re-evaluation of Corporate Fitness

Wilson [11] suggests as a future avenue of research that the concept of fitness based on accuracy of prediction could be extended to classifiers with expectons. An expecton takes the form of an additional condition, but is a prediction of the resulting sensation or stimulus.

Such a system does not only match stimulus action pairs with prediction values, but with some anticipated stimulus. Fitness for such rules may not only be based on accuracy of prediction but may also, or separately, represent the accuracy of the expecton in predicting the next sensation. This principle can be adapted for use in CXCS.

Although rules in CXCS do not have an expecton, if their "link forward" is active, they are still associated with information about some anticipated stimulus. This is in the form of the condition of the rule that they are linked to. Actually such a rule may be associated with information regarding any number of anticipated future stimuli, dependent on the length of the corporation.

With this in mind, an alternative measure of fitness may be employed in CXCS for rules with active forward links, based not only on the accuracy of their prediction but also scaled according to the ratio of "control hits"(C_h) to the total number of times that control is received (C_c). If a rule takes control of the performance component and on the next time-step the following rule in the corporation matches the subsequent stimulus and therefore inherits control of the system successfully, then this is considered to be a "control hit". If the rule fails to match the presented stimulus, then this is a "control miss". As such, this ratio acts as a measure of how accurately the corporation is representative of some perceived aspect of the test environment. C_h and C_c are updated each time the rule takes control of the system (but on formation of $[M]_{+1}$). The control ratio parameter is adjusted as follows: $C_r = C_h / C_c$. This is determined prior to fitness calculations. On fitness adjustments, the relative accuracy parameter, K' is scaled according to the control ratio before the fitness parameter is updated in the usual manner. As previously, corporate fitness is consistent for all rules in the corporation and is based on the lowest member fitness in the corporation. Corporate niche size estimate is again set to the mean match set size estimate of member rules.

Figure 8 shows performance plots of this modified version of CXCS in the same series of four delayed reward tasks as used in section 5. Rule linkage rate is again set to 0.25 and all other parameters are as in section 5.

The new fitness evaluation produces improved results in the harder delayed reward tasks. At the end of testing the system generally contains about 100 corporations, most of which are of the appropriate length for the particular task (i.e. of length two or three). On inspection of the plots it is clear that the excessive number of exploration trials are creating difficulties for the performance component. This is evident by the abrupt performance improvement when exploration is turned off for the last 1000 trials of testing.

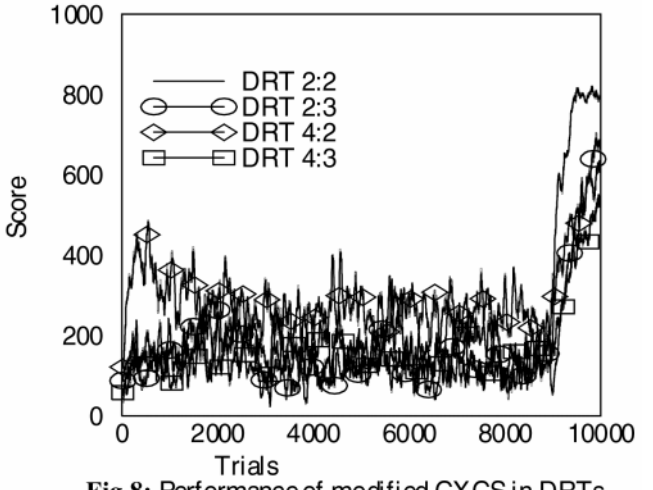


Fig 8: Performance of modified CXCS in DRTs.

6.2 A Variable Exploration Rate

An adjustment to the exploration/exploitation decision process may offer performance benefits. One approach experimented with here is to reduce the probability of exploration periodically throughout the evaluation runs.

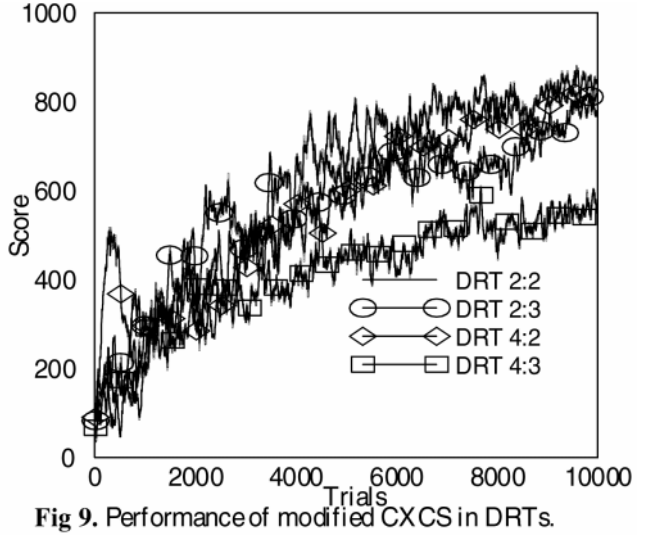


Fig 9. Performance of modified CXCS in DRTs.

The initial probability is 0.5 and every 100 trials this is reduced by one tenth of the current value. This simulated annealing of the system over time offers the performance component increasing opportunities to evaluate the corporate rule structures. Due to the persistent nature of corporations this has become a somewhat serial affair, and so perhaps the performance component now requires a reduction in

the relative rate of genetic activity. Performance plots (Figure 9) show that this modification offers some benefit to CXCS in the harder of the four delayed reward tasks. Apart from the variable exploration rate, the system is identical to the system presented in section 6.1.

Within the population, accurate and thus fit corporations are present which collectively map the entire problem space of these tasks, i.e. for each map all possible routes tend to be represented along with reasonable prediction values for these routes, as represented by the predictions of component rules within the corporations.

One final consideration is the evaluation of the modified CXCS model in Woods2. It is important to ensure that the new mechanisms do not disrupt system stability in a less controlled environment than the time constrained delayed reward tasks. Figure 10 shows performance in Woods2 of the modified CXCS. All parameters are as in section 5.2. At the end of 4,000 trials the system reaches food in an average of two steps to food. This plot is comparable with the earlier plot (Figure 2) for CXCS with the original configuration; improvements for delayed reward tasks have no significant effect on stimulus response tasks here.

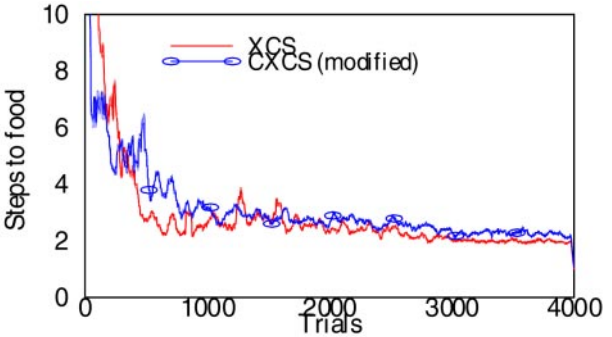


Fig 10. Performance in Woods2

7 Conclusions

This work has shown that it is possible to include the corporate concept in a system based on XCS and achieve similar benefits to those gained in ZCS. The resultant system, CXCS, demonstrates reasonably equivalent abilities to XCS in tackling Woods2, an environment which offers many opportunities for generalization. It is also capable of making some internal associations necessary to solve the delayed reward tasks presented here.

It has further been shown that performance improvements in the delayed reward tasks can be gained by re-evaluating corporate rule fitnesses according to their consistency in maintaining control of the system, and also by reducing the exploration rate during training. The modified CXCS was tested in Woods2 and it was shown that the modifications do not excessively disrupt performance in this environment.

From rule-base inspection, the modified fitness calculation appears to encourage increased generalization within corporations, certainly for “followers”, and to an extent which accuracy permits. This is most apparent when examining the rule-base

after the system has been trained in Woods2 (not shown). A full investigation of the generalization abilities of our CXCS represents future work.

Acknowledgments. The authors thank Pier Luca Lanzi for useful guidance during the development of XCS.

References

[1] Booker, L. (1985) “Improving the performance of Genetic Algorithms in Classifier Systems.” Proceedings of the First International Conference on Genetic Algorithms and their Applications. (pp. 80-92). Lawrence Erlbaum Assoc.

[2] Frey, P.W., & Slate, D.J. (1991). “Letter recognition using Holland-style adaptive classifiers.” Machine Learning, 6, 161-182.

[3] Holland, J. H. (1975) (Ed.) Adaptation in Natural and Artificial Systems. Univ. of Michigan Press, Ann Arbor.

[4] Smith, R. E. (1992) "Memory exploitation in learning classifier systems." Evolutionary Computation, 2(3): 199-220.

[5] Tomlinson, A. & Bull, L. (1998) “A Corporate Classifier System.” In Eiben, A.E., Back, T., Schoenauer, M. & Schwefel, H.P.(Eds.) Parallel Problem Solving from Nature, PPSN V, (pp. 550-559), Springer.

[6] Tomlinson, A. & Bull, L. (1999a) “On Corporate Classifier Systems: Increasing the Benefits From Rule Linkage.” in Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M. & Smith, R. E.(Eds) GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference,(649-656), Morgan Kaufmann.

[7] Tomlinson, A. & Bull, L. (1999b) “A Zeroth Level Corporate Classifier System.” In A.S.Wu (Ed.) GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference Workshop Program, (pp 306-313).

[8] Wilson, S. W. & Goldberg, D. E. (1989) "A critical review of classifier systems." In Schaffer, J. D. (Ed.) Proceedings of the Third International Conference on Genetic Algorithms, (pp. 244-255), Morgan Kaufmann.

[9] Wilson, S. W. (1985) “Knowledge growth in an artificial animal.” Proceedings of the First International Conference on Genetic Algorithms and Their Applications, (pp. 16-23). Hillsdale, NJ: Lawrence Erlbaum Assoc.

[10] Wilson, S. W. (1994) "ZCS: A zeroth level classifier system." Evolutionary Computation, 2 (1): 1-18.

[11] Wilson, S.W. (1995) "Classifier Fitness Based on Accuracy." Evolutionary Computation, 3(2): 149-176.

Get Real! XCS with Continuous-Valued Inputs

Stewart W. Wilson

The University of Illinois, Urbana-Champaign IL 61801, USA
Prediction Dynamics, Concord MA 01742, USA
wilson@prediction-dynamics.com

Abstract. Classifier systems have traditionally taken binary strings as inputs, yet in many real problems such as data inference, the inputs have real components. A modified XCS classifier system is described that learns a non-linear real-vector classification task.

1 Introduction

A classifier system is an on-line learning system that seeks to gain reinforcement from its environment based on an evolving set of condition-action rules called classifiers. Via a Darwinian process, classifiers useful in gaining reinforcement are selected and propagated over others less useful, leading to increasing system performance. The classifier system idea is due to Holland [2], who laid out a comprehensive framework that included generalization of classifier conditions, internal message-passing and reinforcement, and computational completeness. For a number of years, the idea has inspired many. A "population" of hypotheses, generated plausibly but partly randomly from other hypotheses and subject to confirmation by the environment, intuitively corresponds to our own mental experience, and indeed suggests a cognitive model. Classifiers that read input detectors and respond by posting messages seem to bridge the gap between physical stimuli and concepts. Generalization by classifiers to cover related input situations seems to correspond to the formation of rudimentary perceptions. The extension to learning and thinking of principles of variation and selection—successful and ubiquitous elsewhere in life—seems natural and deserving of extensive investigation.

While its genes were promising, the evolution of the classifier system idea has not been rapid. As a complex adaptive system that also needed to be successful against a relatively independent environment, learning classifier systems (LCS) were found not to work well "out of the box". Instead, progress has come via simplifications of LCS structure, controlled experiments with environments that tested individual aspects of the system, and some changes in the original framework. Among the problems encountered were performance better than random but rarely reaching optimality, difficulty coupling sequential messages, and inaccurate generalization. One result of this uncertain workability was that an LCS was rarely the technique of choice in practical applications.

Yet applicability in at least one area—inference from data—may in fact be great. Data inference is classification, which LCSs are for! A classifier system—inherently non-linear—should be able to find non-linear categories in complex

P.L. Lanzi, W. Stolzmann, and S.W. Wilson (Eds.): LCS '99, LNAI 1813, pp. 209–219, 2000.

data. And because it consists of discrete rules, an LCS has the virtue, compared with other techniques such as networks and nearest-neighboring, of exhibiting its results with a transparency that permits human comprehension. Finally, compared with other techniques, classifier systems, because they evolve general rules, should be able to learn at a complexity cost proportional to the target concept and not the underlying input space.

XCS [11,12,13] is a new kind of LCS that reaches optimal performance on quite difficult problems and finds accurate maximal generalizations. Its applicability to data inference tasks is supported by good results with XCS on the Monk's Problems [9], adaptation of LCSs to risk-of-outcome analysis [3], demonstration of generalized (s-expression) conditions in XCS [7], and demonstration of noise rejection by XCS [6]. One area that has not received attention, but is important in data inference, is classification of input vectors having real-valued components.

Continuous variables such as temperature, concentration, or age may be decisive in classification, with certain ranges of the variables implying one class and other ranges implying another. LCSs traditionally have taken binary (bitstring) inputs. Sometimes a binary variable is taken to represent a pre-thresholded continuous one, but then the thresholding has not been done adaptively. In a version of XCS taking real inputs—call it "XCSR"—optimally decisive thresholds should be found automatically. Fuzzy classifier systems [1]—LCSs with rules based on fuzzy logic—take real inputs and generate discrete or continuous outputs. Most fuzzy systems, however, do not adapt their membership functions, though this may, and should, change eventually. Just the same, it is desirable to try to develop XCSR, which if it works will have the advantages of XCS and avoid the addition of fuzzy techniques.

This paper reports two first experiments with XCSR. The results are interesting, and the real domain is found to introduce issues new to LCSs. The next section explains the test problem used, the "real multiplexer". XCSR is introduced in the following section, first by explaining how it differs from XCS, and then briefly reviewing the elements that remain the same. The two experiments are then presented, followed by discussion and conclusion.

2 Test Problem

To test XCSR, we wanted a non-linear classification problem for real vectors. As a simplification, it was assumed that in the input vector $x = (x_0, ..., x_n)$, each component x_i is restricted to the range $0.0 \leq x_i < 1.0$. If in an actual problem the data ranges are known in advance, such scaling implies no loss of generality. If not, XCSR may still be capable of adapting to the actual ranges, but this has not been investigated as yet. We also assumed the vector was to be classified into one of just two classes, 0 or 1, e.g., healthy (0) or diseased (1), as might occur in epidemiological inference. Finally, it was assumed that the class value depended on whether or not certain input variables were in certain ranges, e.g., $(0.12 \leq x_2 < 0.38) \wedge (0.31 \leq x_4 < 1.0) \Rightarrow$ class 1.

Given these assumptions, our test problem was an adaptation to real values of the relatively well-known Boolean 6-multiplexer problem. This provided a non-linear two-class task of moderate difficulty, and the results could be compared with previous results on the Boolean version.

The Boolean 6-multiplexer function takes as input a six-bit string and outputs a truth value, 1 or 0. One way to get the output is to consider the two leftmost string bits as an unsigned binary number addressing one of the four remaining bits. Thus in the string 011010, the first two bits "01" address bit number 1 of the remaining bits. The value of that bit is 0, which becomes the function value. Alternatively, in disjunctive normal form, the function is given by $F_6 = b'_0 b'_1 b_2 + b'_0 b_1 b_3 + b_0 b'_1 b_4 + b_0 b_1 b_5$, where the subscripts index the bits left to right and primes denote negation.

To define the corresponding "real 6-multiplexer" (RF_6), assume we are given a real vector $x = (x_0, ..., x_5)$ in which $0.0 \leq x_i < 1.0$. Then for thresholds $0.0 \leq \theta_i < 1.0$, interpret x_i as 0 if $x_i < \theta_i$, else 1. The value of RF_6 is then the value of F_6 applied to the bitstring that results from these interpretations. Given random vectors x, XCSR is supposed to learn to return the value of RF_6. XCSR has of course no prior knowledge of F_6 nor of the thresholds, θ_i.

Note that in the earlier broad problem statement, it was assumed that classification would depend on variables being in certain ranges. In RF_6, the dependence is on variables being above or below thresholds, and this is not as general. Testing on ranges—and on negations of ranges, or disconnected ranges—awaits further research. However, as will be seen next, XCSR is at present designed for connected ranges and could presumably be extended.

3 XCSR

3.1 Changes from XCS

XCSR differs from XCS only at the input interface, in its mutation operator, and in the details of covering. The differences arise solely from changing the classifier condition from a string from {0,1,#} to a concatenation of "interval predicates", $int_i = (c_i, s_i)$, where c_i and s_i are reals. To address RF_6, XCSR uses classifiers whose conditions have six interval predicates. A classifier matches an input x if and only if $c_i - s_i \leq x_i < c_i + s_i$, for all x_i. Thus c_i can be thought of as the center value of int_i, and s_i, termed "spread", is a delta defined with respect to c_i. A consequence of using interval predicates is that the number of numerical values or alleles in the condition is twice the number of components in x.

Crossover operates in direct analogy to crossover in XCS. The crossover point can occur between any two alleles, i.e., it can occur within an interval predicate as well as between predicates. Mutation, however, is different. Several preliminary experiments were done, and the best method appears to be to mutate an allele by adding an amount $\pm rand(m)$, where m is 0.1, $rand$ picks a real value uniform randomly from $[0, m)$, and the sign is chosen equiprobably. The result is mutation by a scaled random increment.

"Covering" occurs when no existing classifier matches x. In both XCS and XCSR a new classifier is created which does match. In XCSR the new condition has components $(c_0, s_0, ..., c_5, s_5)$, where $c_i = x_i$ and $s_i = rand(s_0)$, with s_0 a constant such as 0.5. In XCSR, as in recent versions of XCS, a covering classifier (with separately generated condition) is created for each possible action. If the population of classifiers has reached its allowable maximum, deletion of classifiers occurs to make room for the new ones (deletion is generally not necessary; initial populations are empty and covering normally occurs only at the very beginning of a run).

3.2 Common Features of XCSR and XCS

The following is an abbreviated description of XCS (see [11,12,4] for more detail). XCS is designed for both single- and multiple-step tasks, but this description will apply only to XCS for independent single-step tasks in which an input is presented, the system makes a decision, and the environment provides some reward. All material in this section applies equally to XCSR.

Each classifier C_j in the population [P] has three principal parameters: (1) *payoff prediction* p_j, which estimates the payoff the system will receive if C_j matches and its action is chosen by the system; (2) *prediction error* ϵ_j, which estimates the error in p_j with respect to actual payoffs received; and (3) *fitness* F_j, computed as later explained. It is convenient to divide the description of a single operating cycle or time-step into the traditional performance, update (reinforcement), and discovery components.

Performance Upon presentation of an input, XCS forms a match set [M] of classifiers whose conditions are satisfied by the input. If no classifiers match, covering takes place as described earlier. Then for each action a_k represented in [M], the system computes a fitness-weighted average P_k of the predictions p_j of each classifier in [M] having that action: $P_k = \sum_j F_j p_j / \sum_j F_j$. P_k is termed the *system prediction* for action k.

Next, XCS chooses an action from those represented in [M] and sends it to the environment. According to the action-selection regime in force, the action may be picked randomly or otherwise probabilistically based on the P_k, or it may be picked deterministically—i.e., the action with the highest P_k is chosen. Finally, an action set [A] is formed consisting of the subset of [M] having the chosen action.

Update In this component, the parameters of the classifiers in [A] are re-estimated according to the reward R returned by the environment as a consequence of the system's taking action a_k. First, the predictions are updated: $p_j \leftarrow p_j + \beta(R - p_j)$. Next, the errors: $\epsilon_j \leftarrow \epsilon_j + \beta(|R - p_j|)$. Third, for each C_j, an *accuracy* κ_j is computed: $\kappa_j = 0.1(\epsilon_j/\epsilon_0)^{-n}$, for $\epsilon_j > \epsilon_0$, else 1.0. Then, from the κ_j, each classifier's *relative accuracy* κ'_j is computed: $\kappa'_j = \kappa_j / \sum_j \kappa_j$. Finally the fitnesses F_j are updated according to: $F_j \leftarrow F_j + \beta(\kappa'_j - F_j)$.

Discovery On some time-steps, XCS executes a genetic algorithm within [A]. Two classifiers are chosen probabilistically based on their fitnesses and copied. The copies are crossed (two-point crossover) with probability χ, and then mutated with probability μ per allele. The resulting offspring are inserted into [P]; if the population size is already at its maximum value, N, two classifiers are deleted. The probability of deletion of a classifier is determined by Kovacs's [5] method and is designed to preferentially remove low-fitness classifiers that have participated in a threshold number of action sets—that is, have had sufficient time for their parameters to be accurately estimated.

Whether or not to execute the GA on a given time-step is determined as follows. The system keeps a count of the number of time-steps since the beginning of a run. Every time a GA occurs, all classifiers in that [A] are "time-stamped" with the current count. Whenever an [A] is formed, the time-stamp values of its members are averaged and the average is subtracted from the current count; if the difference exceeds a threshold θ_{GA}, a GA takes place.

A *macroclassifier* technique is used to speed processing and provide a more perspicuous view of population contents. Whenever a new classifier is generated by the GA (or covering), [P] is scanned to see if there already exists a classifier with the same condition and action. If so, the *numerosity* parameter of the existing classifier is incremented by one, and the new classifier is discarded. If not, the new classifier is inserted into [P]. The resulting population consists entirely of structurally unique classifiers, each with numerosity ≥ 1. If a classifier is chosen for deletion, its numerosity is decremented by 1, unless the result would be 0, in which case the classifier is removed from [P]. All operations in a population of macroclassifiers are carried out as though the population consisted of conventional classifiers; that is, the numerosity is taken into account. In a macroclassifier population, the sum of numerosities equals N, the traditional population size. [P]'s actual size in macroclassifiers, M, is of interest as a measure of the population's space complexity.

4 Experiments

In each of the following experiments, random real vectors, with components $0.0 \leq x_i < 1.0$, were formed and presented to XCSR. Each such presentation was termed a *problem*. The action-selection regime consisted of *explore* problems which occurred with probability 0.5, and *test* problems the rest of the time. On an explore problem, XCSR chose its action uniform randomly from among those in [M], and update and discovery operations were carried out as previously described. Test problems were designed to determine how well the system could do if it chose its action deterministically. The measure of *system performance* was a moving average of the fraction of correct actions (correct values of RF_6) on the previous 50 test problems. Also measured was a similar moving average of $|R - P_k|$ (with P_k the prediction for the chosen action), termed *system error*. On test problems, the update and discovery components were disabled.

In the experiments, $R = 1000$ for the correct action, 0 otherwise. XCSR parameters were as follows: $N = 800$, $\beta = 0.2$, $\epsilon_0 = 10$, $n = 5$, $\theta_{GA} = 12$, $\chi = 0.8$, $\mu = 0.04$, $m = 0.1$, $s_0 = 1.0$. Other values of N, θ_{GA}, χ, μ, m and s_0 were tried in Experiment 1; from this limited evaluation, the values given appeared to be best and were used in both experiments.

4.1 Experiment 1

The aim of this experiment was to imitate, as closely as possible, the Boolean 6-multiplexer task, except that the input would be real. In the Boolean task as traditionally conducted, both outcome cases (1 and 0) are equally probable, as are all input bitstrings. An equivalent regime can be obtained in the real case by setting all thresholds θ_i at 0.5, since the underlying real vectors are uniform randomly generated. These were the thresholds in Experiment 1. Figure 1 shows

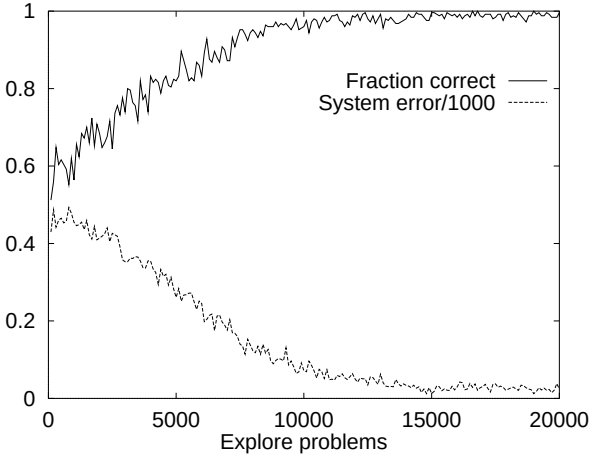


Fig. 1. Fraction correct (solid line) and system error/1000 (dashed) vs. number of explore problems in Experiment 1.

performance and system error averaged over five runs each consisting of 20,000 explore problems. Performance reaches its maximum—approximately 98%—at about 15,000 problems. System error reaches a minimum at a similar point. Compared with typical results on the Boolean 6-multiplexer, arrival at high performance is substantially slower. Recent experiments with XCS on F_6 get to 100% performance in about 1,500 problems, so there is a factor of difference of about ten.

Two sources for the difference come to mind. One is simply the fact that on RF_6, XCSR's classifiers have 12 alleles in their conditions vs. six for XCS on F_6; thus the search space has higher dimensionality. Experiments with XCS on the 11-multiplexer (Wilson 1998)—which requires 11 alleles in the condition—took

about 10,000 problems to reach 100% performance. A second possible source of difference is that whereas each Boolean dimension has just two allele values, each dimension in the real case has in principle indefinitely many values.

Of course the effective resolution of a real dimension depends on the quality of performance desired. High performance can only be reached if the system accurately estimates the thresholds, and this will take longer compared with the Boolean case, where the "thresholds"—in effect, choosing between 0 and 1 in the condition—are in a sense maximally coarse. However, the proper analysis of search time vs. resolution is not clear.

In the Boolean case, performance reaches a solid 100%, whereas here it does not, with the level reached depending on the mutation rate and technique. Preliminary experiments included mutation based on $\pm m$ instead of $\pm rand(m)$. This did not produce as good maximum performance, for any mutation rate. The random function, by introducing arbitrarily small mutation increments, probably contributes to a closer ultimate approach to 100%.

```
                                                                                ACT PRED  ERR FITN NUM
 0. |OOOOOOOOOO|.....oOOOO|OOOOOOOOOO|.....oOOOO|oOOOOOOOOO|OOOOOOOo..| 1   0.  .000   14.  1
 1. |.....oOOOO|.....oOOOO|OOOOOOOOOO|OOOOOOOOOO|oOOOOOOOO|OOOOo.....| 1   0.  .000   53.  2
 2. |.....oOOOO|.....oOOOO|OOOOOOOOOO|.....oOOOO|OOOOOOOOO|OOOOo.....| 1   0.  .000   40.  1
 3. |.....oOOOO|.....oOOOO|OOOOOOOOOO|OOOOOOOOOO|oOOOOOOOO|OOOOo.....| 1   0.  .000   50.  1
 4. |.....oOOOO|.....oOOOO|OOOOOOOOOO|OOOOOOOOOO|OOOOOOOOO|OOOOo.....| 1   0.  .000   50.  1
 5. |.....oOOOO|.....oOOOO|OOOOOOOOOO|OOOOOOOOOO|OOOOOOOOO|OOOOo.....| 1   0.  .000  140.  3
 6. |.....oOOOo|OOOOOo....|OOOOOOOOOO|OOOOOOOOOO|OOOOo.....|OOOOOOOOOO| 1  34.  .081    5.  2
 7. |....oOOOOO|.....oOOOO|OOOOOOOOOO|OOOOOOo...|OOOOOOOOOO|OOOOo.....| 1   0.  .000   56.  2
 8. |.....oOOOO|.....oOOOO|OOOOOOOOOO|....oOOOOO|OOOOOOOOOO|OOOOo.....| 1   0.  .000   41.  1
 9. |....oOOOOO|.....oOOOO|OOOOOOOOOO|.....oOOOO|OOOOOOOOOO|OOOOo.....| 1   0.  .000   58.  1
10. |.....oOOOO|.....oOOOO|OOOOOOOOOO|.....oOOOO|OOOOOOOOOO|OOOOo.....| 1   0.  .000   46.  1
11. |.....oOOOO|.....oOOOO|OOOOOOOOOO|....oOOOOO|OOOOOOOOOO|OOOOo.....| 1   0.  .000   85.  2
12. |.....oOOOO|.....oOOOO|OOOOOOOOOO|.....oOOOO|OOOOOOOOOO|OOOOo.....| 1   0.  .000   43.  1


                                                                          ACT PRED  ERR  FITN NUM
 0. |.572,.985|.924,.393|.322,0.99|.948,.417|.818,.812|.331,.404| 1   0.  .000   14.  1
 1. |.786,.264|0.89,.364|.602,0.99|0.23,.884|.796,.769|.228,.268| 1   0.  .000   53.  2
 2. |.794,.264|0.89,.364|.262,0.99|.868,.344|.665,.769|.228,.268| 1   0.  .000   40.  1
 3. |.794,.264|.807,.262|.602,0.99|0.23,.884|.796,.769|.228,.268| 1   0.  .000   50.  1
 4. |.794,0.28|.807,.262|.684,0.99|0.23,.884|.717,.769|.228,.268| 1   0.  .000   50.  1
 5. |.794,.264|.807,.262|.602,0.99|0.23,.884|.717,.769|.228,.268| 1   0.  .000  140.  3
 6. |.743,.232|.172,.404|.813,.903|0.41,.841|.092,.366|.506,.658| 1  34.  .081    5.  2
 7. |.775,.332|.807,.262|.476,.687|.275,.344|.716,.874|.205,.233| 1   0.  .000   56.  2
 8. |.786,.264|.807,.262|.288,0.99|.818,.322|.717,.783|.181,.269| 1   0.  .000   41.  1
 9. |.798,.357|0.89,.364|.247,0.99|.894,.344|.665,.732|.207,.233| 1   0.  .000   58.  1
10. |.798,.264|0.89,.364|.247,0.99|.894,.344|.665,.732|.207,.269| 1   0.  .000   46.  1
11. |.798,.264|.807,.262|.288,0.99|.818,.322|.717,.783|.207,.269| 1   0.  .000   85.  2
12. |.798,.264|.807,.262|.288, 1.0|.818,.274|.717,.783|.207,.269| 1   0.  .000   43.  1
```

Fig. 2. An action set [A] from Experiment 1. Upper half: condition predicates shown graphically (see text). Lower half: actual c and s values of predicates. Also shown: ACTion, PREDiction, ERRor, FITNess, and NUMerosity of each classifier.

It is interesting to examine some classifiers evolved by XCSR. Figure 2 shows an action set. In the first part of the figure, the classifiers are represented using a crude graphic notation. The unit interval is divided into 10 equal parts. If a part

contains ".", no value in that part is accepted by the predicate; if it contains "o", some values are; and if it contains "0" all values are accepted. Thus ".....o0000" is an interval predicate accepting inputs greater than some value between 0.5 and 0.6 and less than 1.0. The lower half of the figure shows the same classifiers but with the predicates notated directly with their c and s values. Also included in the figure, following the classifiers, are their action, prediction, error, fitness, and numerosity values. The error and fitness values are scaled. Error is shown as $\epsilon_j/1000$, fitness as $1000F_j$.

Notice how the system has "sculpted" the predicates and is in effect finding the thresholds. Most predicates either show ranges between 0.0 and 0.5, 0.5 and 1.0, or are "don't cares" i.e., "0000000000". In Boolean terms, most of the classifiers could be said to have the form $11\#1\#0 : 1 \Rightarrow 0$ or $11\#\#\#0 : 1 \Rightarrow 0$ (the bit before the arrow is the action, the integer after the arrow is the prediction). The latter classifier is more general than the former, while still as accurate, and it is likely eventually to drive the former out (see [11,12,4] for a discussion of accurate generalization).

In the lower half of Figure 2, using the raw c and s values, we can see that the effective thresholds are not quite 0.5, which must account for a good part of the system's residual error. Also interesting is that the number of unique c and s values is relatively small, with the diversity among the classifiers caused mainly by different combinations of those values, i.e., by crossover. While it might be predicted that mutation of real values, especially using $rand(m)$, would make every predicate different in detail, this is not particularly the case.

4.2 Experiment 2

This experiment was like Experiment 1 except that the interpretation thresholds were 0.25, 0.75, 0.25, 0.75, 0.25, and 0.75. In actual data inference problems, the relevant data ranges or thresholds are not only unknown but are in general different from each other. Experiment 2 was designed to model this situation, if somewhat simplistically. The new thresholds do not affect the relative prevalence of the two outcome cases: they are still equally probable, which is generally unlike real-world data sets. In contrast to Experiment 1, however, some inputs in Experiment 2 are much more likely than others. For example, an input whose Boolean interpretation is 010101 has probability $(1/4)^6$, while an input with interpretation 101010 has probability $(3/4)^6$, a ratio of $3^6 = 729$. Thus input frequencies in Experiment 2 were highly variable.

Figure 3 shows the performance and error results. The maximum performance—about 93%—is noticeably lower than in Experiment 1, although it is reached sooner. System error is greater than in Experiment 1. At present, we do not understand these differences, although the fact that many input interpretations are much less likely than other interpretations may be responsible. In 20,000 problems, the system sees some interpretations many fewer times than others—in fact, fewer times than any interpretation in Experiment 1.

Figure 4 shows an action set from late in one run of Experiment 2. The Boolean interpretation of the classifiers is $11\#\#\#1 : 1 \Rightarrow 1000$. It is clear that

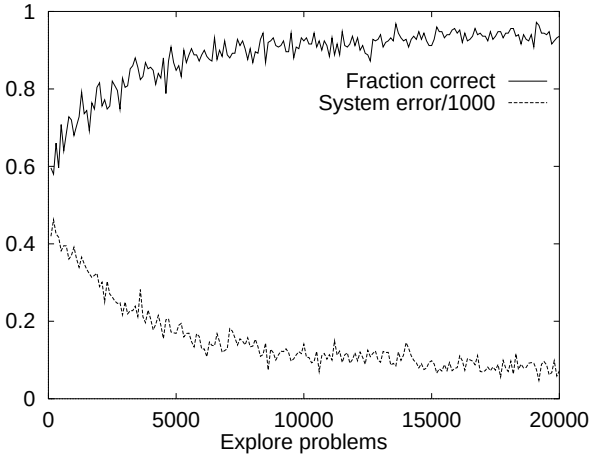


Fig. 3. Fraction correct (solid line) and system error/1000 (dashed) vs. number of explore problems in Experiment 2.

```
                                                                                                ACT PRED ERR  FITN NUM
 0. |.oOOOOOOOO|.......oOO|OOOOOOOOOO|OOOOOOOOOO|OOOOOOOOOO|.......oOo| 1 1000. .000   11.  1
 1. |..oOOOOOOO|.......oOO|OOOOOOOOOO|OOOOOOOOOO|OOOOOOOOOO|.......oOo| 1 1000. .000   11.  1
 2. |.oOOOOOOOO|.......oOO|OOOOOOOOOO|OOOOOOOOOO|OOOOOOOOOO|.......oOo| 1  667. .278   19.  2
 3. |..oOOOOOOO|.......oOO|OOOOOOOOOO|OOOOOOOOOO|OOOOOOOOOO|.......oOO| 1 1000. .000   16.  1
 4. |..oOOOOOOO|.......oOO|OOOOOOOOOO|OOOOOOOOOO|OOOOOOOOOO|.......oOO| 1 1000. .000   16.  1
 5. |..oOOOOOOO|.......oOO|OOOOOOOOOO|oOOOOOOOOO|OOOOOOOOOO|.......oOo| 1 1000. .000  207.  1
 6. |..oOOOOOOO|.......oOO|OOOOOOOOOO|OOOOOOOOOO|OOOOOOOOOO|.......oOO| 1 1000. .000   26.  1
 7. |oOOOOOOOOO|.......oOO|OOOOOOOOOO|OOOOOOOOOO|OOOOOOOOOO|.......oOO| 1  738. .262   19.  1
 8. |oOOOOOOOOO|.......oOO|OOOOOOOOOO|OOOOOOOOOO|OOOOOOOOOO|.......oOO| 1  654. .386   11.  2
 9. |..oOOOOOOO|.......oOO|OOOOOOOOOO|OOOOOOOOOO|OOOOOOOOOO|.......oOO| 1 1000. .000   32.  1
10. |oOOOOOOOOO|.......oOO|OOOOOOOOOO|OOOOOOOOOO|OOOOOOOOOO|.......oOo| 1  751. .353    0.  1
11. |..oOOOOOOO|........oO|OOOOOOOOOO|OOOOOOOOOO|OOOOOOOOOO|.......oOO| 1 1000. .000   31.  1
12. |..oOOOOOOO|.......oOO|OOOOOOOOOO|OOOOOOOOOO|OOOOOOOOOO|.......oOO| 1 1000. .000   33.  1
13. |..oOOOOOOO|.......oOO|OOOOOOOOOO|OOOOOOOOOO|OOOOOOOOOO|.......oOO| 1 1000. .000   37.  1
14. |..oOOOOOOO|.......oOO|OOOOOOOOOO|OOOOOOOOOO|OOOOOOOOOO|.......oOO| 1 1000. .000   33.  1
15. |..oOOOOOOO|.......oOO|OOOOOOOOOO|OOOOOOOOOO|OOOOOOOOOO|.......oOO| 1 1000. .000   33.  1
16. |..oOOOOOOO|.......oOO|OOOOOOOOOO|OOOOOOOOOO|OOOOOOOOOO|.......oOO| 1 1000. .000   29.  1
17. |..oOOOOOOO|.......oOO|OOOOOOOOOO|OOOOOOOOOO|OOOOOOOOOO|.......oOO| 1 1000. .000   99.  3
18. |..oOOOOOOO|.......oOO|OOOOOOOOOO|OOOOOOOOOO|OOOOOOOOOO|.......oOO| 1 1000. .000  154.  6
19. |..oOOOOOOO|.......oOO|OOOOOOOOOO|OOOOOOOOOO|OOOOOOOOOO|.......oOO| 1 1000. .000   34.  1
```

Fig. 4. An action set [A] from Experiment 2. Same notation as upper half of Figure 2.

in most of the classifiers XCSR has "detected" and represented the 0.25 and 0.75 thresholds quite accurately. One can also see that the four classifiers with non-zero error have first predicates that are a little too large.

5 Discussion

Work on XCSR can be taken in several directions. It is easy to suspect that the relative crudeness of the present mutation technique may account for the failure to reach higher performance levels, especially in Experiment 2 where smaller in-

terval predicates were needed. As performance closes in on its maximum, smaller and smaller mutation increments are called for, yet in these experiments there was no such adaptation. It is possible that adapting the increment along lines used in the *Evolutionsstrategie* [8] may be appropriate. Adaptation may also increase the speed of learning, because larger mutation increments could presumably be used at the beginning of a run.

The present test tasks—derived from a Boolean problem—while relatively complex and non-linear, are in many respects not representative of actual data inference problems. There, it is often the case that only a few out of many input variables are relevant to the decision, and the challenge is to identify them in the presence of noise and data contradiction. For this, the noise filtering techniques of [6] may be helpful. In addition, the assumption in this paper of equally probable outcomes is often violated by real data, where the prevalences are likely to be skewed. This can be investigated by adapting XCSR (and XCS) to keep the "four-way" statistics characteristic of fields like epidemiology (i.e., $\{true, false\} \times \{positive, negative\}$ instead of machine learning's *correct/incorrect*). Experiments along these lines are under way.

A large step, obviously required if XCSR and XCS are to participate in practical data inference, is to test the systems in regimes where training is done on one data set and testing occurs on another—both sets often being drawn from a single larger set. Such training/testing regimes are standard in data inference. It is vital for XCS-like systems to meet the challenge of training sets that sample the environment incompletely.

6 Conclusion

This paper has demonstrated that XCSR—and thus classifier systems—can learn to classify real-vector inputs and form accurate maximal generalizations over them. The results are another step toward full realization of the promise of Holland's classifier system idea.

References

1. Bonarini, A.: Evolutionary learning of fuzzy rules: competition and cooperation. In: W. Pedrycz (ed.), Fuzzy Modelling: Paradigms and Practice. Norwell, MA: Kluwer Academic Press (1996) 265-284.
2. Holland, J. H.: Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In: R. S. Michalski, J. G. Carbonell & T. M. Mitchell (Eds.), Machine Learning, An Artificial Intelligence Approach. Volume II. Los Altos, California: Morgan Kaufmann (1986).
3. Holmes, J. H.: Discovering risk of disease with a learning classifier system. In T. Baeck, ed., Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA97). Morgan Kaufmann, San Francisco, CA. (1998) 426-433.
4. Kovacs, T.: XCS classifier system reliably evolves accurate, complete, and minimal representations for Boolean functions. In: Roy, Chawdhry and Pant (Eds), Soft Computing in Engineering Design and Manufacturing, Springer-Verlag London (1997) 59-68.

5. Kovacs, T.: Deletion schemes for classifier systems. In: Banzhaf, W., et al, (eds.), Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99). Morgan Kaufmann: San Francisco, CA (1999) 329-336.
6. Lanzi, P. L. and Colombetti, M.: An extension to the XCS classifier system for stochastic environments. In: Banzhaf, W., et al, (eds.), Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99). Morgan Kaufmann: San Francisco, CA (1999) 353-360.
7. Lanzi, P. L. and Perrucci, A.: Extending the representation of classifier conditions, part II: from messy coding to s-expressions. In: Banzhaf, W., et al, (eds.), Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99). Morgan Kaufmann: San Francisco, CA (1999) 345-352.
8. Rechenberg, I.: Evolutionsstrategie '94. Stuttgart-Bad Cannstatt: Frommann-Holzboog (1994).
9. Saxon, S. and Barry, A.: XCS and the Monk's Problems. This volume.
10. Wilson, S. W.: ZCS: a zeroth level classifier system. Evolutionary Computation **2**(1) (1994) 1-18.
11. Wilson, S. W.: Classifier fitness based on accuracy. Evolutionary Computation **3**(2) (1995) 149-175.
12. Wilson, S. W.: Generalization in the XCS classifier system. In Proceedings of the Third Annual Genetic Programming Conference, J. Koza et al (eds.). San Francisco, CA: Morgan Kaufmann (1998) 665-674.
13. Wilson, S. W.: State of XCS classifier system research. This volume.

XCS and the Monk's Problems

Shaun Saxon[1] and Alwyn Barry[2]

[1] The Database Group,
Colston Centre, Colston Avenue
Bristol, BS1 4UH, UK.
ShaunS@databasegroup.co.uk
[2] Faculty of Computer Studies and Mathematics,
University of the West of England,
Coldharbour Lane, Bristol, BS16 1QY, UK
Alwyn.Barry@uwe.ac.uk

Abstract. It has been known for some time that Learning Classifier Systems (LCS) [15] have potential for application as Data Mining tools. Parodi and Bonelli [25] applied the Boole LCS [36] to the Lymphography data set and reported 82% classification rates. More recent work, such as GA-Miner [10] has sought to extend the application of the GA-based classification system to larger commercial data sets, introducing more complex attribute encoding techniques, static niching, and hybrid genetic operators in order to address the problems presented by large search spaces. Despite these results, the traditional LCS formulation has shown itself to be unreliable in the formation of accurate optimal generalisations, which are vital for the reduction of results to a human readable form. XCS [39,40] has been shown to be capable of generating a complete and optimally accurate mapping of a test environment [18] and therefore presents a new opportunity for the application of Learning Classifier Systems to the classification task in Data Mining. As part of a continuing research effort this paper presents some first results in the application of XCS to a particular Data Mining task. It demonstrates that XCS is able to produce a classification performance and rule set which exceeds the performance of most current Machine Learning techniques when applied to the Monk's problems [34].

1 Introduction

With the increase in the computerization of business, and in particular retail business, large amounts of business data is gathered and stored. Until recently the benefit of attempting to extract useful information from this data only marginally outweighed the cost of maintaining this great a volume of data because the then current data analysis techniques were ill-suited to coping with large and noisy data and were not able to extract information of sufficient quality and quantity. With the reduction in the cost of storing data, there is now pressure to effectively utilise the data for the benefit of the data producers. For example, improvements in the performance of data analysis techniques would be

P.L. Lanzi, W. Stolzmann, and S.W. Wilson (Eds.): LCS '99, LNAI 1813, pp. 223–242, 2000.

beneficial for a large retail chain with computerized stock control and sales systems by enabling it to analyze its large databases to predict customer behavior. Furthermore, the recent popularity of loyalty cards generates customer specific transaction data, which increases the potential for forming more accurate models of customer behavior.

Current techniques for discovering patterns in data, termed *Data Mining*, have various limitations when considered against criteria for an effective data mining method. These criteria include the ability to cope with large and noisy data, being able to produce simple, meaningful and useful results that can be understood by a human user, and the use of a practical amount of computational resources. Evolutionary computation techniques, including those based on genetic algorithms (GA), are able to explore a large search space to find an optimal solution effectively due to their implicitly parallel nature. They are also [largely] undirected and therefore require minimum input of domain knowledge. These robust techniques have therefore been considered in order to counter the limitations of current data mining techniques and have already been successfully applied in a commercial environment [10].

Whilst holding much promise, there are also known difficulties with the application of GA based techniques to the task of Data Mining. Traditional Learning Classifier Systems [15] which can be used for this purpose have a limited ability to generalize rules. A traditional LCS provides generalization pressure by the use of specificity measures that are factored into the action selection process [12,28]. This method does not produce optimal generalization, tending to over generalize in the early stages of learning, only to over specialize as the GA continues to operate [28,37]. Wilson's XCS [39,40] introduced niching techniques and an accuracy-based fitness which enables XCS to produce accurate optimally general co-operative rule sets. This ability to generalize lays the foundations of a concept builder of the kind required for data mining.

In investigating the performance of a Data Mining algorithm a comparison with the performance of existing techniques is useful. The Monk's Problem is one of only a few classification problems that has been used to generate a substantial comparative study [34]. As a first investigation of the abilities of XCS as a Data Mining tool the use of the Monk's Problem is appropriate, providing strong evidence of the qualities of XCS classification in comparison to other machine learning techniques. This paper details these investigations and seeks to identify areas of strength and weakness in XCS for further ongoing research into the application of XCS to the Data Mining of large commercial data sets.

2 Background

This potential for extraction of useful trends and indicators from consumer data can be realized providing that the problems of increased database size can be overcome by improved data analysis and reduction techniques which are constituents of the larger process commonly known as *Knowledge Discovery in Databases* (KDD) [9]. This term is used to describe all of the tasks required to

extract useful information from raw databases and to distinguish the task of automatically discovering patterns in pre-processed data, Data Mining, from data extraction, data cleansing, data reduction, and interpretation and application of results from data mining. Within the context of Data Mining, classification is the process of using a set of pre-classified instances to assign a new instance to one of n classes. A finite vector of attribute values defines an instance. The training set is a set of instances that have been assigned to one of the n classes.

Current data mining techniques [31] have been developed from methods originating in the fields of statistical modeling, neural networks and machine learning. Statistical techniques [22,24] exist for classification (e.g. CART [7] and CHAID [17]), regression (linear, quadratic and logistic) and clustering tasks but are based on assumptions about the size and probability distributions of data, which limits their effectiveness on real data, and can only be applied with the intervention of a statistician. Advanced machine learning techniques have been applied to data mining tasks with varying degrees of success. Decision tree (ID3, C4.5) and production rule (AQ15, CN2) learners produce fairly accurate but complex representations of patterns in the data although these representations may be simplified by using heuristic search. Generalization occurs when grouping attribute values and when decision trees are pruned. These techniques often require considerable computation and can be applied to limited data types therefore much effort needs to be put into data reduction prior to application. For further information on these and similar techniques, [23] provides an accessible and detailed review chapter. Artificial neural networks have been used widely with considerable success for Data Mining tasks [20,30] and are implemented in many commercial data mining software packages. They are able to learn complex models of data and provide good generalization but the interpretation and application of the neural network models are difficult tasks for the human user.

In order to counter the problems with the current data mining techniques there have been projects which sought to apply evolutionary computing techniques to various areas within KDD. Evolutionary techniques have primarily concentrated upon the KDD task of Feature Selection (e.g. [26]), and advances have been made in the modification of GA techniques to develop and maintain both single (GABL, GABIL, [33]) and multiple (COGIN, [13,14]) target concepts. Recent work within Feature Selection has moved onto the application of Genetic Programming with pruning techniques introduced to produce human readable results (eg. [27]). Applying evolutionary computing to Data Mining is a more complex task, directly tackling the learning problem rather than meta-learning. Early work by Parodi and Bonelli [25] with the Animat LCS [36] illustrated the potential of LCS for Data Mining within a simple data mining problem, and two significant research projects have sought to extend these results to commercial data sets (see REGAL [11] and GA-Miner [10]). Significantly, both of these projects introduce high level attribute and attribute vector encoding with directed mutation and crossover operators in order to cope with the large search space presented by commercial data sets. Problems with these approaches remain, however. In particular, with GA-Miner for example, the establishment

and maintenance of multiple co-operative rules in the presence of the GA has been tackled using static techniques that can require domain knowledge, and the methods applied to obtain generalization are primitive. These problems are common to all traditional Michigan style LCS implementations.

The XCS Classifier System [39,40] represents a major step forward for the Michigan style LCS. Deriving from Wilson's Animat [36] and ZCS [38], the XCS simplifies the Holland style classifier system in an attempt to produce a LCS whose operation and dynamics can be readily understood and predicted. It borrows strength update mechanisms from Reinforcement Learning using a modified Widrow-Hoff update mechanism [35] to provide a multi-parameter strength regime that more accurately reflects the different roles of strength within action selection and the GA. XCS also re-introduces GA mechanisms recommended by Booker [5] which provide a niching facility to allow co-operative sets of rules to co-exist within a population whilst encouraging competing rules to converge on optimum rule attributes within a niche. By using accuracy as the GA selection criteria, XCS is the first LCS to be able to claim to *reliably generate the most general accurate classifiers* — the so-called *Generalization Hypothesis* [40].

Further work by Kovacs [19] has demonstrated that the provision of further operators can sufficiently focus the population once exploration is complete to reliably produce a minimum rule set consisting of the most general accurate classifiers, and has led to the *Optimality Hypothesis* which suggests that using these operators the rule set generated will be the optimal rule set for a given static problem.

It would appear, therefore, that the application of a possibly modified form of XCS to Data Mining has the potential to address the significant problems which remain unsolved within previous Michigan style LCS based data miners. As part of a continuing project to develop an effective Data Mining tool using Evolutionary Computing techniques, an investigation into the potential of XCS is underway. Learning from previous efforts, it is expected that XCS will need to be modified to accommodate more complex attribute and attribute vector encoding and that hybrid genetic operators will be applied. Before venturing further in these investigations, however, it is important to identify a baseline performance of XCS in order to ascertain its appropriateness for the task of Data Mining. It was decided that XCS should be applied to a small data set in order to demonstrate the effectiveness of its generalization and niching mechanisms within the domain of Data Mining.

3 Experimental Investigation

3.1 Test Environment

For an initial investigation of the suitability of XCS to the task of Data Mining the use of a 'standard' test data set was considered important. Unfortunately within the field of Data Mining no such test sets exist, with the best attempt at a standard test set being the data sets used with the Statlog project [21] which

due to their complexity and quantity are inappropriate for use at this stage. Within the Machine Learning community a number of data sets have a sufficient usage to make them a 'de facto' standard. The Monk's test set, held on the UCI Repository [3] was selected as an appropriate test set because of its small size with high internal complexity, and the extensive set of published comparative results available with the test data.

The Monk's problems are binary classification problems of differing complexity based on a pre-classified artificial robot data set created to empirically compare learning classifiers [34]. There are three problems each of which involves learning target concepts described by conjunctions of logical relations on attributes with their values. Training and test data sets are created by partitioning the data set containing all possible instances and pre-classifying each instance according to the problem. 5% noise is added to the training data for the third Monk's problem. Objective assessment of the performance of XCS is feasible because the data has been pre-classified according to explicit and well-defined target concepts which enables a direct comparison to be made with the performance of other Data Mining techniques on these problems.

The three test sets provided with the Monk's problem are individually significant to the evaluation of XCS for Data Mining. The first test set is solved using a simple boolean expression which, whilst not directly representable in a single XCS classifier condition, can be optimally represented (including the corresponding negative cases) using the standard XCS ternary conditions by small set of 12 cooperating classifiers. Four of the six attributes must be removed from consideration using generalization, thereby providing a test of the generalization capabilities of XCS. The second experiment requires a complex set of disjunctions of conjunctions to be fully represented, and presents a considerable challenge to most Machine Learning techniques. This experiment will test the ability of XCS to maintain a population of many cooperative classifiers in order to represent the classification accurately. In real data it is difficult to clean the data completely, and the use of accuracy as the basis of XCS learning could be compromised by noisy data. Thus, the Monk's 3 data set will illustrate the ability of XCS to cope with typical amounts of noise found in data after data cleansing without hindering the learning operation.

3.2 The Learning System

A XCS implementation conforming to the XCS used within [40,18] was obtained [2]. In all experiments the parameterization of the XCS was set as follows: $p_1 = 10.0$, $\epsilon_1 = 0.01$, $F_1 = 0.01$, $\beta = 0.2$, $\theta = 25$, $\epsilon_0 = 0.01$, $\alpha = 0.1$, $\chi = 0.8$, $\mu = 0.04$, $\phi = 0.5$, $P_{\#} = 0.33$ (see Appendix B of [18] for a parameter glossary). These parameter settings were taken from the settings used by [39], and were considered appropriate due to the similarity of the Monk's search problem to the Boolean Multiplexor problems investigated by Wilson. During our investigations alternative parameter settings were investigated and not adopted. The message length, condition length, and maximum population size were set according to the needs of each experiment.

The Monk's problems are derived from a data set of 432 instances, each consisting of six categorical integer valued attributes (see [34]). Initially the input encoding was devised as a binary string with each attribute represented by a minimal corresponding binary form concatenated to produce the input bit vector. For example, attribute 1 is a three valued attribute (values 1, 2, and 3), represented in this encoding by two bits with value 1 coded as `00`, value 2 coded as `01`, and value 3 coded as `10`. This gives a string length of ten to represent all six attributes. The minimum string length required to represent the entire attribute space of 432 is 9 ($2^9 = 512$). Thus, this representation produces a small degree of redundancy. The encoding assumes that the attribute values are ordered in some way, even arbitrarily. This assumption is satisfied in the Monk's data because the categorical attribute values are represented as successive integers starting at one. The output message is a single binary bit representing the predicted class for instances matching the classifier condition.

A training and test environment was constructed to present the data instances sampled at random from one of the standard Monk's training sets (see App. A) on each iteration of the XCS. Upon receiving the chosen output message from the XCS, the environment returns a maximum reward if the output message is the same as the class of the instance used to provide the input message and a minimum reward is returned otherwise. To test the learnt classifier population, the test set is presented to the XCS after a predetermined number of exploration (learning) trials, set as appropriate for each experiment, and the performance of XCS captured during this phase with classifier induction and strength modification disabled. The results presented are an average of 10 runs of XCS on the training set and with constant parameterization. The results of this test phase are presented only if the performance of the XCS on the test data set was different from that achieved within the training set at the end of the learning phase. Coverage Tables, used in the Monk's problems study to compare the completeness of concept learning, were also captured at the end of the learning phase (see App. B).

The graphs show three measures of XCS performance: system prediction, system error and macro-classifier population size. System prediction indicates the proportion of exploit iterations, out of the previous 50 exploitation iterations, in which XCS correctly classifies a data instance. The system error is a moving average of the error in the prediction of the chosen Action Set [A] (the set of classifiers chosen to perform the action in an iteration), again calculated over the previous 50 exploitation iterations. The population size is the proportion of the number of macro-classifiers in the population to the total number of micro-classifiers that the population may hold. For an XCS that successfully learns an optimal set of classifiers the prediction rises steeply to 1.0 and stays at that level. Occasional fluctuations just below this maximum may occur when the GA deletes a classifier during search for the optimal population (termed [O]; [18]). Macro-classifier population size will typically increase to a peak at around three-quarters of the total population size as the XCS explores as many classifiers as possible. The pressure to obtain optimally general classifiers drives the population down

as the 'good' classifiers increase in numerosity and the 'poor' ones are deleted. Eventually the population will stabilize when the XCS has found the optimal set. A sign of a weak performance within XCS (when it doesn't explore a high enough proportion of the population and hence is unable to converge on the optimal solution) is when the population size oscillates around a relatively high proportion (> 0.3) without further reduction. Note that the proportion of macro-classifiers in the stable population depends on the population limit set as a parameter and that this limit is set using knowledge of the domain.

3.3 The Monk's 1 Problem Test

The Monk's 1 problem test is a relatively simple test designed to demonstrate that a new classification algorithm is capable of simple concept learning. As such, the application of this test set to XCS should confirm or deny the basic hypothesis that XCS is an appropriate basis for the development of a Data Mining tool. From results reported for the testing of other Machine Learning algorithms on this test set [34] it would be expected that XCS should be able to demonstrate accurate classification (demonstrated by accurate prediction) within a relatively short learning period. In addition, it is desirable that XCS be able to identify the optimal classifier set for this problem. The optimal set of twelve classifiers for the Monk's 1 problem using binary encoding is:

```
#00#######→1 ######0000→1
#######1#1→1 ######1#1#→1
#1####0#1#→0 ##1###0#1#→0
#1####1#0#→0 ##1###1#0#→0
#1#####0#1→0 ##1####0#1→0
#1#####1#0→0 ##1####1#0→0
```

Note that XCS retains accurately wrong classifiers therefore each of these classifiers will have an associated classifier with the same condition, the opposite action, and a prediction equal to the minimum reward value.

The condition size and message size for the binary encoding used with this test were set at 10. The population size limit was chosen from a consideration of the predicted size of [O]. If we want to find all the optimal classifiers each with an average numerosity of fifteen then the minimum population size should be $24 \times 15 = 360$ microclassifiers. Thus, for this experiment a population size of 400 was deemed adequate.

Results. The performance measures averaged over 10 runs of XCS on the first Monk's training set indicate that XCS had learned the target concepts after only $10,000$ exploitations. The performance curve in Fig. 1 indicates that XCS has successfully established a population that correctly classifies the data set. An analysis of the populations resulting from the individual runs revealed that all of the populations had discovered [O] with the members of [O] obtaining high numerosity, accurate prediction and fitness. In three of the final populations one

of the optimally general classifiers did not have a relatively high numerosity or experience but was still perfectly accurate and had a high fitness, demonstrating that this final member of [O] had been established relatively late within the run in these cases. On average each final population held 80 macroclassifiers, 24 of which made up [O]. All of the remaining classifiers had low numerosity (< 5) and small experience, indicating that they were the product of continued but unnecessary GA search. These results confirm the hypothesis that XCS is able to induce and establish the accurate and optimally general rule set from a simple data set of pre-classified data. The performance of XCS was comparable with the performance of neural networks and production rule classifiers (CN2 and those based on AQ-15), and exceeded that of all the ID3 based decision tree classifiers reported in [34].

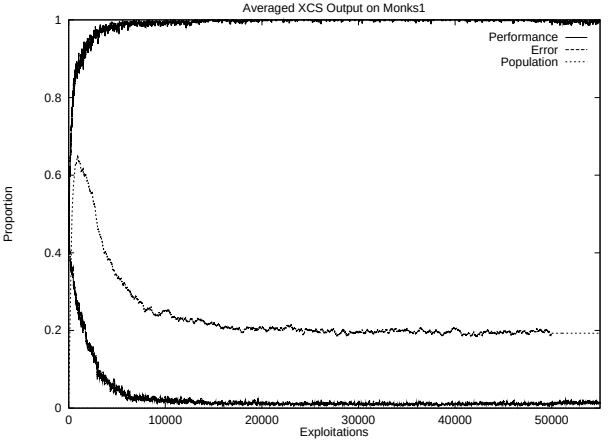


Fig. 1. XCS performance during training using the Monk's 1 Test data set, averaged over 10 runs with parameterization as described in Sect. 3.2. The performance on the test set remained at the same level for the final 5,000 iterations.

3.4 The Monk's 2 Problem Test

The Monk's 2 data set represents a much more complex relationship than that seen in Monk's 1. In the comparison of performance [34] only the neural network based classifiers and AQ17DCI obtained 100% classification performance, with most decision tree classifiers obtaining 65–70% correct classification. The complexity of the task is indicated by the fact that in order to represent the concepts optimally within the ternary condition encoding of XCS a set of 104 binary-encoded classifiers is required for positive classification and 100 classifiers for negative classification. It must be noted, however, that the target concept in

Monk's 2 Problem cannot be simply described when using disjunctions of conjunctions of attribute-value pairs and therefore this problem is artificially hard for XCS using the standard condition representation.

The XCS message encoding used within this test was initially kept the same as that used for the Monk's 1 test. Knowing the size of the solution space, however, the population size was increased to 1600 classifiers. This is potentially small for the predicted number of target classifiers but investigations into larger population size did not show improvements sufficient to justify the extra computational resources required.

Results. It can be seen from the XCS performance in Fig. 2 that it struggles to find the optimal set of classifiers. The system prediction converges slowly towards 1.0 but still averages only about 0.98 at 50,000 exploitations. The proportion of macro-classifiers in the population peaks at only 0.6 and then falls to ≈ 0.35 over the whole 50,000 exploitations, indicating that the XCS has not explored enough of the solution space to find all the optimally general classifiers. System error stabilizes at about 0.05 which is significantly larger than the accuracy cutoff of 0.01 supporting the suggestion that insufficient search has been performed. Inspection of the final populations and the corresponding coverage tables revealed that XCS had only learnt a small proportion of the optimal rule set in each of the runs if only the high numerosity classifiers are considered. A larger proportion of the optimal rule set was present if 'weak' classifiers were considered.

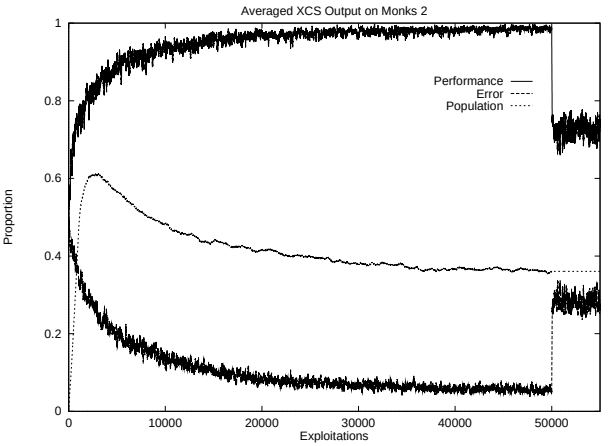


Fig. 2. XCS performance in the Monk's 2 Test data set using a Binary Encoding, averaged over 10 runs, population size 1600.

From these results it was hypothesized that XCS was unable to find suitable classifiers due to the deceptive nature of the encoding used. To test this hypothesis a second encoding type was introduced - an enumeration encoding. The enumeration encoding maps an attribute with n possible values to a binary string of length n. The attribute value is mapped to an integer ranking, i, by setting the i^{th} lowest bit to 1 and all the others to 0. This gives a string length of 17 for the Monk's Problems and a condition search space of 3^{17}. In fact, using this encoding in XCS allows the same attribute vector to be encoded in more than one way. For example, the attribute-value pair $(A1, 3)$ can be represented using the ternary strings, `100`, `1#0`, `10#`, `1##`, and `#00` because the attribute-value pair $(A1, 3)$ can only ever be encoded in a binary string as `100` in an input message. Also, it is more convenient to express inequalities such as $A1 \neq 1$, which in an enumeration encoding has a single representation `##0` whereas in a binary-valued encoding two strings are required: `1#` and `01`. This reduces the number of classifiers required from 204 for the binary encoding to 42 — 15 and 27 classifiers for positive and negative classification respectively. It was expected that the greater flexibility of the enumeration encoding in representing attribute-value relations will enable XCS to optimally classify for the Monk's 2 Problem and to find the optimal rule set.

The encoding was adjusted within the test and trial environment and the experiment was repeated with the new encoding. The population size was maintained at 1600 micro-classifiers in order to allow a direct comparison with the results from the binary encoding.

The averaged results of the 10 runs are depicted in Fig. 3. With the enumeration encoding the system prediction reaches its maximum of 1.0 at around $25,000$ exploitations with the system error correspondingly dropping to around 0.0, as expected. This demonstrates that XCS was able to identify a correct classification for the test data set. An examination of the final populations revealed that [O] was not completely formed, with approximately 30 of the 42 members of [O] present and the remainder partially covered by two or more classifiers with conditions that remained too specific. This was resulted in a less than 100% classification rate on the test data set.

These results confirmed the hypothesis that the representation of attribute values in a classifier is a significant factor in XCS performance [6]. Clearly the performance using the enumeration encoding is as least as good as the Machine Learning algorithms reported in the Monk's problem study. The results reported for the application of Back-Prop neural networks, although marginally better, could not generate generalized concepts that were open to examination, which is a key requirement of an effective Data Mining technique.

3.5 Monk's 3 Problem Test

The target concept of the Monk's 3 Problem is of similar complexity to Monk's 1 but it incorporates noise by including 6 misclassified examples ($\approx 5\%$) in the training set. This test data set could pose significant problems for XCS, as the ability to maintain accurate classifiers will be dependent upon the value of the

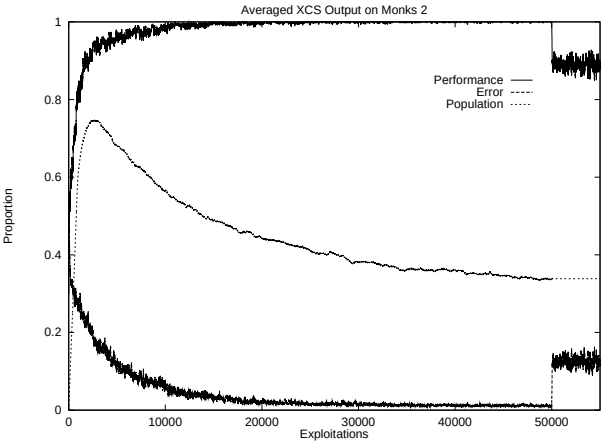


Fig. 3. XCS performance in the Monk's 2 Test data set using the Enumeration Encoding (averaged over 10 runs, population size = 1600), demonstrating the ability of XCS to perform optimally given a suitable classifier encoding. System Prediction dropped to ≈ 0.92 on the test data set.

parameters of the accuracy function. Data that contains too much noise could be preventing XCS from forming accurate generalizations and thereby hinder the search process of the GA within XCS. Alternatively, where a lower level of noise is present XCS may over-specify ('over-fit' to the training data), hindering performance on the test set. It is hypothesized that the limited extent of noise added to the Monk's 3 data set will not hinder the ability of XCS to establish the best [O] available from the training set, although XCS may over-fit to the training set.

This experiment was run using 10 runs in each case of both the Binary encoding and the Enumeration encoding, with the condition and message size set appropriately for each encoding. The population size was set to 400 because of the lower level of complexity of this problem, requiring 22 classifiers to map the [noiseless] problem completely with optimal generality when using the Binary encoding.

Results. No difference in performance was found as a result of differences in the encoding form used. The graph in Fig. 4 shows the results of running XCS using the Binary encoding on the Monk's 3 training and test data sets. Training XCS gives a classification performance of ≈ 0.95 and a relatively large system error. This is a consequence of the noise in the training set, with a maximum achievable performance of 0.95 on average. When XCS is switched to run on the test data set the classification performance improves and the system error drops. An examination of the populations at the end of each run revealed that XCS had successfully established a set of high numerosity optimally general classifiers

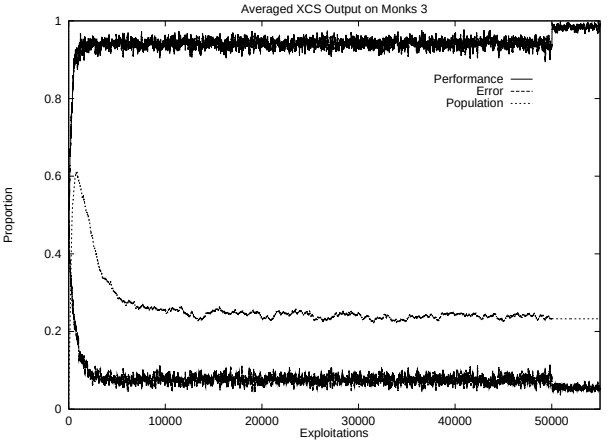


Fig. 4. XCS performance in the Monk's 3 Test data set (averaged over 10 runs, population size was 400). The last 2000 exploration iterations were conducted without the induction algorithms using the Test data set. The optimal expected performance was 95% due to the noisy training set.

close to [O] (which will never be achieved because of the presence of noise). We hypothesize that the improvement in performance during the test phase is due to the presence of other classifiers in the population as a result of continued GA exploration which cover the over generality due to noise of the high numerosity classifiers.

The results achieved by XCS are comparable with those obtained by Neural Network classifiers in the Monk's problem study, and marginally better than the Decision Tree classifiers. They were not as good as the AQ15 based Production Rule classifiers which typically achieved 100% performance in the test set due to the larger (normally redundant) coverage of the classification space which they maintain. These results are encouraging, although staged experiments introducing graduated amounts of noise are required to fully evaluate the ability of XCS to cope with noisy data.

4 Discussion

The performance of XCS within the Monk's problems is very encouraging. Comparisons with existing Machine Learning techniques have already been made in Sect. 3, which demonstrate that the performance of XCS is at least as good as, and in many cases better than classifiers based on other Machine Learning techniques. Furthermore, the ability of XCS to determine the compact rule representation provided by the optimally general classifier set in these tests, and potentially present these classifiers in a human readable form, is a clear advantage.

In the Monk's 2 Problem the XCS did not form a completely generalized mapping of the target concepts. In these populations a number of classifiers existed which were marginally over specific, indicating that more GA recombination was required in order to obtain full generalization. Kovacs [19] has shown that an additional deletion operator, based on a mechanism termed *condensation* [40], reduces a population to the optimal set in single-step problems. It is likely that a deletion operator of this type could be employed on all classification tasks to enable XCS to reliably produce a minimal set of optimal rules even when learning is halted prior to establishing [O].

In data mining applications the target concepts are rarely known, although sometimes estimated by a domain expert, so we have resisted the temptation to introduce any encoding that is tailored to the problem. At least one of the production rule algorithms compared in the Monk's problem study [1] produces 100% accuracy on both the Monk's 2 test sets. This is wholly due to the fact that the algorithm had been set to learn concepts of based on the number of attributes, with the same specific value which enables the target concepts to be represented by only two production rules:

1. `if (equals (number of attributes with value x, 2)) then 1`

2. `if (not-Equals (number of attributes with value x, 2)) then 0`

Without specific knowledge of the target concepts the classifier system cannot be setup to use rules such as these so more general representations are needed. For these artificial problems the target concepts are known and therefore an encoding that is concept-specific would theoretically perform better on this problem than the more general ternary encoding used.

Theoretically, the enumeration encoding should be used for all of the Monk's problems because all of the attributes are categorically valued and the binary encoding becomes unwieldy when relations between attributes and logical negations need to be represented. Practically, because the enumeration encoding is longer than the binary for the Monk's problem and there are many more redundant classifiers than when learning relatively simple target concepts such as in the Monk's 1 problem, the compactness of the binary encoding makes it just as, or more, effective than the enumeration encoding. These encodings are suitable only for discretely-valued attributes so future work will include investigation into encodings for continuously valued attributes and alternatives encodings for discrete attributes. It is recognized that GA-Miner and other LCS based Data Mining tools which have been developed have employed high level attribute representations and more complex classifier condition formulations which are advantageous in many circumstances.

The difficulty in learning the concepts for the Monk's 2 training set using the Binary encoding require further investigation. Although intuitively XCS would be expected to increase its level of exploration when correctly classifying rules cannot be found, the degree of exploration used by XCS was actually reduced. This would suggest that the GA in XCS was unable to recombine classifiers in a

way that produced significant improvements in their accuracy, so preventing a clear search direction emerging which would be exploited by the GA. This may indicate a potential problem within XCS, or it may be the case that the binary encoding simply presented a deceptive landscape.

An important criterion for an effective data mining technique is the ability to cope with noisy data. The large number of samples taken from only 124 training examples presented the danger of over-fitting to the training data and succumbing to bias within the training data. The tension between obtaining accurate classification and achieving maximally general classifiers in the operation of XCS appeared to prevent this. As yet, it is difficult to ascertain the proportion of noise (and/or bias) in the data which XCS is able to absorb before performance degrades significantly.

5 Future Work

The Monk's problems are based on a data set with six attributes each having no more than four values making a total of 432 possible attribute vectors. A typical commercial data set that has been prepared for analysis will have tens or hundreds of ordinal and nominal, continuous and discrete attributes and will have a dimensionality at least two or three orders higher than the Monk's data sets. The results presented in this paper have demonstrated that XCS classifies the Monk's data sets more accurately than many current classifier systems, and at least as well as those systems based on artificial neural networks,. In order to establish the effectiveness of XCS and GA-based classifier systems as Data Mining techniques, then the performance of XCS on the large and noisy commercial data must be investigated.

XCS employs the ternary alphabet which is used in the canonical LCS and which grew out of the binary encodings traditionally used in GAs [4]. The results in this paper demonstrated that even when restricted to this alphabet, changing the encoding causes an observable change in the performance of XCS and is itself affected by the nature of the concepts to be learnt. An effective classifier encoding must be able to compactly and accurately represent patterns in data, of which there is typically little prior information, whilst keeping disruptions of the generalization mechanisms of XCS to the minimum. There has been much work [8,16,29,32] on representing discrete and continuous values in a GA but these representations are not always adequate for a classifier. For example, standard binary string representations of many-valued discrete attributes will produce prohibitively long and inefficient bit strings if there are several of these attributes that are sparsely valued. Coping with the uncertainty of noisy data will require fuzzy or windowed representations [6]. Mixing representations in a single classifier has implications for the crossover operators used by XCS and thus the correct operation of XCS niching and generalization. Thus, any new or modified representations will necessitate creating original operators and investigating their effect on XCS performance.

Establishing a set of effective encodings will enable XCS to be tested over a wider range of test data sets and allow a more detailed comparison of its performance against other classifier systems.

The performance of XCS on the data sets used to compare Machine Learning, statistical and Artificial Neural Network techniques in the Statlog project [21] would provide a good measure of XCS's relative effectiveness as a classifying system. Several of the data sets used within the Statlog project were taken from real commercial sources and are characteristically large and noisy and would therefore be appropriate performance reference points. For effective comparison between Data Mining investigations within the LCS community it is vital that these common data sets are adopted.

XCS has the ability to identify the attributes that have a significant effect on the classification performance by completely generalizing the contribution of the less important attributes. There will be a practical limit on the number of attributes that XCS can handle which will be exceeded by many commercial data sets so it is important to apply suitable feature selection techniques. One such technique, taken from statistical modelling, is to increment or decrement the number of attributes depending on both the classification performance of the preceding model generated by the data miner and the relative value of each particular attribute in this model. Statistics on the distribution of attribute values for each class are used to measure of the importance of an attribute. XCS provides an excellent platform for integrating these mechanisms dynamically so those attributes can be added and subtracted in a single run of XCS.

6 Conclusions

In the application of the Monk's problems to XCS is has been shown that XCS was able to perform at least as well as traditional Machine Learning techniques in both simple and complex classification tasks, and in the presence or absence of small proportions of noise within the data. In some cases XCS was seen to perform better than Machine Learning techniques which are commonly found in commercial data mining applications. XCS was also able to produce and maintain an easily identifiable accurately general set of classifiers representing the concepts within the data sets. If combined with software to convert the ternary classifier representation into a standard production rule format, XCS will generate a model that can be readily understood and re-applied by potential clients of a data mining application.

Whilst encouraging, the Monk's problems are limited in their correlation to data sets used within a commercial Data Mining environment. Work is underway on expanding the application of XCS to graded real world data sets that will allow the XCS mechanism to be scaled to cope with large or sparse data sets. In particular, the encoding problems seen within the Monk's 2 problem indicate a need to find appropriate attribute representation, and appropriate condition representations. Large data will also require more input from hybrid genetic and non-genetic operators that will take into account knowledge about the distribu-

tion of attributes within the data set. It is also possible that methods akin to those used to establish linkage within genotypes may bear fruit, and this avenue will be investigated.

If GA-based Data Mining is to compete with the statistical modelling, artificial neural networks and decision-tree learners that are well-established Data Mining techniques then it must at least be able to offer the ability for supervised classification of large and noisy data over many different domains. Of the previous work in GA-based techniques for Data Mining outlined earlier only REGAL and GA-Miner have been used successfully to mine commercial data by using hybrid techniques to solve some of the practical problems of mining commercial data but these use domain knowledge and primitive generalisation operators. XCS has the potential to extend the capabilities of GA-based data mining systems over and above other Data Mining techniques because it has the ability to produce the optimally general descriptions of patterns in any data. This, together with the robust nature of GAs, will mean that a GA-based data mining system with XCS capabilities will have an accuracy comparable with that of Neural Networks, and will provide an explicit human-readable description of the patterns in the data equivalent to those produced by decision–tree learners.

Acknowledgements. Project funded by UK Department of Trade and Industry under the auspices of the Teaching Company Directorate. The authors wish to acknowledge the advice and discussions with Stewart Wilson and Tim Kovacs in the development and testing of the XCS implementation. They also acknowledge the help, support, and resources given by The Database Group and Andrew Greenyer, in the conduct of this project.

References

1. J. Bala and E. Bloedorn. Applying various AQ programs to the MONK's problems: Results and brief descriptions of the methods. In *The Monk's problems: A Performance Comparison of Different Learning Algorithms*. 1991.
2. Alwyn Barry. The XCS classifier system. Technical report, Faculty of Computer Science and Mathematics, University of the West of England, 1998.
3. C. Blake, E. Keogh, and C. J. Merz. UCI Repository of Machine Learning Databases. http://www.ics.uci.edu/˜mlearn/MLRepository.html, 1998. Irvine, CA: University of California, Dept. of Information and Computer Science.
4. L. B. Booker, D. E. Goldberg, and J. H. Holland. Classifier Systems and Genetic Algorithms. In *Artificial Intelligence*, volume 40, pages 235–282. September 1989.
5. Lashon B. Booker. Triggered rule discovery in classifier systems. In J. David Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 265–274, George Mason University, June 1989. Morgan Kaufmann.
6. Lashon B. Booker. Representing attribute-based concepts in a classifier system. In Gregory J. E. Rawlings, editor, *Foundations of Genetic Algorithms*, pages 115–127. Morgan Kaufmann, San Mateo, 1991.
7. L. Brieman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, 1984.

8. Lawrence Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.
9. U. M. Fayyad, G. Piatetsky-Shapiro, and Padraig Smyth. From data mining to knowledge discovery: An overview. In *Advances in Knowledge Discovery and Data Mining*. 1996.
10. I. W. Flockhart. GA-miner: Parallel data mining and hierarchical genetic algorithms. Technical Report EPCC-AIKMS-GA-MINER-REPORT 1.0, University of Edinburgh, 1995.
11. A. Giordana and F. Neri. Search-intensive concept induction. *Evolutionary Computation Journal*, 3(4):375–416, 1996.
12. David E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley, Reading, MA, 1989.
13. David P. Greene and Stephen F. Smith. Competition-based induction of decision models from examples. *Machine Learning*, 13:229–257, 1993.
14. David P. Greene and Stephen F. Smith. Using coverage as a model-building constraint in learning classifier systems. *Evolutionary Computation*, 2(1), 1994.
15. John H. Holland. Escaping brittleness: the possibilities of general purpose algorithms applied to parallel rule-based systems. In Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell, editors, *Machine Learning, an Artificial Intelligence approach*, volume 2, pages 593–623. Morgan Kaufmann, San Mateo, California, 1986.
16. Cezary Z. Janikow and Zbigniew Michalewicz. An experimental comparison of binary and floating point representations in genetic algorithms. In Lashon B. Booker and Richard K. Belew, editors, *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 31–36, San Diego, CA, July 1991. Morgan Kaufmann.
17. G. V. Kass. An exploratory technique for investigating large quantities of categorical data. *Applied Statistics*, 29:119–127, 1980.
18. Tim Kovacs. Evolving optimal populations with XCS classifier systems. Technical Report CS-96-17 and CSRP-96-17, also Master's thesis, School of Computer Science, University of Birmingham, UK, 1996.
19. Tim Kovacs. XCS classifier system reliably evolves accurate, complete and minimal representations for boolean functions. In Roy, Chawdhry, and Pant, editors, *Soft Computing in Engineering Design and Manufacturing*, pages 59–68. Springer-Verlag, London, 1997.
20. H. Lu, R. Setiono, and H. Liu. NeuroRule: A connectionist approach to data mining. In Umeshwar Dayal, Peter M. D. Gray, and Shojiro Nishio, editors, *VLDB '95: Proceedings of the 21st International Conference on Very Large Data Bases, Zurich, Switzerland, Sept. 11–15, 1995*, pages 478–489, Los Altos, CA 94022, USA, 1995. Morgan Kaufmann.
21. D. Michie, D. J. Spiegelhalter, and C. C. Taylor, editors. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, 1994.
22. J. M. O. Mitchell. Classical statistical methods. In D. Michie, D. J. Spiegelhalter, and C. C. Taylor, editors, *Machine Learning, Neural and Statistical Classification*, pages 17–28. Ellis Horwood, 1994.
23. Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
24. R. Molina, N. Perez de al Blanca, and C. C. Taylor. Modern statistical techniques. In D Michie, D J Spiegelhalter, and C C Taylor, editors, *Machine Learning, Neural and Statistical Classification*, pages 29–49. Ellis-Horwood, 1994.
25. A. Parodi and P. Bonelli. The animat and the physician. In J. A. Meyer and S. W. Wilson, editors, *Proceedings of the First International Conference on Simulation of Adaptive Behavior – From Animals to Animats I (SAB-90)*. MIT Press, 1990.

26. W. F. Punch, E. D. Goodman, Min Pei, Lai Chia-Shun, P. Hoyland, and R. Enbody. Further research on feature selection and classification using genetic algorithms. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 557–564, 1993.
27. M. L. Raymer, W. F. Punch, E. D. Goodman, and L. A. Kuhn. Genetic programming for improved data mining: An application to the biochemistry of protein interactions. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 375–380, Stanford University, CA, USA, 28–31 July 1996. MIT Press. GP-96 Also available as TR GARAGe96-04-01.
28. Rick L. Riolo. Bucket brigade performance: Ii. default hierarchies. In John J. Grefenstette, editor, *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms (ICGA-2)*, pages 196–201. New Jersey: Lawrence Erlbaum Associates, 1987.
29. S. Ronald. Robust encodings in genetic algorithms: A survey of encoding issues. In *Proceedings of the IEEE Conference on Evolutionary Computation*, pages 43–48, 1997.
30. R. Rowher, M. Wynne-Jones, and F. Wysotzki. Neural Networks. In D. Michie, D. J. Spiegelhalter, and C. C. Taylor, editors, *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, 1994.
31. Shaun Saxon. Data mining techniques. Technical report, Faculty of Computer Science and Mathematics, University of the West of England, 1998.
32. N. Schraundolph and R. Belew. Dynamic parameter encoding for genetic algorithms. Technical Report CS-90-175, University of California, 1992.
33. W. M. Spears and K. A. DeJong. Using genetic algorithms for supervised concept learning. In N. G. Bourbakis, editor, *Artficial Intelligence Methods and Applications*. World Scientific, 1992.
34. S. B. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. De Jong, S. Džeroski, S. E. Fahlman, D. Fisher, R. Hamann, K. Kaufman, S. Keller, I. Kononenko, J. Kreuziger, R. S. Michalski, T. Mitchell, P. Pachowicz, Y. Reich, H. Vafaie, W. Van de Welde, W. Wenzel, J. Wnek, and J. Zhang. The MONK's problems: A performance comparison of different learning algorithms. Technical Report CS-91-197, Carnegie Mellon University, Pittsburgh, PA, 1991.
35. C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Oxford, May 1989.
36. Stewart W. Wilson. Knowledge growth in an artificial animal. In John J. Grefenstette, editor, *Proceedings of an International Conference on Genetic Algorithms and their Applications*, pages 16–23, Pittsburgh, PA, July 1985. Lawrence Erlbaum Associates.
37. Stewart W. Wilson. Bid competition and specificity reconsidered. *Complex Systems*, 2(6):705–723, 1988.
38. Stewart W. Wilson. ZCS: A zeroth level classifier system. *Evolutionary Computation*, 2(1):1–18, 1994.
39. Stewart W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.
40. Stewart W. Wilson. Generalization in the XCS classifier system. In J. Koza et al., editor, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, San Francisco, CA, 1998. Morgan Kaufmann.

A The Monk's Problems

The concepts represented by each of the Monk's problems are as follows:

1. $A1 = 1$ OR $A5 = 1$
2. Exactly two of $\{A1 = 1, A2 = 1, A3 = 1, A4 = 1, A5 = 1, A6 = 1\}$
3. ($A5 = 3$ AND $A4 = 1$) OR ($A5 \neq 4$ AND $A2 \neq 3$) with 5% noise in training set

B Coverage Tables

The coverage tables presented in this appendix are mappings of the entire Monk's attribute space with each point marked as being correctly classified as 1 or 0 or incorrectly classified (X) and are closely based on those presented in [34]. The first three attributes are mapped to the rows of the table and the last three to the columns, thus making every point on the table correspond to only one point in the six-dimensional attribute space.

B.1 Monk's 1 Problem

These coverage tables are produced after test iterations of XCS and indicate the population's ability to classify the every point in the attribute space.

The coverage table for perfect classification of the Monk' s 1 problem is shown in Fig. 5. This was produced in all but 5 of the 30 runs in the Monk's 1 experiment. Of the five other runs the coverage tables showed that the worst run had 21 misclassifications with the other four having no more than 8 misclassifications.

```
111111111111111111111111
111111111111111111111111
110000001100000011000000
110000001100000011000000
110000001100000011000000
110000001100000011000000
110000001100000011000000
110000001100000011000000
110000001100000011000000
111111111111111111111111
111111111111111111111111
110000001100000011000000
110000001100000011000000
110000001100000011000000
110000001100000011000000
110000001100000011000000
110000001100000011000000
111111111111111111111111
111111111111111111111111
```

Fig. 5. The coverage table for 100% classification accuracy of the Monk's 1 problem as produced from 25 out of 30 runs

B.2 Monk's 2 Problem

The typical coverage table for the Monk's 2 problem (Fig. 6) indicates that although XCS has learnt the training data perfectly the system classifies with $\approx 92\%$ accuracy over the whole set.

```
000000000000X0X0X000X0X0X
000X0X0X00X1010100X10101
000000000001010100010101
0X0X0X0X0110101001101010
000000000001010100010101
0X0X0101X110101001101010
000000000001010100010101
0X0101010110X0X00110X0X0
000101010110101001101010
011010101X0000001X000000
000101010110101001101010
011010101X0000001X000000
000000000001010100010101
0X0101010110X0X00110X0X0
000101010110101001101010
011010101X0000001X000000
000101010110101001101010
011010101X0000001X000000
```

Fig. 6. A typical coverage table for the Monk's 2 problem produced by XCS using the enumeration encoding

B.3 Monk's 3 Problem

The coverage table produced on the majority of runs of XCS on the third Monk's problem (Fig. 7) demonstrates 100% classification accuracy. On other runs the coverage table showed that XCS had misclassified 6 out of 432 (1.4%) examples.

```
111111001111110011111100
111111001111110011111100
111111001111110011111100
111111001111110011111100
000011000000000000000000
000011000000000000000000
111111001111110011111100
111111001111110011111100
111111001111110011111100
111111001111110011111100
000011000000000000000000
000011000000000000000000
111111001111110011111100
111111001111110011111100
111111001111110011111100
111111001111110011111100
000011000000000000000000
000011000000000000000000
```

Fig. 7. A perfect coverage table produced by XCS for the third Monk's problem

Learning Classifier Systems Applied to Knowledge Discovery in Clinical Research Databases

John H. Holmes

Center for Clinical Epidemiology and Biostatistics
University of Pennsylvania School of Medicine
jholmes@ccen.med.upenn.edu

Abstract. A stimulus-response learning classifier system (LCS), EpiCS, was developed from the BOOLE and NEWBOOLE models to address the needs of knowledge discovery in databases used in clinical research. Two specific needs were investigated: the derivation of accurate estimates of disease risk, and the ability to deal with rare clinical outcomes. EpiCS was shown to have excellent classification accuracy, compared to logistic regression, when using risk estimates as the primary means for classification. This was especially true in data with low disease prevalence. EpiCS was designed to accommodate differential negative reinforcement when false positive or false negative decisions were made by the system. This feature was investigated to determine its effect on learning rate and classification accuracy. Tested across a range of disease prevalences, the learning rate improved when erroneous decisions were differentially negatively reinforced. However, classification accuracy was not affected by differential negative reinforcement.

1 Introduction

The idea of searching through databases for "hidden" knowledge, or concepts that are not pre-conceived by the searcher, has been gaining considerable attention over the last several years. This enterprise has been referred to as "knowledge discovery in databases," or simply, *knowledge discovery*, the "non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data" [5]. Knowledge discovery is aided by a number of tools, applied in the process of *data mining*; thus, one *mines* data to *discover* knowledge. While most of the effort in knowledge discovery research has focused on financial and other business-related databases, medical databases of all types pose a unique problem, in that the characteristics of the data they contain are complex in form and content. Databases used in clinical research illustrate this complexity, especially those created for *epidemiologic surveillance*, or "[...]the systematic collection of data pertaining to the occurrence of specific diseases, the analysis and interpretation of these data, and the dissemination of consolidated and processed information[...]" [20].

Surveillance data are often incomplete or missing, contradictory, or conflicting, due to the nature of the data themselves; they may even be downright erroneous due to poorly implemented systems and hastily collected data. Furthermore, these data

P.L. Lanzi, W. Stolzmann, and S.W. Wilson (Eds.): LCS '99, LNAI 1813, pp.243-261, 2000.

often represent rare outcomes or they contain obscure relationships that are difficult to characterize, especially using traditional statistical methods. Thus, epidemiologic surveillance databases provide an important platform for investigating the use of knowledge discovery tools in clinical research, in that they represent a significant part of the universe of medical databases, while presenting significant challenges for those who perform knowledge discovery research.

This paper examines the application of a learning classifier system (LCS) to two specific concerns in clinical research: discovery of knowledge required to ascertain an individual's risk of developing a disease outcome, and the ability of a LCS to learn classification rules in data where the classes are distributed unevenly. While there have been some prior investigations into the application of LCS to knowledge discovery in medical data, most notably [2], this work has not addressed these concerns. It is hoped that this paper will benefit LCS researchers by informing them of two specific facets of clinical data and the challenges they pose for those who seek to apply the LCS to clinical research knowledge discovery.

2 Background: Two Facets of Clinical Research and Their Implications for Knowledge Discovery

2.1 Assessment of Disease Risk

An essential goal of clinical research is the elicitation of cause and effect relationships, or *causation*. Traditionally, epidemiologists [22] and classical philosophers [14] have avoided the notion of absolute causation, that an event or other factor can be shown to cause another, because of the possibility of coincidence (Hume's problem), incorrect sampling, or some other flaw inherent in observation. Sample-based statistical methods of inference allow the epidemiologist to avoid gracefully the problem of defending causation. Rather than assert that a factor *X* (such as a toxin exposure) will *cause* disease *Y*, epidemiologists are more comfortable in saying that individuals with exposure to factor *X* are at increased *risk* of developing disease *Y*. Risk is the probability of occurrence of some event; thus, risk is often expressed in terms of chance: "Individual A has a 30% higher chance of contracting a pneumococcal infection than Individual B." A risk factor is some condition, event, disease, or other characteristic which is statistically associated with an increase in risk for a given disease. Frequently, the outcome of interest in an epidemiologic investigation is binary in value, such as diseased/non-diseased, or dead/alive. In addition, it is important to adjust for a number of factors that may affect the risk associated with a given factor; this adjustment is performed by multivariate modeling. Because clinical outcomes are often expressed dichotomously (disease/no disease, dead/alive), such modeling is performed using logistic regression.

Logistic Modeling. When performing multivariate statistical analysis where a binary outcome variable is used, the preferred method is logistic regression, in which the risk of developing an outcome is expressed as a function of a set of predictor (or independent) variables. The dependent variable in the logistic model is the natural logarithm of the odds of disease:

$$\ln\left[\frac{P_x}{1-P_x}\right] = \alpha + b_1 X_1 + \ldots + b_n X_n \quad (1)$$

where P_x is the probability of disease for a specified covariate x, is the log odds of developing the outcome (or logit), accounting for all exposure variables in the model (rather than a single exposure variable), α is the intercept, and b_i is the coefficient for the independent variable X_i. Rewriting this equation produces a method for representing the estimated probability of developing the outcome of interest:

$$P_y = \frac{1}{1+e^{-(\alpha+\beta_1 x_1+\ldots+\beta_n x_n)}} \quad (2)$$

where P_y is the estimated probability of developing an outcome, given the presence of risk factors $x_1 \ldots x_n$. The formula in Equation (2) is used to derive *clinical prediction rules.*

Clinical Prediction Rules. Clinical prediction rules are used to classify patients in terms of their risk of developing a specific outcome. While the ultimate goal of applying a prediction rule is to determine a categorical diagnostic classification into which a patient will fall (such as "Disease" or "No Disease"), the categories are actually created from a continuous measure of risk derived in Equation (2).

For example, assume the following logistic model was derived from data collected during a food-borne infection outbreak:

log odds of disease = (-0.03 +(1.672*ChickenSalad)+(0.013*Rolls)+(0.045*IceCream))

where the intercept is –0.03 and the value of each variable is multiplied by its coefficient. In this example, each of the independent variables is coded dichotomously (0=No, 1=Yes) to indicate exposure (whether or not the item was eaten). Thus, for an individual who ate chicken salad, rolls, and ice cream:

Variable	Coefficient	Value of variable	Coefficient*Value
Intercept	-0.03	-	-0.03
Chicken Salad	1.672	1 (1=yes)	1.672
Rolls	0.013	1	0.013
Ice Cream	0.045	1	0.045
		Total	1.700

This individual would be at increased risk of disease, as the probability of disease is $1/1+e^{-1.700}$, or 0.85, which corresponds to an odds of disease of 0.85/(1-0.85), or 5.67. In this case, the real culprit is chicken salad, as can be seen from the size of its coefficient, relative to the coefficients for Rolls and Ice Cream.

2.2 The Challenges of Rarely Occurring Disease Outcomes

As noted above, much of the data gathering activity of clinical research occurs as a result of epidemiologic surveillance, which seeks to study the "[...]occurrence and distribution of diseases and other health-related conditions in populations[...]" (Kelsey, Thompson, and Evans 1986). Disease "occurrence" is measured in terms of *incidence* and *prevalence*. Incidence reflects the rate of change from a healthy state to a diseased state over time, indicating the occurrence of new cases of disease within a population per unit time. Incidence is usually expressed as a rate, such as 20.4 cases per 100,000 persons per year. Prevalence is a quantitative measure of the existence of disease in a population at a given point in time and is expressed as a proportion, such as 50.2 cases per 100,000 per persons, or a 0.05.

During the early course of epidemiologic surveillance, it is extremely rare to find individuals afflicted with a given disease outcome at the same rate as one finds those without the disease. For example, even in the course of investigating an infection outbreak in a small group of people exposed to an organism known to cause disease in the majority of those exposed to it (such as *Salmonella*), an investigator would ordinarily find relatively few incident cases early on, due to the time required to incubate the organism. However, it is especially during this early period of surveillance that a robust classification rule would be needed, so that appropriate interventions may be planned and implemented. In order to address the requirement of accurate classification of individuals early in a clinical investigation, when disease or other outcomes are rare, the LCS researcher must attend to the characteristics of clinical data, especially those influenced by the prevalence of disease.

These two concerns, ascertainment of disease risk and the problem of rare disease or other clinical outcomes, are addressed in the design of EpiCS, a LCS intended for use in knowledge discovery in clinical research databases

3 EpiCS: A Learning Classifier System for Clinical Research

The author developed a stimulus-response LCS, BOOLE++ [11] and later, EpiCS [12], which was based on Wilson's early work on the Animat [26], BOOLE [27] and NEWBOOLE [1]. The reader is referred to these works for descriptions of BOOLE and NEWBOOLE. EpiCS is an object-oriented system in design and implementation, and is currently deployed on an Intel platform. Like other learning classifier systems, EpiCS's representation scheme expresses a rule as a *classifier*, or a condition-action pair, consisting of a taxon and an action bit, respectively, and contained in a classifier population of constant size.

3.1 Novel Design Features of EpiCS

EpiCS departs from NEWBOOLE and its predecessor, BOOLE, in several ways: algorithms for controlling under- and over-generalization, a method for determining risk as a measure of classification, and a method for differentially reinforcing erroneous decisions made by the system as a means of addressing the problem of rare disease outcomes.

Controlling Over-Generalization. Clinical data are generally "noisy," in that they are often full of contradictions between exposure factors and outcomes. For example, not everyone exposed to a particular factor will develop a given disease. Early investigations with EpiCS showed that the system tended to over-generalize the population when trained with noisy data, such that the classifications made by the system were useless. In these situations, the classifiers would be sufficiently strong to survive training, but would still contain a surfeit of non-specific bits (*s). Robertson and Riolo [21] suggested that overgeneralization can be controlled in classifier systems, via taxation or classifier deletion based on time since last use. In EpiCS, a "governor" is implemented, which evaluates each classifier in the population at each iteration, and re-initializes overly general classifiers with randomly-assigned bits. The effect of this procedure is to delete overly general classifiers and replace them with less general classifiers. After experimentation with a range of values from 0.50 to 1.00, the governor was ultimately parameterized at 0.85; thus, a classifier with *s comprising more than 85% of its length would be re-initialized.

Controlling Under-Generalization. In using EpiCS on clinical data, it was found that a significant proportion (10%) of testing cases could not be classified, correctly or incorrectly. Examination of the classifier population at the end of training revealed that it had a preponderance of highly specific classifiers. As a result, the classifier population at the end of training was ill-equipped to classify testing cases, and this resulted in poor classification performance. Experimenting with the penalty factor (*p)* over a range of values (0.25, 0.50, and 0.75) led to the observation that the value of *p* originally used in NEWBOOLE (0.95) was apparently too high, resulting in the premature demise of general (but useful) classifiers. It was found that decreasing *p* to 0.50 in EpiCS resulted in much-improved performance and decreased numbers of unclassifiable cases on testing.

Determination of Outcome Risk. EpiCS is unique among LCS in that it calculates an estimate of outcome risk for a novel, previously unseen observation, given a specific combination of features in that observation's taxon. This differs from the usual approach of assigning a single nominal-level classification to a novel case. In EpiCS, a different paradigm is used to determine the type of decision made by the system; this is based upon the proportions of positives and negatives in the *match set* (the subset of classifiers in a population matching the taxon of an input case).

For example, a case in the training set would be labeled as a positive if, in the match set, there existed a preponderance of classifiers predicting positive for disease. This would be the situation even if the strongest classifier in the match set were negative. Thus, rather than produce an output decision based upon strength, the system produces a prediction based upon the *prevalent* outcome present in the match set. The output of the system would now be *risk of disease*, rather than a dichotomous decision (that being the presence or absence of disease). The advantage of this approach is that it provides a decision which is continuous in nature and which can be "cut" at various points along its range. These *cutpoints* can be used as "decision thresholds" at which a case could be determined to be in one class or another.

To implement this risk-based classification paradigm, each case in the testing set is presented to the trained classifier system and evaluated for the probabilities of

presence and absence of disease; these are determined from the proportion of classifiers matching a given input case taxon. For example, suppose an input case (reduced for purposes of illustration to a five-bit taxon, 01100) were presented to the system. In this example, the action bit determines the disease status (0=No disease, 1=Disease). Further suppose that four candidate classifiers (those with taxa matching that of the input taxon) were extant in the match set, each with a probability based on proportionate representation in the match set:

Classifiers	Proportion in match set
*10*0:1	0.60
0*100:0	0.05
01*00:1	0.22
0***0:1	0.13

Assuming that the action bit represented the presence of disease with a 1 and the absence with a 0, the *classifier system-derived probability of disease* (CSPD) for the given input case would be 0.95:

$$\text{CSPD} = \frac{\sum(\text{probabilities of classifiers associated with disease})}{\sum(\text{probabilities of all classifiers with matching taxa})} = \frac{0.95}{1.0} = 0.95$$

This procedure is repeated for each case in the testing set, such that each testing case will have been assigned a risk of outcome by EpiCS.

Differential Reinforcement. Before using a LCS in clinical settings, one would want to evaluate its *validity*. Validity reflects the overall ability of a diagnostic tool, such as a clinical test, measurement, or clinical prediction rule, to classify correctly the pre-clinical status of an individual [10]. In evaluating the validity of a tool, the availability of some type of standard is presumed; in clinical research, this standard is often referred to as the *gold standard*, implying that there is no more accurate indicator of a patient's diagnostic status. An individual classified as positive by the test and having the disease, as verified by the gold standard, would be called a *true positive*. Likewise, an individual with a negative test who did not have the disease would be a *true negative*. A *false positive* is one who was free of the disease in question, but had a positive test result, while a *false negative* is one who had the disease but tested negatively. Figure 1 shows the types of correct and erroneous classifications.

	True classification of a case ("Gold Standard")	
Tool's decision	Positive	Negative
Positive	True positive	False positive
Negative	False negative	True negative

Fig. 1. A confusion matrix comparing the output of a diagnostic tool with gold-standard knowledge.

Because of the differential nature of erroneous clinical, one would want the capability of manipulating the ratio of false positives to false negatives. The cost of

making a false negative decision is considerably higher than a false positive, especially in a disease such as cancer. While the cost of a false positive in many cancers is limited to the cost of therapy (which *may* include death or other adverse reaction to therapy), the cost of a false negative in these diseases is nearly always death. The false negative rate is sensitive to decreasing prevalence, while the false positive rate is sensitive to increasing prevalence. This is merely a function of the relatively small number of positives compared to the larger number of negatives, but it has important consequences for reinforcement learning in medical domains. If one accepts that false negative decisions are potentially more dangerous than false positives, one would want to bias decision errors in favor of false positives, with the aim of reducing false negative errors disproportionately to false positive errors.

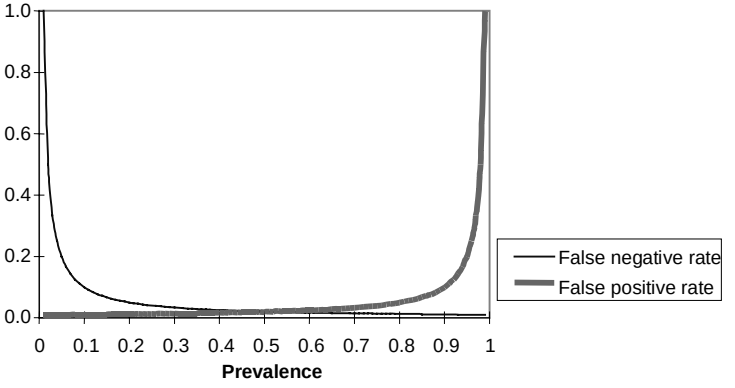


Fig. 2. The effect of a single erroneous decision on false negative and false positive rates. For purposes of illustration, this plot assumes a sample size of 100.

The effect of a single erroneous decision on the false negative and positive rates across a range of prevalences in a data set of 100 observations is shown in Figure 2. At very low prevalence, such as 5%, a single false negative will result in a false negative rate of 20% (one false negative out of five gold standard positives). A single false positive would result in a false positive rate of 1/95, or 1.05%.

The ratio of false positives to false negatives has been used as an element in a reward function in a classifier system. Previous work [23] put the optimal ratio of 8:1, but did not examine the effect of adjusting the ratio on data at different prevalences. In a LCS using payoff/penalty reinforcement, the penalty could be adjusted to bias the negative reinforcement an inaccurate classifier receives, and this strategy is employed in EpiCS.

4 Methods

4.1 Evaluation Metrics

This paper summarizes the results of investigations using EpiCS to address the two facets of clinical research described above. These investigations used a common suite

of evaluation metrics, themselves borrowed from the domain of clinical research and commonly used to evaluate the classification performance of diagnostic tests, of which EpiCS could be considered one. These metrics include sensitivity, specificity, and area under the receiver-operating characteristic curve.

Sensitivity and Specificity. Sensitivity indicates a test's ability to classify correctly gold standard-positive cases; as a result, sensitivity is often referred to as the true positive rate:

$$Sensitivity = \frac{True\ positive\ decisions}{All\ gold\ standard\ positives}$$

Sensitivity is influenced by the distribution of the classes in data, with a tendency toward increased false negatives in data where the prevalence of positive examples is low. In these data, sensitivity will correspondingly be lower than at higher positive base rates, simply because the proportion is based on a smaller denominator. For example, if there are 10 gold-standard-positives, and five true positive decisions, the sensitivity is 0.50. However, if there are 100 gold-standard positives and the same number (five) of true positives, the sensitivity is 0.95.

Specificity, or true negative rate, measures the ability of a test to classify correctly those without disease:

$$Specificity = \frac{True\ negative\ decisions}{All\ gold\ standard\ negatives}$$

Specificity is also influenced by the distribution of classes in data.

The relationship between the true positive and false positive rates can be demonstrated graphically on a *receiver-operating characteristic curve*.

The Receiver Operating Characteristic (ROC) Curve. The ROC curve is created by plotting the true positive rate (sensitivity) on the vertical axis against the false positive rate (1-specificity) on the horizontal axis. Figure 3 shows the ROC curve for a single true positive/false positive pair. The significance of the straight line through the origin will be discussed below.

The ROC curve is used to determine the overall usefulness of a diagnostic test. In order to determine whether a test is useful, it must be evaluated for its discrimination accuracy, or its ability to classify normal and abnormal patients. The measure used for this purpose is the area under the ROC curve.

While the graphical representation of the ROC curve is of interest in determining appropriate diagnostic cutoffs for a given test, the *area under the ROC curve* (θ) is important for demonstrating the ability of the test to classify both true positives and true negatives, simultaneously, as a *single measure*. The area under the ROC curve has been used extensively in medical decision making as a standard method for evaluating diagnostic test performance [7, 24]. In addition, it has been proposed for use in knowledge discovery and data mining domains [19, 13].

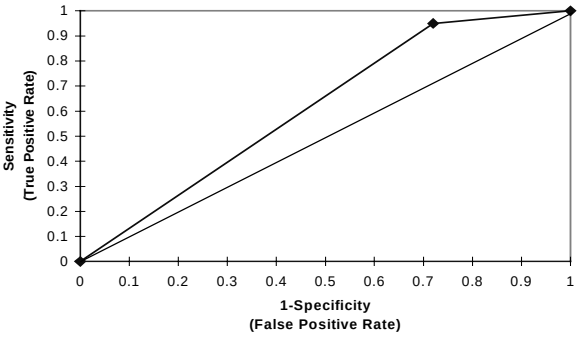


Fig. 3. Receiver-Operating Characteristic Curve for a single true positive/false positive pair.

The area under the ROC curve represents the probability of a true response in a two-alternative forced-choice (Yes-No) task; thus, the quantity (1-area) is the false alarm rate [8]. The ROC curve for a test which classifies well will have a "shoulder" closer to the upper left-hand corner of the plot, and farther from the 45-degree diagonal (shown in Figure 3). This test would have a high sensitivity and a high specificity and, as a result, a higher θ. A test which contains no information would plot on the 45-degree diagonal through the origin. The area under such a "curve" would be 0.50, indicating that the test would discriminate only as well as a coin-flip. A nonparametric method based on the Wilcoxson statistic (W) and its standard error (SE_w) is regularly used in approximating θ and its standard error, SE_θ [9]. The plausibility of calculating θ as derived from a single true positive/false positive pair has been shown [17].

All θ reported in this paper have been corrected for cases which could not be classified on testing, using the following formula:

$$Corrected\ \theta = \frac{\theta}{1 + \mathrm{IR}} \tag{3}$$

where

$$IR = \frac{\text{Number of unclassifiable cases}}{\text{Total number of cases to be classified}} \tag{4}$$

Software tools exist for constructing ROC curves, calculating areas, and comparing them by various means [3, 18]. In addition, the reader is referred to [9] and [16] for a thorough description of the algorithm for approximating the area under the ROC curve and its standard error.

4.2 Testbed Data

The testbed data were artificially created to represent surveillance of liver cancer in a chemical manufacturing plant. While EpiCS has been applied to a number of "real" datasets, and these investigations are reported elsewhere [11, 12], the design of one of

the investigations reported here required substantial manipulation of the data to ensure comparability across a series of disease prevalences. No existing "real" dataset could provide this versatility.

Simulation datasets consisting of 15 demographic, medical history, and exposure variables, one outcome variable, and 500 observations were created using the random data generator routines supplied with the EpiInfo [4] epidemiologic analysis software package. The datasets represented epidemiologic surveillance for hepatocellular (liver) carcinoma in a group of individuals working in a chemical factory who may have been exposed to one or more suspected hepatocarcinogens. All variables were coded dichotomously, with 0s or 1s used to indicate the absence or presence, respectively, of a variable. The dataset was created in such a way as to bias one variable (exposure to Vinyl Chloride (ViCl), a known hepatocarcinogen) toward association with the disease outcome. All other variables were randomly assigned a value of 0 or 1 using a random normal deviate procedure. The datasets represented a range of outcome prevalence rates: 50%, 25%, 15%, 10%, and 5%.

Training and testing sets were created by randomly selecting records from each of the five datasets at a sampling fraction of 0.50 without replacement; thus, training and testing sets were equal in size and mutually exclusive. Care was taken to ensure that all training and testing sets contained equivalent prevalence of positive (diseased) cases. For example, 250 records, 125 with hepatocellular carcinoma and 125 without, were selected at random from the 50% prevalence dataset; these records comprised the training set. The remaining 250 records in the dataset comprised the testing set. In the datasets at 25% and 15% prevalence, the number of positive cases in the training sets approximated that in the testing sets within a count of one. The results of applying this sampling scheme is shown in Table 1.

Table 1. Distribution of positive and negative cases over the range of prevalences used in this investigation.

	Main Dataset		Training Set		Testing Set	
Prevalence	Positives	Negatives	Positives	Negatives	Positives	Negatives
0.50	250	250	125	125	125	125
0.25	125	375	63	187	62	188
0.15	75	425	37	213	38	212
0.10	50	450	25	225	25	225
0.05	25	475	12	238	12	238

4.3 Experimental Procedure

General. A total of 20 trials, each consisting of a training period and a testing epoch, were performed. During the training epoch, cases selected in random order from the training set were presented to the system over a total of 30,000 iterations; a single case presentation comprised one iteration. The system was evaluated by calculating the area under the receiver operating characteristic curve (θ) at every 100th iteration to monitor learning performance. This was accomplished by testing the system with every case in the training set. At the conclusion of training, the system was tested

with each case in the testing set, and its classification performance determined by calculating θ.

Parameterization. EpiCS was parameterized as described below in Table 2. Population size (P) is the number of classifiers in the system's population, and was held at a steady state. Reward is the base value available to be shared among accurate classifiers in the correct set [C]. The generality factor (G) is used to bias R in favor of more general classifiers in P. The strengths of classifiers in [notC] are multiplied by the penalty coefficient (p) to reduce their strengths. A "baseline" p was set to .50; p was adjusted to reflect various false positive to false negative ratios, as described in Section 3. The correct set tax (e) was applied to all classifiers advocating a correct decision. The remaining parameters (r, X, and μ) refer to the genetic algorithm used in the discovery component of EpiCS.

The testbed data required 15 positions on a taxon. It is generally accepted that longer strings will require larger populations to prevent population overcrowding with overly specific strings and to improve system performance. Although a method has been proposed for calculating the optimal population size for genetic algorithms [6], the best heuristic for determining population size for classifier systems is that "more is better" [21]. Given this heuristic, and given that NEWBOOLE was parameterized at a population size of 1,000 for the 11-multiplexer problem, the performance of EpiCS using several population sizes between 400 and 2,000 was investigated. It was found that population sizes over 1,000 caused serious degradation in system performance, proportional to increasing population sizes. In addition, little improvement in learning rate was noted with increasing population size. Population sizes less than 800 resulted in overly general populations of classifiers, likely owing to the generalization pressure described in [21]. Increasing the population size from 800 to 1,000 improved the learning rate and significantly decreased overgeneralization. As a result, the population size for the investigation of epidemiologic surveillance data was fixed at 1,000.

Table 2. Parameters applied to EpiCS.

Parameter	Value
Population size (P)	1000
Reward (R)	300
Generality factor (G)	4.0
Penalty coefficient (p)	0.50
Correct set tax (e)	0.10
Genetic algorithm calls per iteration (r)	4
Crossover rate (X)	0.50
Mutation rate (μ)	0.001

4.4 Investigation-Specific Methods

Risk Prediction. In order to compare the CSPD with an accepted method of determining the probability of disease, the *logistic regression-derived probability of disease* (LRPD), or risk estimate, was calculated for each of the testing cases for

using a clinical decision rule derived from the training cases in the appropriate dataset. It was shown above (Equation 2) that probability of disease can be estimated from a decision rule derived from a logistic model. To derive the decision rule, non-stepwise logistic regression was run on the training set, using diagnosis of hepatocellular carcinoma as the dependent variable and the 15 history and exposure variables as independent terms. The prediction rule was then applied to each case in the testing set in order to obtain the LRPD for each testing case.

CSPDs were averaged over the 20 trials at each of the four prevalences, and compared with the prevalence-specific LRPD. The risk assessment investigation did not use the testbed data at 5% prevalence, as logistic regression failed to converge at this level.

Effects of Differential Reinforcement. This study investigated the effect of manipulating the penalty factor, *p*, to adjust the proportion of false negative decisions to a level deemed acceptable in datasets representing a range of prevalences. Specifically, *p* was biased to penalize false negative decisions more heavily than false positive decisions in the hope of rapidly decreasing the false negative rate over the course of training with the goal of obtaining improved learning rates and classification performance during testing with novel data. The rationale for this approach is found in the dynamics of the reinforcement component of the LCS. For example, if *p* were biased such that the strength of classifiers advocating a false negative decision were reduced by 75% and the strength of those advocating a false positive decision were reduced by 50%, one should expect that, over time, the entire body of inaccurate classifiers will contain twice as many false positive as false negative classifiers. Thus, if an error were to be made by the system, it is twice as likely to be a false positive than a false negative. Over time, the mixture of false positive and false negative classifiers in the set of inaccurate classifiers should approximate the biasing scheme of *p*.

Payoff/penalty parameterization was performed using a range of biasing schemes, presented in Table 3:

Table 3. Values of *p* over the range of FP:FN biases.

FP:FN ratio	*p* if decision is a false positive	*p* if decision is a false negative
1:1	0.50	0.50
2:1	0.50	0.75
4:1	0.50	0.875
10:1	0.50	0.95

The biasing scheme was designed using a base penalty (*p*) of 0.50. This is the penalty assigned to all false positives, and is used for calculating the penalty assigned to false negatives. Thus, the penalty assigned to false negatives is calculated relative to the 50% reduction assigned to all false positives. For example, at a FP:FN ratio of 1:1 (which is *unbiased*), the penalties are equivalent for both false positive and false negative decisions; the strengths of both types of inaccurate classifiers were reduced by 50%, such that after penalty, 50% of the strength of each such classifier would remain. However, at a biased ratio of 2:1, false negatives were penalized at twice the rate employed in the 1:1 scheme, such that 25% (1-*p*, or 1-0.75) of the strength prior

to penalty remained in a false negative, compared to 50% in a false positive. The relative *target* reduction in strength, or the new strength after penalty, of a false negative classifier is proportionally twice that of a false positive. For example, a false positive and a false negative classifier, both with strengths of 100, would be reduced to 50 and 25, respectively, under a 2:1 ratio. The target reduction would be 50 and 12.5, respectively, for a 4:1 ratio, and so on.

Learning Metrics. Specialized metrics were used to evaluate the learning performance of EpiCS as it encountered clinical data at different prevalences. At each 100th iteration during the training epoch, the θ was plotted over time; this plot provides a visual display of the learning performance of the system, represented as a *learning curve*. At the end of training, the learning curve data were evaluated for the first appearance of a *shoulder*, defined as the number of iterations to reach the point where the curve first reaches 95% of the maximum θ value. The shoulder of each curve obtained for each FP:FN ratio, was calculated and used with the θ at that shoulder to compute a learning rate (λ):

$$\text{Learning rate} = \lambda = \frac{\theta \text{ of curve at its shoulder}}{\text{Shoulder}} \bullet 1000 \tag{5}$$

Learning rates obtained for the curves of biased FP:FN ratios were compared with those from the unbiased, or baseline, curves, to produce a difference score, *D*, which reflects the percent improvement in learning rate a specific biasing scheme provides over baseline:

$$\text{Difference score} = \text{D} = \frac{\lambda_{\text{baseline}} - \lambda_{\text{biased}}}{\lambda_{\text{baseline}}} \bullet 100 \tag{6}$$

D was calculated across mean λs for each FP:FN ratio and prevalence; a threshold of 15% was set arbitrarily, such that *D* had to exceed 15% to be considered significant.

5 Results

5.1 Risk Prediction

Figure 4 summarizes the θ obtained by applying the logistic regression-derived decision rule to the testing sets, and the mean θ for the 20 trials at each of the four prevalences studied. For both methods, θ decreased concomitantly with decreasing prevalence; this is due to larger number of false positive classifications made by either method as prevalence decreases. More striking, however, is EpiCS's performance, compared with logistic regression. Across all prevalences, EpiCS consistently proved to be a better classifier of unseen, novel cases than the logistic regression-derived decision rule. At the 15% and 10% prevalences, EpiCS provided a clinically acceptable θ on average, although the variability observed at these prevalences

demonstrated that a single learning-testing trial would be insufficient to classify novel cases reliably.

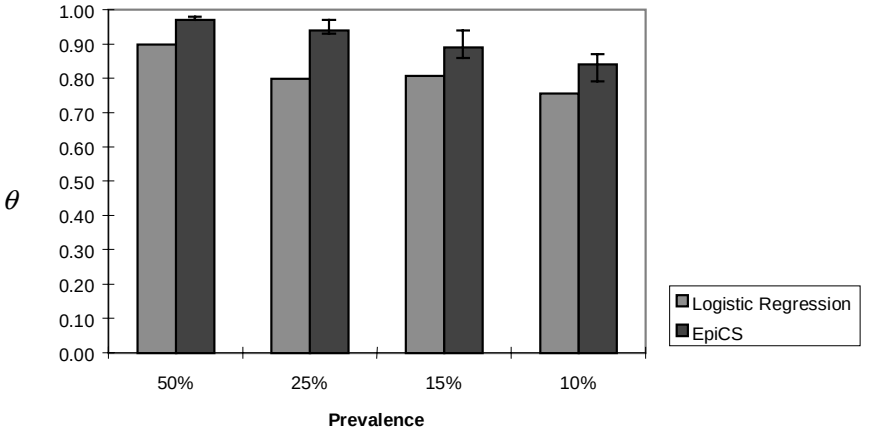


Fig. 4. Areas under the receiver operating characteristic curves for logistic regression and EpiCS across all four prevalences. The latter were averaged over 20 trials. The error bars represent one standard deviation.

5.2 Effects of Differential Negative Reinforcement on Learning and Classification Performance in Data with Unequal Class Distributions

Table 4 shows the effect during the training epoch of parameterizing the reinforcement component of EpiCS with various FP:FN ratios. At 50% prevalence, there was no significant difference (no D>15%) between the shoulders of the unbiased and biased curves, indicating that the biasing of FP:FN ratios had no effect on learning rate at this prevalence. At 25% prevalence, considerable separation is found between the shoulder at a FP:FN ratio of 1:1 and shoulders at the biased ratios., with Ds ranging between 46.2% and 61.5%. Thus, changing the FP:FN ratios affected the learning rate, with the earliest shoulder occurring at a ratio of 4:1; at this ratio, the learning curve represented a 61.5% improvement in the λ over baseline. The trend in shoulder differences between the biased ratios and the 1:1 ratio that was observed in the 25% prevalence data was even more pronounced at the lower prevalences.

Table 4. Biased curve difference score data obtained during the training epochs at various FP:FN ratios. Difference scores are referenced to the shoulders obtained at FP:FN ratio of 1:1 at the respective prevalence level.

	Prevalence				
FP:FN	50%	25%	15%	10%	5%
2:1	9.1	46.2	57.1	61.5	44.4
4:1	0.0	61.5	47.6	69.2	11.1
10:1	6.1	50.0	52.3	100.0	88.8

Figure 5 shows the effect of adjusting the FP:FN ratio on the classification ability of EpiCS based on θ obtained at testing with novel cases; error bars denote one standard deviation. At 50% prevalence, the classification performance of EpiCS was best at a 1:1 ratio, although it was nearly as good at 10:1; classification performance was not statistically different across all FP:FN ratios. At 25% prevalence, classification performance was not as good as at 50%. As in the 50% prevalence data, classification performance was essentially identical across all FP:FN ratios. At 15% prevalence, no appreciable difference was noted across the three lowest FP:FN ratios. All outcome measures were similar, although a slight yet statistically insignificant trend in increasing θ, and sensitivity was noted with increasing FP:FN ratio. In addition, the outcome measures generally exhibited decreasing variability as the FP:FN ratio increased. For example, the mean θ, at a ratio of 1:1 was 0.84, with a standard deviation of 0.09; at a ratio of 10:1, the mean θ, was 0.87, with a standard deviation of 0.04. However, these differences were not statistically significant. In the 10% prevalence data, biasing the FP:FN ratio was found to have no significant effect on the outcome measures. In addition, the decreasing trend in variability of these measures with increasing FP:FN ratios, noted at 25% and 15% prevalence, was not found at 10% prevalence. At 5% prevalence, no biasing effect was found; in addition, the variance was higher than that found at the higher prevalences. However, as seen in at 15% prevalence, increasing the FP:FN bias tended to decrease variance. Overall, classification performance on testing was impaired with decreasing prevalence, and differential penalty biasing had no apparent effect on performance at testing.

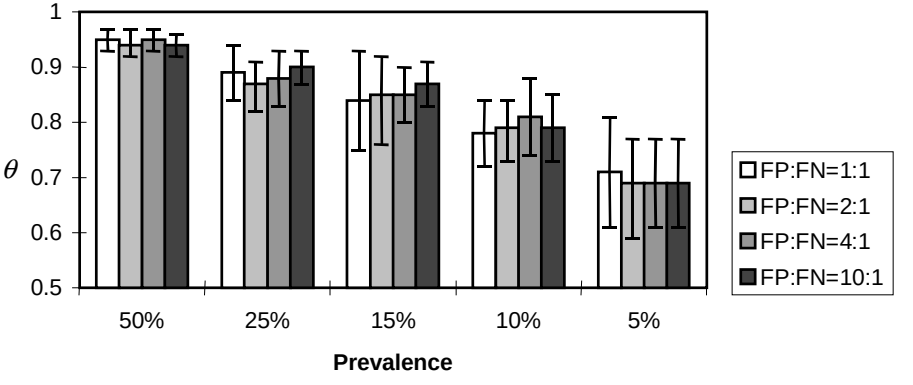


Fig. 5. Areas under the ROC curve obtained on testing at each prevalence for each FP:FN ratio.

6 Discussion

6.1 Risk Assessment

This investigation compared the classification performance of EpiCS with that of a traditional statistical method for classification, logistic regression. First, EpiCS was found to perform better than logistic regression in deriving an estimate of risk of disease for noisy, epidemiologic data. Excellent classification ability was noted for EpiCS over all training-testing trials. Second, this investigation demonstrated the

successful implementation of risk-based output from a learning classifier system, rather than a single, categorical classification. Third, logistic regression, like all inferential statistical methods, relies on sufficient numbers of cases to be valid. In addition, these statistical methods rely on numerous assumptions as to the characteristics of the data. In the early phases of epidemics, or in the case of rare diseases, where small numbers of disease-positive cases are available, it is often difficult to meet these assumptions. However, researchers still need a method for characterizing diseased and non-diseased individuals so that those at risk may be identified. A learning classifier system such as EpiCS fills this need because it is not constrained by the assumptions that may hamper traditional statistical analysis.

Finally, epidemiologists and practicing clinicians are seldom content to accept a single categorical classification for a given patient. They are much more comfortable with probabilities of outcomes, given specific patterns of features, so they can frame a patient's chances of having a disease along a continuous scale, rather than a simple yes or no classification. The ability of a classifier system to provide a continuous measure of risk is paramount to the goal of implementing genetics-based machine learning systems in clinical domains.

6.2 Differential Negative Reinforcement

Biasing the FP:FN ratio had no effect on learning rate or classification ability when using data at 50% prevalence. In addition, no significant effect on classification ability was found at the remaining three prevalences. However, a progressive improvement of learning rate was observed when biasing the ratios using data of decreasing prevalence. This was evident in observing the differences between the shoulders of biased and unbiased curves.

There appeared to be an optimum FP:FN ratio at each prevalence. For example, at 25% prevalence, the largest shoulder difference between the 1:1 unbiased ratio occurred when biasing at the 4:1 ratio. The largest difference at 15% prevalence was at the 10:1 ratio, which was also optimal at 10% prevalence. At 5% prevalence, the largest difference was found at 10:1 as well. Clearly, the maximum differences were found in ratios that reflected (although not exactly) the ratios of negative to positive cases in the data. In summary, biasing the FP:FN ratio improved learning rate in the lower prevalence data, and the amount of bias that was required to obtain the optimum learning rate appeared to be proportional to the prevalence.

EpiCS did not do as well on testing with lower prevalence data. Specifically, the decreased sensitivity, with increased variance, contributes to the lower θ. This appears to be caused by the increasingly large numbers of true negatives exerting pressure on the true positives, resulting in a decision which is more likely to classify as negative a truly positive case. Although not a focus of this investigation, an inspection of the match sets [M] created during the testing epochs revealed that their membership was divided approximately according to the proportion of positives and negatives in the training set. That is, the classifiers matching a given testing case were less likely to advocate a disease-positive in lower prevalence data; this relationship appeared to be linear with decreasing prevalence. Unfortunately, epidemiologic data are virtually never "perfect" in the sense that there is a crisp definition between sets of features defining one class or another. Rather, there is usually a great deal of noise in these data, such that a single pattern may classify both

positive and negative. This promises to be an intriguing problem for research into the application of a LCS to epidemiologic domains.

7 Conclusion

This paper summarized two investigations conducted into applying a stimulus response learning classifier system, EpiCS, to the task of discovering knowledge in clinical databases. The types of knowledge in databases of this type are several, although these investigations focused on clinical prediction rules that could be used to classify an individual base don disease risk. To this end, EpiCS was specifically configured to calculate risk estimates based on the composition of match set membership; this represents a new approach to deriving class membership of unseen cases. EpiCS addresses the need of clinical researchers to ascertain disease risk accurately, especially in those outcomes where classification is not merely dichotomous, or where the surface between two classes is movable along a continuum. Further research into the risk ascertainment algorithm is needed, however, as several questions go begging at this stage in EpiCS's development. For example, how well does EpiCS's risk algorithm deal with missing data, which are very common in clinical research? If this question can be addressed successfully, this would place EpiCS even further ahead of logistic regression as a method for determining risk of disease.

In allowing the differential negative reinforcement of erroneous classifications, EpiCS also addresses the reality that clinical data often have unequal class prevalences. It is well-known that classifiers in general, and the LCS is no exception, produce biased estimates of accuracy in the face of low (or high) prevalence of one class over another. EpiCS allows the manipulation of reinforcement of false positive and false negative decisions, with the goal of reducing learning time, and hopefully improving overall classification accuracy. As was shown in this investigation, the former goal was met, to some extent, although the relationship between the extent of differential negative reinforcement and empirically observed learning rates is not entirely clear. Unfortunately, differential negative reinforcement contributed nothing to the improvement of classification accuracy. However, both of these areas merit further investigation; without it, the LCS will remain with the other classifiers, such as neural networks, decision tree inducers, and nearest neighbor methods, as unable to deal with unevenly distributed clinical data.

References

1. Bonelli, P., Parodi, A., Sen, S., and Wilson, S.: NEWBOOLE: A fast GBML system. In *International Conference on Machine Learning*, pages 153-159, San Mateo, California, 1990. Morgan Kaufmann.
2. Bonelli, P. and Parodi, A.: An efficient classifier system and its experimental comparison with two representative learning methods on three medical domains. In R.K. Belew and L.B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms* (ICGA-4), pages 288-295. San Mateo, CA: Morgan Kaufmann, 1991.

3. Centor, R. and Keightley, G.E.: Receiver operating characteristic (ROC) curve area analysis using the ROC ANALYZER. System. In Kingsland, L.C., editor *Proceedings of the Thirteenth Annual Symposium on Computer Applications in Medical Care*, pages 222-226, Silver Spring, MD: IEEE Computer Society Press, 1989.
4. Dean, A.D.. Dean, J.A., Burton, J.H., and Dicker, R.C.: Epi Info, Version 5: a word processing, database, and statistics program for epidemiology on microcomputers. Centers for Disease Control, Atlanta, Georgia, 1990.
5. Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R.: *Advances in Knowledge Discovery and Data Mining*. Menlo Park, CA: AAAI Press; 1996.
6. Goldberg, D.E.: *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, MA, 1989.
7. Good, W.F., Gur, D, Straub, W.H., and Feist, J.H.: Comparing imaging systems by ROC studies. Detection versus interpretation. *Investigative Radiology*. 24(11): 932-3, 1989.
8. Green, D.M. and Swets, J.A.: *Signal Detection Theory and Psychophysic*s. New York: John Wiley Sons; 1966.
9. Hanley, J.A. and McNeil, B.J.: The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology* 143:29-36, 1982.
10. Hennekens, C.H., Buring, J.E., and Mayrent, S.L., editors: *Epidemiology in Medicine*. Boston: Little, Brown and Company; 1987.
11. Holmes, J.H.: A genetics-based machine learning approach to knowledge discovery in clinical data, *Journal of the American Medical Informatics Association Supplement* (1996) 883.
12. Holmes, J.H.: Discovering Risk of Disease with a Learning Classifier System. In Baeck, T., editor, *Proceedings of the Seventh International Conference on Genetic Algorithms* (ICGA-7). pages 426-433. San Francisco, CA: Morgan Kaufmann, 1997.
13. Holmes, J.H., Durbin D.R., and Winston F.K.: The Learning Classifier System: An evolutionary computation approach to knowledge discovery in epidemiologic surveillance. *Artificial Intelligence in Medicine* (Accepted for publication).
14. Hume, D.A.: *A Treatise of Human Nature* (1739). Second edition. Oxford: Clarendon Press; 1978.
15. Kelsey, J.L., Thompson, W.D., and Evans, A.S.: *Methods in Observational Epidemiology*. New York: Oxford University Press; 1986.
16. McNeil, B.J. and Hanley, J.A.: Statistical approaches to the analysis of receiver operating characteristic (ROC) curves. *Medical Decision Making*. 4:137-150, 1984.
17. McNichol, D.A.: *Primer of Signal Detection Theory*. London: George Allen and Unwin, Ltd.; 1972.
18. Metz, C.E., Shen, J.-H., and Kronman, H.B.: LabROC4: A program for maximum likelihood estimation of a binormal ROC curve and its associated parameters from a set of continuously-distributed data. University of Chicago; 1993.
19. Provost, F. and Fawcett, T.: Analysis and visualization of classifier performance: comparison under imprecise class and cost distributions. In Heckerman, D., Mannila, H., Pregibon, D., and Uthurusamy, R., editors, P*roceedings of the Third International Conference on Knowledge Discovery and Data Mining*, pages 43-48. Menlo Park, CA: AAAI Press, 1997.
20. Raska, K.: Epidemiologic surveillance in the control of infectious diseases. *Review of Infectious Disease*, 5: 1112- 1117, 1983.
21. Robertson, G.G. and Riolo, R.L.: A tale of two classifier systems. *Machine Learning* 3:139-159, 1988.
22. Rothman, K.J.: *Modern Epidemiology*. Boston: Little, Brown and Company, 1986.
23. Sedbrook, T.A., Wright, H., and Wright, R.: Application of a genetic classifier for patient triage. In Belew, R.K. and Booker, L.B., editors, *Proceedings of the Fourth International Conference on Genetic Algorithms (ICGA-4), pages* 334-338. San Mateo, CA: Morgan Kaufmann, 1991.

24. Somoza, E., Soutullo-Esperon, L., and Mossman, D.: Evaluation and optimization of diagnostic tests using receiver operating characteristic analysis and information theory. *International Journal of Biomedical Computing*, 24(3): 153-89, 1989.
25. Wilson, S.W.: Knowledge growth in an artificial animal. In Grefenstette, J.J., editor, *Proceedings of an International Conference on Genetic Algorithms and their Applications*, pages 16-23, Pittsburgh, PA, July 1985. Lawrence Erlbaum Associates.
26. Wilson, S.W.: Classifier systems and the animat problem, *Machine Learning*, 2:199-228, 1987.

An Adaptive Agent Based Economic Model

Sonia Schulenburg and Peter Ross

Artificial Intelligence Applications Institute
Division of Informatics
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN, Scotland
{sonias,peter}@dai.ed.ac.uk

Abstract. In this paper we describe a simple model of adaptive agents of different types, represented by Learning Classifier Systems (LCS), which make investment decisions about a risk free bond and a risky asset under a well defined stock market environment. Our main aim is to explore the degree of reliability that artificially intelligent agents can have when applied to real life economic problems. We do this by evaluating whether an LCS is able to represent competent traders in a real market scenario in which daily stock prices and dividends are given to the agents exogenously, so permitting us to focus on the dynamics and evolution of the behaviour of these evolving traders without having to be concerned about how their actions affect the market.
We present results of adaptive and non-adaptive simulations over a period of ten years of real data of a specific stock and show that the artificial agents, by displaying different and rich behaviours evolved throughout the simulations, are able to discover and refine novel and successful sets of market strategies that can outperform baseline strategies such as buy-and-hold or merely keeping money in the bank at a good rate of interest, even though the agents pay commission on every trade.

1 Introduction

It is common to treat stock prices, foreign exchange rates, etc., as time series and to use time-series techniques such as recurrent or complex-valued networks to try to predict their behaviour. However, such methods do not directly model the causes of the behaviour and therefore can be problematic (e.g. see [7]) when the underlying causes change – for example, there is the question how much of the historical data is relevant to the current situation, and the question of when retraining becomes necessary. Our understanding of precisely what it is that generates such nancial time series is still very limited. This paper takes an alternative approach and seeks to model agent (that is, trader) behaviour rather than the time series, because traders behaviour is one of the major causes of market behaviour.

There is already some work in this area – see [16] for a good review. Current models typically use (i) various non-adaptive heterogeneous agents of each type

P.L. Lanzi, W. Stolzmann, and S.W. Wilson (Eds.): LCS 99, LNAI 1813, pp. 263–282, 2000.

(n_1 of type 1, n_2 of type 2, etc.) [2,20] which can have a high computational cost, or, (ii) a number of agents who di er from each other in the strategies they evolve (usually through a genetic algorithm) while receiving the same type of information about the market [6,19]. In such models, apart from the divergence in the rule sets, the agents are identical [17]. We believe that these restrictions do not allow the model to capture realistic features of diverse traders; in a real market, not all investors receive the same market information, nor all at the same time.

Instead, we propose using only one agent of each type, where the type is characterised by the speci c set of market statistics that it pays attention to. One agent therefore represents the aggregate e ect of all such traders. We hypothesise that such aggregates are simpler than individual agents would need to be; di erent individual behaviours average out, so to speak. For example, a group of mathematically-inclined traders may all pay attention to various moving-average price statistics. Although any one individual may have a very sophisticated decision procedure involving such statistics, the net e ect of all decisions by members of the group might perhaps be modelled by a simple decision process that uses such information. Ideally the agent, representing a group of traders, will have a variable-sized e ect on the market. This expresses the fact that if a real trader becomes successful by abasing his/her decisions on certain information, others will tend to emulate his/her approach and so the impact of that type on the market increases.

In this chapter, agents decision-making processes are represented by Michigan-style learning classi er systems (LCSs) running simultaneously, one for each trader type. This may be overly simplistic – for example, S-classi er systems in which individual conditions are LISP-like S-expressions may prove to be better – but it will be useful to explore how e ective it can be.

2 Background

Traditional approaches for modelling nancial markets such as rational expectation models usually assume that agents have complete knowledge of the structure of the market as well as other agents beliefs and objectives [21]. Such theories state that a market remains in equilibrium if the agents display a rational behaviour in anticipating changes in supply and demand. Simply looking at stock market data, though, one can observe that there is a type of volatility that does not match the rational expectations model. In addition to this, it is also assumed that traders react to the same information in a linear fashion and that in the aggregate all traders are rational (even if they do not behave rationally individually). In fact, the common assumptions are that trader skills and experience are fairly evenly distributed, and that trader reasoning is inductive [4,5].

Although in the nancial world it has been di cult to isolate a single factor that originates moves in the markets, it is an aim of this work to explore how much the psychology of traders – peoples expectations and reactions – a ect the process. Clearly other causes also a ect prices, for example: (a) supply and

demand, (b) external influences such as political or natural factors, e.g. an earthquake, (c) economic statistics and (d) technical considerations [1]. Not all of these can be modelled. In this paper, several types of agents are each represented by an LCS. However, different types receive different environmental messages. The agents do not necessarily communicate or interact directly with each other.

We have defined three arbitrary or ordinary types of traders, none of which is truly economical in the sense of maximising utilities of consumption (i.e., possessing a constant absolute risk-aversion utility function) like the agents of the Santa Fe Artificial Stock Market [3,19]. Instead, they apply their market forecasting hypotheses or mental models based on their knowledge of the stock's price, dividends and splits history in order to decide how to invest their money on a daily basis. The main results reported here use 10 years of historical data about the stock price of Merck & Co., available from `quote.yahoo.com`. As it happens, this performed very well over the period, and it is worth remembering that it is not hard for a competent trader to make money in such circumstances. We have run similar experiments using data about the stock price of Forest Oil which, like many in the oil business, fell fairly steadily instead. In our experiments each agent makes one decision (buy, sell or hold) per day, also specifying a size of trade expressed as a percentage of shares held (if selling) or of cash held (if buying).

The purpose of using real historical data is to concentrate on the dynamical behaviour of the different types of agents. As this is an on-going research, the next stage will be to allow the prices to be calculated endogenously in order to analyse the properties of the time series generated, and the dynamics of the agents' behaviours over time. For this we intend to use the kind of 'economic agent' developed in work on the SFI stock market simulation – see [3,17,18,19], among others. For recent experiments with the SFI stock market simulation, about the effect of changing the system's evolutionary learning rate, etc., see [13,14,15].

3 The Model

In the model used in this chapter, agents make decisions based on easily-obtained statistics such as various price moving averages, first differences in prices, volume ratios, etc. – but a different information set for each type of agent. This corresponds to the idea that traders react differently given the same information, they tend to take into consideration different facts about a current market situation. An agent's decision process is expressed as a set of classifier-type rules, bit-encoded; each condition bit expresses the truth or falsehood of a certain condition. The actual conditions used vary between agents, and are described later.

3.1 Market Structure

Time is discrete, indexed by t and represents one trading cycle equal to one day in real terms. Other components of the market are de ned in [22]. The model consists of:

1. Three heterogeneous **adaptive agents**. Note that in this chapter the stock price does not change according to supply and demand of the arti cial agent types, but rather by changes of real phenomena outside their scope.
2. One **non-intelligent agent** which represents the so called "Buy and Hold" strategy – in this case, buying only on day one (it is mainly a bull market). Many people believe that such a strategy is best in the long run. This semi-agent is of great relevance because we need to compare our traders performances against a strategy which works very well for highly performing stocks. But the fact that any real trader can decide to hold his/her possessions for a long period of time and that he/she is doing nothing with the shares in that time does not necessarily imply a lack of intelligence!
3. One **Bank** agent which represents the strategy of leaving all the money in the bank earning compounded interest – in this case at 8% p.a.
4. The **Information Set**. This is the available raw data about the market, as well as data which has been processed in various ways. It includes basic daily information about the stock such as its current price, volume of transactions, splits and dividends, and some derived information such as price di erences, moving averages, current standing of the buy-and-hold strategy and the bank investment. Although it would be simplest to give each di erent agent type the same information, in the present model it is also possible to choose to vary what an agent sees according to its nature. The agent types are distinguished from each other by the fact that each day, they receive di erent sets of information-environmental messages.
5. Two assets traded: a **risk free bond** with in nite supply paying a xed interest rate (the Bank), and a **risky stock**. For an endogenous price formation scenario, the stock will be issued in N units paying a stochastic dividend given externally, and for the exogenous price scenario that is mainy discussed here, the stock is in in nite supply, the only limitation is that the agent must be able to a ord the amount of shares it wishes to buy.

3.2 Decision Making

Although in general terms the goal of the arti cial agents is to maximise their pro ts by making investment decisions regarding their current portfolio, they form their expectations in very di erent ways. Each day, they all have a choice of (i) leaving their money in the Bank, in which case there is a xed interest rate paid on a daily basis equivalent to 8% annually, or (ii) they can buy or sell a stock, represented by Merck & Co.

Representation. The framework for representing the adaptive agents relies in the following LCS components, more throughly described in [10,11]:

1. The performance system, consisting of (i) detectors for extracting information from the environment, (ii) classi ers to represent the agent's capabilities for processing the environment and (iii) e ectors, through which the agent acts on its environment.
2. Credit Assignment Algorithm. For situations in which many rules re at the same time, it is responsible for strengthening rules that set the stage for later more rewarding activities.
3. Rule Discovery Algorithm, through which plausible hypotheses are generated and past experience is incorporated.

Learning and Rule Evolution. There are two levels of learning in the model:

The **rst level** happens rapidly as the agent learns which of his strategies are accurate and worth acting upon, and which ones should be ignored. This happens during the auction among currently matched classi ers in the apportionment of credit algorithm.

On each trading cycle zero or more rules can match the current state of the market, but only the best will be used in that period as it relates to a speci c market descriptor. The credit-distribution system used in the present model is a simpli ed bucket brigade algorithm in which a payment equal to the bid of the current winner is transferred to the old winner only, no chains are implemented because we are not interested in rewarding sequences of classi ers for more than one step. For each agent there exists an option of setting this payment distribution on or o .

The **second level** occurs after several trading periods have gone by, when the structure of these mental strategies is modi ed by the use of a GA which replaces some of the worst performing rules by new ones created with useful pieces of good rules.

In general terms, a genetic algorithm is activated to allow the agent to perform the following operations:

1. **Specialisation.** Transformation of general rules into more speci c rules.
2. **Diversi cation.** Introducing more heterogeneity in the rule set.
3. **Creation.** Adding new rules into the system when an incoming message does not match any rule.
4. **Generalisation.** Birth of o spring from parents with high tness through crossover.

A proportion of the population is selected for reproduction during a given genetic algorithm invocation. In the examples for the simulations shown in section number 5, this gure is usually set to 20% (as suggested in [8] when solving the six-multiplexer problem starting from a randomly generated set of 100 rules). Using roulette wheel selection, a pair of mates is selected, where crossover and mutation take place in the normal way and the replacement candidates are

chosen from a low-performance subpopulation. Since the search is for a well adapted set of rules, we use crowding replacement to choose the classi ers that die, inserting new o spring in their place on the basis of similarity. For more details on how the genetic algorithm works, refer to [8].

4 The Three Arbitrary Agents Model

In order to keep the model as simple as possible, we have introduced only three types of traders (that is, three agents) in the market and they all receive di erent sets of market information and compute their optimum portfolio by using di erent models. In what follows the three types (agents) are called **Tt1**, **Tt2**, and **Tt3**. One type can not evolve into another type, but it can go broke and thus halt while another one can get rich. It is of course possible for an agent to learn to ignore any particular eld in the daily market information it receives, but it cannot ask for extra elds beyond those given to it.

Each type of agent controls its market position by using and evolving a set of classi ers. These classi ers are used to match binary conditions in the market, which requires prede ning a set of binary states or environmental messages for each agent that can be used when making decisions. All three agent types receive the rst di erence in price (1 if the current price of the stock is higher than the previous day's, 0 otherwise), followed by a basic set of market information which di ers for each type of agent.

The rst set contains **price statistics** and **moving averages**, based on standard moving average types of trading rules, the second set basically contains **volume statistics**, and the third set incorporates apart from the price and volume, some other **news** such as the accumulated wealth of both the buy-and-hold strategy and the bank's, and a recollection of its past action. Only these simple environmental messages are reported here, because we want to study simple models rst. Later versions might, for example, allow every agent to make use of what each of the other agents chose to do on the previous day, or might allow a poorly performing agent to switch from seeking to improve its wealth to seeking to emulate another, better-performing agent but still using its own information set for the purpose.

For present purposes Tt1's state vector is seven bits long, Tt2's is ve, Tt3's is 6. Table 1 shows the meaning of each position for Tt1 and Tt2 during the current trading period t. The actual value of each cell is 1 if the condition is satis ed. For example, bit number 1 is 1 if the price today is higher than yesterday's: P_t is the price on day t, P_{MA5} is the ve-day price moving average, $P_{highest}$ and $V_{highest}$ are the highest price and volume for any day so far, etc.

Similarly, the third trader's market state vector is summarised in table 2. The third trader di ers from the others in a couple of ways. The rst one is that it receives both the di erence in price and the di erence in volume, with the idea that it will use this in conjunction with an additional piece of information, and that is "how" it stands in the market. For this we have given it a comparison of its daily accumulated shares with respect to the total shares owned by the the

Table 1. Environmental message for Trader Types 1 and 2

Trader type 1

Bit Number	Representation
1	$P_t > P_{t-1}$
2	$P_t > 1.2 * P_{MA5}$
3	$P_t > 1.1 * P_{MA10}$
4	$P_t > 1.05 * P_{MA20}$
5	$P_t > 1.025 * P_{MA30}$
6	$P_t > P_{highest}$
7	$P_t < P_{lowest}$

Trader type 2

Bit Number	Representation
1	$P_t > P_{t-1}$
2	$V_t > V_{t-1}$
3	$V_t > V_{MA20}$
4	$V_t > V_{highest}$
5	$V_t < V_{lowest}$

buy-and-hold investment, represented by $Shares_t$, as well as information about the accumulated wealth of its competitors: the buy-and-hold and bank. Note that it also receives its own previous action, so there is a "reflective" aspect to this trader. The other two types do not receive a note of their previous decision as part of the environmental message.

Table 2. Environmental message for Trader Type 3

Bit Number	Representation
1	$P_t > P_{t-1}$
2	$V_t > V_{t-1}$
3	$Shares_t >$ Buy and Hold $Shares_t$
4	$W_t > W_t$ Bank
5	$W_t > W_t$ Buy and Hold
6	Tt3 action at (t-1)

As usual classifier rules contain two parts: $< condition >:< action >$. The condition is a bit string matching the current market state vector which has a fixed length with each position taken any of the three values 0, 1 or #. The 1 and 0 match corresponding bits in the state vector, while # is a wild-card symbol which matches both 0 and 1. Wild-cards make it possible for an agent to learn to ignore some information.

The action is a bit string of length 4 indicating the decision of whether to buy, sell or hold possessions on the current day. The first bit represents a buy (1) or sell (0) signal, followed by three more bits which represent the percentage of available cash to use if buying, or the percentage of shares to sell if selling. Table 3 shows the mapping between bits 2-4 and a percentage. Buying or selling 0% corresponds to simply holding.

The factor s in this table is a scaling factor which increases linearly from 0.1 at day 0, in ten equal steps to 1.0 at day 2,000; without such scaling, a novice trader might be wiped out early on before having the chance to learn

Table 3. Transaction percentages

Bits 2-4	Percentage
000	0% (i.e. hold)
001	$15s\%$
010	$30s\%$
011	$45s\%$
100	$60s\%$
101	$75s\%$
110	$90s\%$
111	$100s\%$

about trends. Although economic and nancial models are mainly based on continuous real valued data, we believe that in this model it is not necessary to have to use continuous entities explicitly in the trader's rules. This binary-encoded information seems to be enough to capture the causal reasoning in such simpli ed models, and has the added bene t of being easy to process.

4.1 The Trading Process

Each agent buys, sells, or holds its possessions and adapts by receiving feedback from the changing environment by monitoring, updating and evolving new procedures. The idea is that agents therefore continually form individual, hypothetical expectational models of "theories of the market," test these, and trade on the ones that predict best. An agent might interpret a certain state of the market (events) according to its past performance. Thus there is a dynamical approach in the sense that future behaviour is a ected by past performance importing feedback (new observations) from the environment. This inductive way of reasoning is explained in more detail in [4].

A basic accounting procedure takes place for every agent. At the beginning of each trading cycle, the agent's holdings and cash accounts are updated, including stock splits, cash dividends and commissions for all transactions executed. If the trader decided to buy it must own all the cash incurred in the transactions, including the commission fee. When selling, the trader must own all the shares it wants to sell. Agents cannot borrow money or sell short. At the end of the day, the wealth of each trader is updated by adding the interest paid during one cycle to the cash account. The wealth $W_{i(t)}$ at time t of agent i is thus given by the equation:

$$W_{i(t)} = (1 + r)M_{i(t)} + H_{i(t)}p(t), \tag{1}$$

where $(1 + r)M_i(t)$ is the cash invested at an interest rate r and $H_{i(t)}p(t)$ are the holdings calculated at the current price of the stock $p(t)$.

5 Simulation of the Three Arbitrary Agents Model

The set of experiments of this section refer to the model using real data, the three agents Tt1, Tt2 and Tt3, and the non-intelligent "Buy and Hold" agent and the trivial "Bank" agent, all described earlier in section 3.1. We want to keep everything as real as it can be, therefore there is a commission C of 0.1% incurred for each transaction. The compound interest rate paid by the bank has been set to 8% annually. Tt1, Tt2 and Tt3 are able to buy, sell or hold their possessions as they wish. On day one they are given $10,000 (ten thousand US dollars). In addition to the initial cash, they could also be given a number of shares to start with.

The idea is that the apportionment of credit algorithm, along with the genetic algorithm should provide each individual agent with experience-based rules for buying, selling and holding. We want to test whether this holds for the proposed model.

5.1 The Adaptive Phase

In this first phase the simulation starts by randomly initialising the strategies of each agent, all with strengths values equal to 10. There are 100 classifier rules per agent, although they are designed to have any number of classifiers, completely independent of each other. The probability of having # symbols in classifier conditions is 0.5. The rest is filled equiprobably with 0s and 1s. The reason we set the probabilities this way is because we want to start with more general rules to match the given environmental states. The action part has no # symbols; bits are 0 or 1 with equal probability. The measure of performance of any given agent is the amount of money it accumulates through its actions and it is calculated at the end of each day.

The full set of data we used for the two sets of experiments corresponds to 10 years of real prices and volumes for the stock Merck & Co. It is divided in a 9 year training period – the adaptive phase – and a 1 year testing period, the non-adaptive phase. Splits and dividends declared by the company over that period are already discounted in the price series, so there are no jumps in prices due to splits or in wealth due to payment of dividends. For the first set of experiments, the random scenario, we only used the first 9 years of data. This set of data is taken as a training set, while the remaining year will be used later on for testing purposes. Nine years of financial data corresponds to 2,275 trading days. We ran several simulations with different seed values with three agents trading daily to select the best performance of each type recording the parameters used as well as the rule configurations evolved. The parameters for the three agents are shown in table 4. *GA period* refers to the number of generations or cycles that have to pass before the genetic action takes place. In all cases presented in this paper, when the GA is ON, 20% of the current population undergoes reproduction, crossover, mutation and replacement by crowding.

Automatch flag refers to a covering algorithm similar to the one suggested in [24], that creates a matching classifier when there is no current match and

Table 4. Parameters for Adaptive Simulation

Parameter	Tt1	Tt2	Tt3
GA period	200	100	100
Automatch Flag	ON	ON	ON
Specificity Flag	ON	ON	ON
Noise Flag	OFF	OFF	OFF
Bucket Brigade Flag	ON	ON	ON
Reinforcement	0.5	0.5	0.5
Number of shares	0	0	0
Initial Cash given	10,000	10,000	10,000
Commission per trade	0.1%	0.1%	0.1%
Annual Interest Rate	8%	8%	8%
Seed	0.9	0.95	0.9

inserts it in the population replacing the classi er with the lowest performance. The action is selected according to price behaviour (i.e. if the price today is higher than yesterday, the action is set to 1) and the proportion is selected randomly. The classi er's strength is chosen to be the larger of 10 and the average value of the population's strength. Additional experiments showed us that using the initial value of 10 in all cases did not improve performance; on several occasions, a non-matching situation was encountered when the average of strengths was well above the default value of 10 and the inserted classi er became one with very low strength and ended up disappearing, either because other non-matching classi er was inserted, or because the genetic algorithm would select it for replacement, forcing the system to make unnecessary re-insertions of the same classi er later on, when the same environmental message appeared again. Note that the number of classi er rules per agent is a xed quantity equal to 100, not a dynamic list because we want to develop the simplest possible type of agent, one able to generalise good enough, as well as to specialise by recognising more narrow market states. If we simply add to the population of rules new market states, we would end up with a very large set of speci c rules and the complexity of the exploration would increase. In all runs with Merck data the use of this automatch algorithm showed improvements.

The ags *speci city*, *noise* and *bucket brigade* refer to the apportionment of credit subsystem, where an auction takes place to select a winning classi er from the set of matched classi ers, as described in [8]. When the bucket brigade ag is on, a payment equal to the current winner's *bid* (amount proportional to the classi er's strength and speci city) is given to the previous winner. This corresponds to the *implicit bucket brigade* algorithm rst developed by S. Wilson in [23]. In general, our results show higher trader's performance when this ag is on. This is due to the fact that the payment received by the old winner balances the bid payments (and others such as taxes) it made; otherwise it could lose strength when its action might have actually contributed to more wealth.

And nally, *reinforcement* represents the amount of payment awarded to those classi ers that satisfy speci c market criteria. Our reinforcement scheme corresponds to a modi cation of Holland and Reitman's epochal credit allocation plan scheme, the *pro t sharing plan* (PSP) [12,9], where a constant fraction of the current reward is paid to each classi er that becomes active since the last receipt of reward. Instead of paying a constant fraction, we reward classi ers according to their *speci city*: a classi er's parameter which determines the number of non # symbols in its condition part. The more speci c the classi er is, the higher the percentage of the reinforcement reward it gets. Rationing the reward payments this way allowed us to get a good distribution of low speci city rules to cover more general market scenarios with respect to more speci c rules to cover the more speci c cases. In addition to this, classi ers are paid if and only if they satisfy certain conditions. There are a number of market characteristics to consider when designing the reward scheme. We have tried many di erent criteria, i.e. the simplest criterion corresponds to paying a reward to all matched classi ers whose action (yesterday) was a decision to buy when today's price is 2% higher than yesterday's price or 2% higher than the last 6-week moving average. Another criterion rewards classi ers that suggested holding when the actual price uctuates 20% above or below the last 6-week moving average. Note that the reward payment is made once the current price of the stock is known, i.e. rewarding the classi ers that matched on the previous day AND satis ed the given criterion.

Figure 1 shows, in the left-hand graph, the wealth of Tt1 and of the bank and buy-and-hold strategies over the 9-year training period. Even though the agent starts with randomly selected classi ers, it manages to outperform both of the strategies shown. As it can be seen, at around day 700 there was a two year bear market, in which period the arti cial agent started to outperform the buy-and-hold strategy by far. It is at this point when the agent found rules that proved to be pro table such as 11##1#0, *buy when today's price is higher than yesterday's AND higher than 120% of last week's moving average AND higher than 102.5% of the last 6 weeks' moving average AND is not lower than the lowest price seen* and so forth. The right-hand graph in gure 1 shows the number of shares the agent owns during the same period of time against the buy-and-hold strategy. The biggest oscillations in shares happened around day 1,000, exactly when its wealth outperformed the buy-and-hold strategy. It is on this gure where we can get an idea of the agent's trading volume. Recall that a commission is paid by the agent for every buy/sell transaction. These particular results of wealth correspond to a single run. However, runs using di erent random number seeds show considerable di erences in performance.

Figure 2 shows the same information for Trader Type 3. Around day 1,000, it starts pro ting from selling shares when the price is dropping. In terms of share dealing, Tt3 also shows an increase in transactions around day 1,000, but in addition to this, there is another period of oscillations around day 2,000. By looking at the oscillations in shares owned we can see the di erent trading behaviours displayed by Tt1 and Tt3.

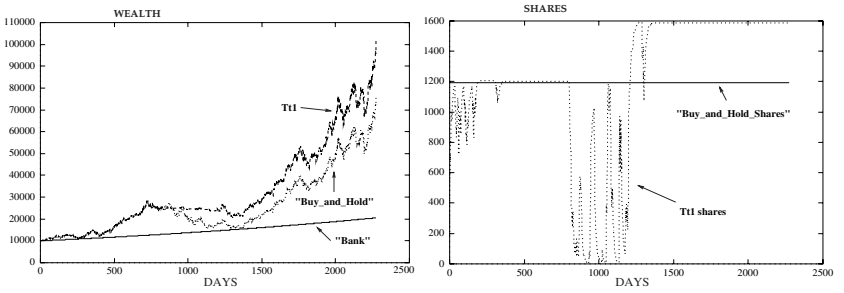


Fig. 1. Wealth and shares held for Tt1

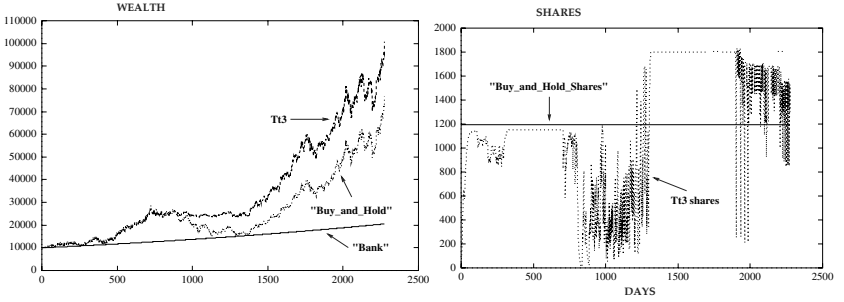


Fig. 2. Wealth and shares held for Tt3

Taking a closer look at the trader's wealth and shares during the period of days 700 to 1,300, in figure 3 we can see more clearly how the bank slowly but steadily increased its wealth, while the buy-and-hold strategy dropped significantly due to exogenous market forces, but Tt3 managed to get out of this downward trend not only by selling its current owned shares, but also by profiting from ups and downs thanks to an increase in its daily transactions. At the beginning of the period, the agent had in possesion fewer shares than the buy-and-hold strategy, but by trading accurately, it ended up having 50% more shares than the buy-and-hold strategy.

A more detailed description of the trader's actions is shown in figure 4. The letters next to each data point (S, B, H) indicate when the agent decided to sell, buy or hold. Pleasingly, the trader's buying decisions dominate at bottom prices, while the selling decisions dominate at peak prices. Note that such decisions are based on the trader's expectations about the future price. These trading behaviours are emerging properties of the system rather than explicit trading rules incorporated into the trader's pool of beliefs either by an expert's advice or by past observation, this is still the training period.

A specific examination of which rules were showed that agents do indeed learn to buy on the upward trend and sell on the downward trend, although the precise details vary from agent to agent. In particular, space does not allow us to discuss what each agent regards as a trend. The scaling factor s that limits the size of

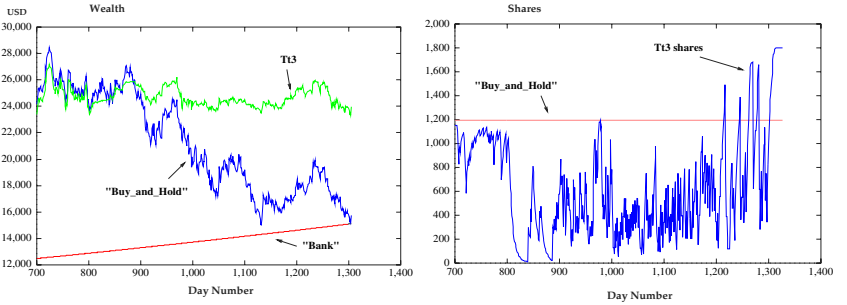


Fig. 3. Wealth and shares for Tt3 from day 700 to 1,300

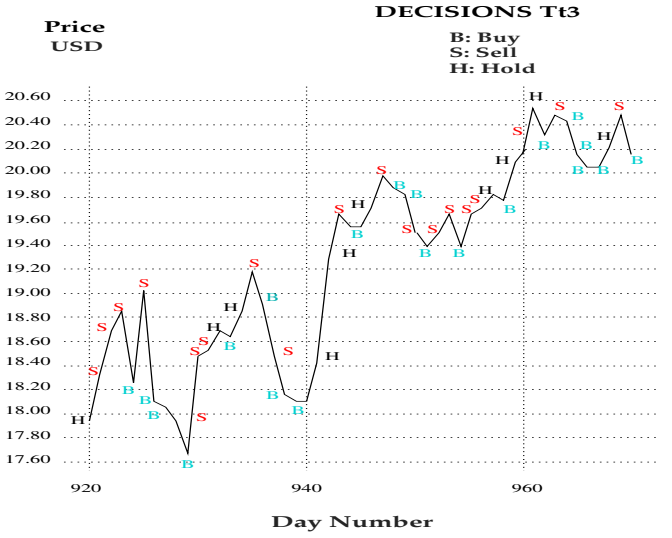


Fig. 4. Daily Decisions made by Trader Type 3 from day 920 to 970. Buy, Sell and Hold correspond to S, B and H respectively

early transactions (see table 3) is the reason why the right-hand graph in gure 2 shows smaller numbers of shares traded at the beginning. But these oscillations gradually increase until around day 2,000, when the swings of share transactions ranges from 200 to 1,800 in one trading cycle.

Table 5 shows the initial and nal wealth of the bank, buy-and-hold, Tt1, Tt2 and Tt3 in runs with the parameters previously shown. Note that the highest return corresponds to Tt1, followed by Tt3, then Tt2, the buy-and-hold strategy and nally the bank investment.

Table 5. Returns of all Agents from Adaptive Simulation

2,275 trading days	Bank	B&H	Tt1	Tt2	Tt3
Initial Capital	10,000	10,000	10,000	10,000	10,000
Final Capital	20,546	76,733	102,180	82,006	100,755
% Increase w.r.t. Bank	0	273	397	299	390
% Increase w.r.t. buy-and-hold	-73	0	33	7	31
% Return w.r.t. initial investment	105	667	922	720	908

5.2 The Non-adaptive Phase

The purpose of this phase is to test whether the evolved agents can then continue to trade reasonably successfuly for a further calendar year. The parameters for this simulation are shown in table 6. All classi ers start with strength equal to 10. The simulation runs for a total of 253 trading days, equivalent to one calendar year, starting the day immediately after the previous phase ended.

Table 6. Parameters for Non-Adaptive Simulation

Parameter	Tt1	Tt2	Tt3
GA period	OFF	OFF	OFF
Automatch Flag	on/off	on/off	on/off
Specificity Flag	ON	ON	ON
Noise Flag	OFF	OFF	OFF
Bucket Brigade Flag	ON	ON	ON
Reinforcement	0.5	0.5	0.5
Number of shares	0	0	0
Initial Cash given	10,000	10,000	10,000
Commission per trade	0.1%	0.1%	0.1%
Annual Interest Rate	8%	8%	8%
Seed	N/A	N/A	N/A

Again, the left-hand graph in gure 5 shows the wealth of Trader Type 1 against the wealth of the bank and buy-and-hold strategies, followed by the shares distribution in the right-hand graph. We have drawn a square around day number 2,425 in both graphs to show that the kind of decisions taken by the agent at that speci c time contributed to its accumulation of wealth. There are three drops in shares that correspond to selling at peak or near peak prices.

Figure 6 shows the decisions taken by the agent during some of those days. Most decisions were a signal to buy because by this time the price of the stock is higher than normal. The trader has learned that it corresponds to a better investment than leaving the money in the bank. But eventhough it buys most of the time, it decides to sell just before the price is about to drop. These

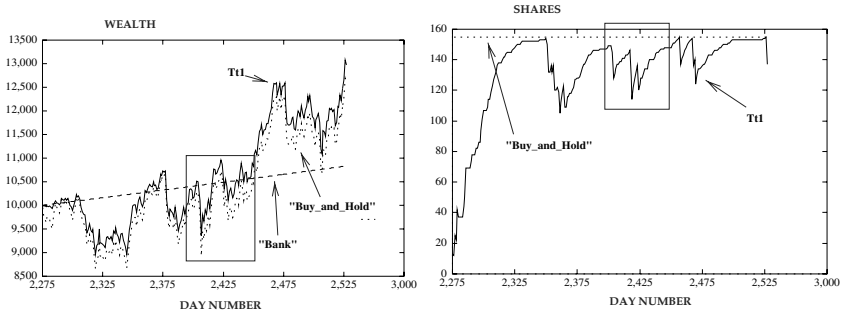


Fig. 5. Wealth and shares of Tt1 over the test set

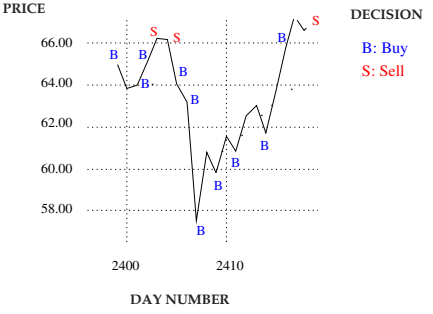


Fig. 6. Transactions Performed by Trader Type 1 over a segment of the Test Set

transactions are the most relevant for its accumulation of wealth during the test period.

Summarising the results of this non-adaptive phase, table 7 shows the initial and final wealth of the bank investment, buy-and-hold strategy and Tt1, for runs with the parameters previously shown. During this testing phase, the artificial trader had acquired a useful set of trading models that uses accordingly to outperform the bank and the buy-and-hold strategy.

Runs with this same testing set of data but turning the GA on (invoking it every 20, 50, 80 and 90 out of the 253 days) are shown in table 8. The first column refers to results previously shown with the GA turned off. These figures reflect only the averages over 11 runs with different seed values in the range of 0.0 to 1.0, increasing in steps of 0.1. Although results appear more detrimental because "on average" performance drops when adding further learning to the set of rules already learned, taking a closer look at the individual performances with each seed, we were able to find the highest performing element (with GA every 50 generations) found so far over this data set, which represents a 23% increase over bank, and 6% increase over buy-and hold strategy.

Table 7. Returns of Tt1 from Non-Adaptive Simulation

253 trading days	Bank	B&H	Trader
Initial Capital	10,000	10,000	10,000
Final Capital	10,834	12,587	12,968
% Increase w.r.t. Bank	0	16	20
% Increase w.r.t. buy-and-hold	-14	0	3
% RETURN w.r.t. initial investment	8.34	25.87	29.68

Table 8. Returns of Tt1 from Adaptive-Adapted Simulation

253 trading days	GA 20	GA 50	GA 90	GA OFF
% Increase w.r.t. Bank	8	17	19	20
% Increase w.r.t. buy-and-hold	-7	1	3	3

5.3 Adaptive vs. Non-adaptive

In order to analyse separately the learning due to apportionment of credit among the original (random start) rules and that due to the injection of new rules by the GA, in table 9 we show comparisons in average performances of runs with various values of GA invocation (GA ON) against runs in complete absence of genetic action (GA OFF). In these experiments we used the original 9-year set of data, starting with 100 random classi ers, which should be able to contain a moderate subset of rules (although not very rich for this di cult problem).

Table 9. Testing the relevance of the GA as a rule discovery heuristic

2275 trading days	GA 50	GA 75	GA 100	GA 200	GA 500	GA 1000	GA OFF
% Increase w.r.t. Bank	163	185	155	129	125	90	113
% Increase w.r.t. buy-and-hold	-30	-24	-32	-39	-40	-49	-43

Results with the GA OFF indicate how the apportionment of credit algorithm adjusts the strength values, and those with GA ON indicate the e ect of evolving new rules, which indeed is bene cial, except in the case in which the GA is invoked every 1,000 iterations. This means that it is only called 2 times during the total number of days, so when it is called, it only does a little damage to the rules. All the other cases show an improvement in average performance.

As mentioned before, 20% of the current population of 100 classi ers undergoes reproduction, crossover, mutation and replacement by crowding. This

means, for example, that the population of rules with a GA invoked every 100 cycles is turned over 4.55 times in 2,275 iterations.[1]

5.4 Variable Sized E ect on the Market

In this section we consider the dynamics of the di erent trader types in an adaptive single run over the nine year training period of data. Note that in the simulations previously shown, prices are real, not created endogenously. So, instead of analysing the price dynamics, we will analyse the dynamics of the trader types. Their actions do not have an e ect on the market, but the market has an e ect in the composition of their strategies and in their accumulation of wealth.

There are three examples shown in table 10. The rst run uses seed 0.9 and GA period of 100. The second one, seed 0.95, GA period 100 and the third run the GA period is 200, seed 0.9. Other parameters are the same as in table 4. The values under Tt1, Tt2 and Tt3 indicate the wealth of each agent as a percentage of the total wealth of the market at the beginning and at the end of the 9 year period - rst and last day. By looking at the impact of the total wealth that each type of agent represents in the market, we can easily infer that this proportion varies from time to time.

At the beginning of the simulation each type represents 1/3 of the total wealth of the market (each type is given the same amount of money, equal to $10,000 dollars), while at the end of the run their share of the market has obviously changed. For example, Tt1 represents 22%, Tt2 22% and Tt3 56%. It is clear that the behaviour of Tt3 gets to be more signi cant than the others as time progresses. The most successful will have a greater e ect on the market because the more wealth an agent has, the more transactions it can a ord, thus increasing its impact on the market. The opposite e ect can be shown in the experiment with seed 0.95, where Tt1 ends up representing 31% of the market, Tt2 goes up to 55% and Tt3 dramatically decreases its share in the market to 14%. On the third run Tt1 dominates, and Tt3 decreases from the starting 33% to 22%. These results correspond to the idea that if a real trader is successful, others will tend to emulate his/her approach increasing the impact of that type on the market.

Table 10. Percentage of Total Wealth at time t on three single runs

Seed	Day	Tt1	Tt2	Tt3
	0	33	33	33
0.90	2,275	22	22	56
0.95	2,275	31	55	14
0.90	2,275	46	33	21

[1] (0.2)(100)(2,275)/100 = 455 new offspring

6 Conclusions

All three types of non-economically oriented traders were able to survive and furthermore, outperformed the bank and buy-and-hold strategies in both phases: adaptive and non-adaptive.

Thus, Learning Classi er Systems seem to be able to at least represent competent traders. A reasonable conclusion is that the agents were successful in nding pro table rules exploiting the market inertia, for example, making money by selling late in a bull market and by buying in a bear market, or by selling while the price is dropping until the trend changes and it starts buying.

From the experiments shown it is also interesting to note that although each agent is independently using reinforcement learning techniques to optimise its own environmental reward, there is a global incentive to trade without having any explicit or implicit information sharing between the agents. Technical trading is a valid outcome in this market. With upward/downward trends in price there is in general, a collection of wise decisions leading at the end to a very pro table situation. Also, informed agents prevail. Speci cally, the second type of trader, who decides on his investment based on a few items derived from the share price and the current volume of transactions, is able to end the period with an increased wealth.

This paper describes work in progress. We are currently working on implementing the variant in which prices are determined endogenously. It would then be feasible to add more features to this simplistic economy such as having more instruments to trade or even more markets interacting.

At this point, key questions to address regarding the possible nal results obtained from this model involve, rst, the claim that such a system can be used to model certain market phenomena i.e., by generating features found in real markets, such as occasional price swings. Second, there is still the question of whether a system of agents can create behaviour that emulates real nancial data either qualitatively or in ne detail, by using evolutionary methods to con gure the parameters suitably. Third, given that some historical data can be emulated, there is the question of whether it would be possible to continue to track the data into the near future by means of simple perturbations of the model, which can be found in a simple neighbourhood search.

Acknowledgements. The rst author would like to thank Consejo Nacional de Ciencia y Tecnolog a (CoNaCyT) and Fundacion Mexicana para la Educacion, la Tecnolog a y la Ciencia, A.C. (FUNED) for nancial support.

References

1. What determines prices? *Applied Derivatives Trading*, July 1997. In Beginners Guide.
2. T. Ankenbrand and M. Tomassini. Agent based simulation of multiple financial markets. *Neural Network World*, 4:397–405, May 1997.

3. W. B. Arthur, J. H. Holland, B. LeBaron, R. Palmer, and P. Tayler. Asset pricing under endogenous expectations in an artificial stock market. Working Paper 96-12-093, Santa Fe Instituite, December 1996.
4. W. Brian Arthur. On lerning and adaptation in the economy. Working paper 92-07-038, Santa Fe Institute, 1992.
5. W. Brian Arthur. Inductive reasoning and bounded rationality. *American Economic Association Papers and Proceedings*, (84):406–411, 1994.
6. Michael de la Maza and Deniz Yuret. Experimenting with a market simulation. *The magazine of Artificial Intelligence in Finance*, 1(3), 1994.
7. Guido J. Deboeck. *Trading on the Edge. Neural, Genetic and Fuzzy Systems for Chaotic Financial Markets.* John Willey & Sons, Inc., New York, NY, 1st edition, 1994.
8. David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, MA., 1989.
9. John J. Grefenstette. Credit assignment in rule discovery systems based on genetic algorithms. *Machine Learning*, 3:225–245, 1988.
10. John H. Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, 1992.
11. John H. Holland. *HIDDEN ORDER. How Adaptation Builds Complexity.* Perseus Books, Reading, MA., 1995.
12. John H. Holland and J. S. Reitman. Cognitive systems based on adaptive algorithms. In D. A. Waterman and F. Hayes-Roth, editors, *Pattern-directed inference systems.* New York: Academic Press, 1978. Reprinted in: Evolutionary Computation. The Fossil Record. David B. Fogel (Ed.) IEEE Press, 1998. ISBN: 0-7803-3481-7.
13. Shareen Joshi and Mark A. Bedau. An explanation of generic behavior in an evolving financial market. Working paper 98-12-114, Santa Fe Institute, 1998.
14. Shareen Joshi, Jeffrey Parker, and Mark A. Bedau. Technical trading creates a prisoner's dilema: Results from an agent-based model. Working paper 98-12-115, Santa Fe Institute, 1998.
15. Shareen Joshi, Jeffrey Parker, and Mark A. Bedau. Financial markets can be at sub-optimal equilibria. Working paper 99-03-023, Santa Fe Institute, 1999.
16. Blake LeBaron. Agent based computational finance: Suggested readings and early research. Working paper, Graduate School of International Economics and Finance, Brandeis University, October 1998.
17. Blake Lebaron, W. Brian Arthur, and R. Palmer. The Time Series Properties of an Artificial Stock Market. *Journal of Economic Dynamics and Control*, pages 1487–1516, 1999.
18. R. G. Palmer, W. Brian Arthur, John H. Holland, and Blake LeBaron. An artificial stock market. *Arfif Life Robotics*, 3:27–31, 1999.
19. R. G. Palmer, W. Brian Arthur, John H. Holland, Blake LeBaron, and P. Tayler. Artificial economic life: a simple model of a stockmarket. *Physica D*, D 75:264–274, 1994.
20. Vijay Rajan and James R. Slage. The use of artificially intelligent agents with bounded rationality in the study of economic markets. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI'96)*, volume 1, pages 102–107. AAAI Press, The MIT Press, August 1996.
21. Bryan R. Routledge. Artificial selection: Genetic algorithms and learning in a rational expectations model. Technical report, Faculty of Commerce and Business Administration, University of British Columbia, November 1994.

22. Sonia Schulenburg and Peter Ross. An evolutionary approach to modelling the behaviours of financial traders. In *Genetic and Evolutionary Computation Conference Late Braking Papers*, pages 245–253, Orlando, Florida, 1999.
23. S. W. Wilson. Knowledge growth in an artificial animal. In John J. Grefenstette, editor, *Proceedings of the 1st International Conference on Genetic Algorithms and their Applications (ICGA85)*, pages 16–23, Pittsburgh, PA, July 1985. Lawrence Erlbaum Associates.
24. Stewart W. Wilson and David E. Goldberg. A Critical Review of Classifier Systems. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 244–255. Morgan Kauffmann, 1989.

The Fighter Aircraft LCS: A Case of Different LCS Goals and Techniques

Robert E. Smith[1], B.A. Dike[2], B. Ravichandran, A. El-Fallah, and R.K. Mehra[3]

[1]The Intelligent Computing Systems Centre,
The University of The West of England, Bristol, UK
robert.smith@uwe.ac.uk
2 The Boeing Company, St. Louis, Missouri
bruce.a.dike@boeing.com
[3] Scientific Systems, Woburn, MA
{ravi, adel, rkm}@ssci.com

Abstract. A system employed by the authors to acquire novel fighter aircraft manoeuvres from combat simulation is more akin to the traditional LCS model than to more recent systems. Given the difficulties often experienced in LCS research on simple problems, one must ask how a relatively primitive LCS has had consistent success in the complex domain of fighter aircraft manoeuvring. This paper presents the fighter aircraft LCS, in greater detail than in previous publications. Positive results from the system are discussed. The paper then focuses on the primary reasons the fighter aircraft LCS has avoided the difficulties of the traditional LCS. The authors believe the system's success has three primary origins: differences in credit assignment, differences in action encoding, and (possibly most importantly) a difference in system goals. In the fighter aircraft system, the goal has been simply the discovery of *innovative, novel* tactics, rather than online *control*. The paper concludes by discussing the most salient features of the fighter aircraft learning system, and how those features may be profitably combined with other LCS developments.

1 Introduction

This paper reports the authors' ongoing experience with a system for discovering novel fighter combat maneuvers, using a genetics-based machine learning process, and combat simulation. Despite the difficulties often experienced with LCSs, this complex, real-world application has proved very successful. In effect, the adaptive system is taking the place of a test pilot, in discovering complex maneuvers from experience. The goal of this work is distinct from that of many other studies, *in that innovation, and discovery of novelty, is, in itself, valuable.* This makes the details of aims and techniques somewhat distinct from other LCSs.

The paper discusses problems with past LCSs, recent innovations, the details of the fighter aircraft application, and the motivations and details of the techniques employed in the LCS used in this application. In many ways, this system is more similar to more traditional LCSs, than to more advanced, more recent systems. The paper reveals that, although the fighter combat LCS could likely benefit from many recent LCS innovations, it performs successfully, given the goals of its application.

P.L. Lanzi, W. Stolzmann, and S.W. Wilson (Eds.): LCS '99, LNAI 1813, pp.283-300, 2000.

The paper presents results from the fighter combat LCS, both where one player adapts to a fixed strategy opponent, and where two players co-adapt. Reasons for the system's success are explored. General implications of this work for other LCSs are discussed.

2 Issues and Problems with LCSs

In a paper presented at the First International Workshop in LCSs, Booker [1] points out an important part of clarifying the details and concerns of LCS research. His comments are paraphrased here. The LCS is usually described as a *method*: a set of algorithmic details that define how a way to solve a class of problems. However, in many ways the LCS is more of an *approach*: a set of conceptual details that define a certain direction for developing methods. Therefore, the defining issues for the LCS are not necessarily algorithmic, but conceptual. The central problem addressed by workshop's discussions was to clarify these defining, conceptual issues. The workshop's conclusions on these defining issues emerged from a "reverse engineering" process: discussion focused on central technical issues that arise in LCS research, and outlining these issues helped to categorize key LCS concepts. Technical questions included:

- What methods can create cooperation in an LCS, despite the competitive nature of LCS and GA mechanics (e.g., covering a set of possible inputs; performing a tiling that yields an effective input output mapping; forming a set of structures that combine to form a action; forming a parsimonious rule set through default hierarchies; forming sequences of actions through rule chains; etc.)?
- How can one ensure effective credit assignment in an LCS?
- How can the LCS create computationally useful internal message processing in an LCS (to allow LCSs to deal with non-Markovian environments)?
- What is an appropriate syntax or representation in an LCS?

Behind these questions lay a number of typical difficulties that occur in practice with LCSs. These difficulties include [20]:

- Classifier populations can sometimes be "taken over" by rules that have high strength (i.e., high expected payoff), despite the fact that they are supplanting other (lower strength) rules upon which overall system performance depends.
- This problem can be coupled to difficulties in appropriate credit assignment. LCSs often have both complex *structural* credit assignment problems (i.e., balancing credit in default hierarchies) and *temporal* credit assignment problems (i.e., how should credit be propagated through rule chains).
- As a result of structural credit assignment problems (and possible LCS syntactical difficulties) over-general rules often take over populations inappropriately.
- Attempts at the use of internal messages for non-Markovian problems seems to simply aggravate these problems, and lead to problems of *parasite rules.* These are rules that benefit from indirect payoff through temporal credit assignment, yet cause the eventual degradation of system performance.

A common outcome of all of these problems is a "rise and crash" behavior in LCSs. LCSs will often evolve useful, high performance rule sets, then lose key rules through inappropriate takeover of other rules. Sometimes the system will re-evolve useful rule sets, and repeat the pattern.

3 Recent LCS Advances

The observed problems above have led to a number of modifications to the traditional LCS. Many of these are best illustrated by Wilson's XCS [14,15]. This system is discussed below.

In the XCS, the goal of genetic learning is more strictly clarified by a framework of reinforcement learning [11]. Specifically, consider the mapping from state/action pairs to payoff, $(s,a) \rightarrow P$ (where payoff P includes both reward r and appropriate temporal credit assignment). XCS focuses genetic learning on acquiring accurate generalizations over this "payoff landscape". In XCS this is accomplished by coupling genetic learning to Q-learning, a standard reinforcement learning technique [11,12]. Given an accurate, compressed representation of the payoff landscape, and the (approximate) assurance of correct Q values, one can follow a greedy strategy with respect to these values, and can have some confidenc of Bellman-optimal reinforcement learning control. Note that there is no concrete assurance of convergence to an optimal strategy in XCS, but that no such assurance is available in any reinforcement learning scheme that employs generalization over the state-action space. However, XCS has empirically shown convergence to optimality in several complex environments.

Given this clarification of the goal of the "LCS approach" within XCS, an appropriate set of techniques are adopted. Chief amongst these are:

- The use of an *accuracy* measure for classifier fitness, rather than the traditional *strength* measure. While strength is an estimator of the expected payoff for a classifier, accuracy is related to the variance of classifier's payoff. Low variance (high accuracy) classifiers are considered to be good generalizations over the payoff landscape. Therefore, they are likely to reflect accurate Q-values.
- The use of a non-panmictic GA (that is, a GA that *does not* act across the entire classifier population). In early versions of XCS, the GA was applied only to classifiers in the *match set* (those that match the current environmental message). In later versions the GA was applied only to classifiers in the action set (those that match the current environmental message *and* suggest the same action). This adds selective pressure for the GA to form maximally general classifiers.

In XCS, the effects of these two techniques act in balance to form maximally general, but simultaneously maximally accurate classifiers. This should help to overcome some of the "rise and crash" problems that are typical of LCSs, as noted in Section 2

In XCS (and other, more recent LCS efforts) the nature and goals of the LCS approach are clarified in the context of reinforcement learning control. This addresses most of the questions discussed in Section 2.

Other issues are clarified in XCS as well. Appropriate credit assignment is related to the reinforcement learning control framework. The nature of appropriate cooperation of classifiers is also clarified in two senses: cooperation to form accurate generalizations over the payoff landscape, cooperation through time, via the Bellman optimality principles of reinforcement learning techniques (i.e., *Q*-learning). Other LCS questions (e.g., non-Markovian environments and alternate classifier syntax) are also being addressed within the framework of XCS [15].

XCS clarifies many of the questions associated with the LCS in terms of reinforcement learning control. However, one should also consider how these questions can be clarified in other contexts. The following sections consider another successful LCS, and it's implications for future LCS research.

4 Another LCS

The authors have extensive, ongoing experience with using LCSs for acquiring novel fighter aircraft maneuvers. This is a successful, ongoing project, satisfying real-world goals for industry, NASA, and The United States Air Force [8,10]. Despite the problems outlined in Section 2, this project has used a system that is quite similar to the traditional LCS model. It is important to note that this system is an outgrowth of work that began in 1992, and many of its features were designed incrementally, based on experience, for a particular task. The system is likely to benefit from many of the advances in LCS theory and practice discussed in Section 3. This is a part of our ongoing work. However, given the current system's success, the goals and techniques of this system merit further consideration.

4.1 The Problem

By way of introduction, consider the basic problem of one-versus-one fighter aircraft combat. Two aircraft start their engagement at some initial configuration and velocity in space. In our simulations, an engagement lasts for a pre-specified amount of time (typically 30 seconds). An engagement is divided into discrete time instants (in our simulation, $1/10^{th}$ second instants). At each instant, each "aircraft" must observe the state of the environment, and make a decision as its own action for that instant. A score for a given aircraft can be calculated at the end of an engagement by comparing that aircraft's probability of damaging its opponent to its own probability of being damaged.

Given this basic outline of the problem at hand, we will introduce details of the fighter aircraft LCS.

4.2 The Combat Simulation

The LCS interacts in simulated, 1-versus-1 combat, through AASPEM, the Air-to-Air System Performance Evaluation Model. AASPEM is a U.S. Government computer simulation of air-to-air combat, and is one of the standard models for this topic.

4.3 Detectors and Classifier Conditions

The conditions of the classifier are from the traditional {1, 0, #} alphabet, defined over the encoding of the environmental state shown in Table 1.

Table 1. Encoding of Environmental State in the Fighter Aircraft LCS.

	Bins (represented in binary, plus the # character)							
3 bits:	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**
Own Aspect Angle (Degrees)	< 45	45 to 90	90 to .35	135 to 180	180 to 215	215 to 270	270 to 315	315 to 360
Opponent Aspect Angle (Degrees)	< 45	45 to 90	90 to 135	135 to 180	180 to 215	215 to 270	270 to 315	315 to 360
2 bits:	**00**	**01**	**10**	**11**				
Range (1000 feet)	< 1	1 to 4.5	4.5 to 7.5	> 7.5				
Speed (100 knots)	< 2	2 to 3.5	3.5 to 4.8	> 4.8				
Delta Speed (100 knots)	< -.5	-.5 to .5	.5 to 1	> 1				
Altitude (1000 feet)	< 10	10 to 20	20 to 30	> 30				
Delta Altitude (1000 feet)	< -2	-2 to 2	2 to 4	> 4				
Climb Angle	< -30	-30 to 30	30 to 60	> 60				
Opponent Climb Angle	< -30	-30 to 30	30 to 60	> 60				
Total: 20 bits								

Note that although this encoding is *crisp*, its coarseness, and the open-ended bins sometimes used in the encoding, have a linguistic character that is similar to that of a *fuzzy* encoding. That is, concepts like "high, low, and medium" are embodied in the encoding.

4.4 Effectors and Classifier Actions

The classifier actions directly fire effectors (there are no internal messages). Actions are from the traditional {1,0,#} syntax, and correspond to the coarse encoding shown in Table 2.

Note that these action values are only *suggested* to the system that simulates the fighter combat. In many situations, it may be impossible for the aircraft to obtain the exact value suggested by the active classifier. Fortunately, the nature of the underlying fighter combat simulation returns an aerodynamically feasible setting of these action parameters, regardless of conditions. This feature of the system is discussed in more detail later.

Table 2. Action encoding in the Fighter aircraft LCS.

	Action Values (suggested to AASPEM)							
3 bits:	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**
Relative Bank Angle (Degrees)	0	30	45	90	180	-30	-45	-90
Angle of Attack (Degrees)	0	10	20	30	40	50	60	70
2 bits:	**00**	**01**	**10**	**11**				
Speed (100 Knots)	1	2	3.5	4.8				
Total: 8 bits								

As an example of the encoding, consider the classifier shown below:

```
01010010110111010010 / 11010010
```

This classifer decodes to:

```
IF
          (90 > OwnAspectAngle > 35) AND
          (215 > OpponentAspectAngle > 180) AND
          (7.5 > Range > 4.5) AND
          (Speed > 4.8) AND
          (0.5 > DeltaSpeed -0.5) AND
          (Altitude > 30) AND
          (2 > DeltaAltitude > -2) AND
          (ClimbAngle < -30) AND
          (60 > OpponentClimbAngle> 30)
THEN ATTEMPT TO OBTAIN
          (RelativeBankAngle = -45) AND
          (AngleOfAttack = 40) AND
          (Speed = 3.5)
```

4.5 Match-and-Act

In our system if no classifiers are matched by the current message, a default action for straight, level flight is used. There is no "cover" operator [21].

4.6 Credit Allocation

- At the end of an engagement, the "measure of effectiveness" score for the complete engagement is calculated (see below).
- This score is *assigned* as the fitness for *every* classifier that acted during the engagement (and to any duplicates of these classifiers).
- Note that this score *replaces* the score given by averaging the parent scores when the GA generated the rule. Thus, rules that do not fire simply "inherit" the averaged fitness of their GA parents [9].

4.7 Measures of Effectiveness

Our efforts have included an evaluation of different measures of effectiveness within the genetics-based machine learning system, to determine the relative sensitivity of the process. Initial candidate measures included exchange ratio, time on advantage, time to first kill, and other relevant variables.

The measure of effectiveness ultimately selected to feedback into the GA fitness function was based on the following steps. The base score was based on a linear function of average angular advantage (opponent target aspect angle minus ownship target aspect angle) over the engagement, as shown in Fig. 1.

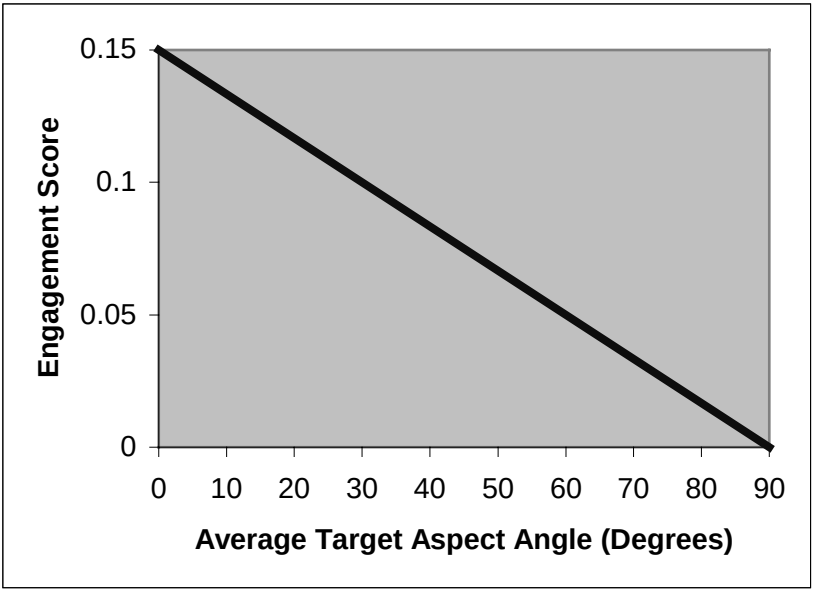


Fig. 1. Measure of effectiveness, which is used as an input to the GA fitness function.

To encourage maneuvers that might enable gun firing opportunities, an additional score was added when the target was within 5 degrees of the aircraft's nose. A tax was applied to non-firing classifiers to discourage the proliferation of parasite classifiers that contain elements of high-performance classifiers but have insufficient material for activation. All non-firing classifiers that were identical to a firing classifier were reassigned the firing classifier's fitness.

4.8 GA Activation

- The GA acts at the end of each 30-second engagement.
- The GA is panmictic (it acts over the entire population).
- In some of our experiments, the *entire* classifier list is replaced each time the GA is applied. This has been surprisingly successful, despite the expected disruption of the classifier list. In recent experiments, we have used a generation gap of 0.5

(replacing half of the classifier population with the GA). This is still a substantially larger portion than are replaced in many LCSs.

- A new, GA-created classifier is assigned a fitness that is the average of the fitness values of its "parent" classifiers.
- The GA used employed tournament selection, with a tournament size ranging from 2 to 8.
- Typical GA parameters are a crossover probability of 0.95, and a mutation rate of 0.02 per bit position. When a condition bit is selected for mutation, it is set to one of the three possible character values (1, 0, or #), with equal probability. Note that this actually yields an effective mutation probability of (0.02)(2/3)=0.0133. Children rules replaced randomly selected rules in the population.

4.9 Conflict Resolution

The matching rule with the highest fitness/strength is selected to act *deterministically*.

4.10 Combat Simulation Starting Conditions

A two-tier approach was employed for defining run conditions and learning system parameters. First, a baseline matrix of initial positions, relative geometries, and energy states was identified in conjunction with NASA requirements. The primary source document for this step was the X-31 Project Pinball II Tactical Utility Summary, which contained results from manned simulation engagements conducted in 1993 at Ottobrunn, Germany [16]. Initial findings from the X-31 Tactical Utility Flight Test conducted at Dryden Flight Research Center were also used to compare with results from this project. The baseline, starting condition, test matrix, as shown in Fig. 2, was based on X-31 manned simulation and flight test conditions, and was tailored to the X-31 performance envelope, flight test range constraints, and other testing considerations.

Note that each of these represents a separate initial condition for a fighter combat simulation. Each learning run consists of repeated engagements, all starting from the same one of these conditions.

The first four start conditions (Defensive (DEF), Offensive (OFF), Slow-Speed Line Abreast (SSLA), and High-Speed Line Abreast (HSLA)) were derived directly from the Pinball II project. The fifth start condition, High Speed Head-On Pass (HSHP), was added to the matrix to provide an additional geometry which would not exclusively result in a close turning fight. The opponent aircraft was an F/A-18. The baseline matrix formed a set of core conditions to generate X-31 tactics results for a balanced cross-section of tactically relevant conditions. The test conditions specified the initial geometries, X-31 and opponent speeds, altitudes and ranges.

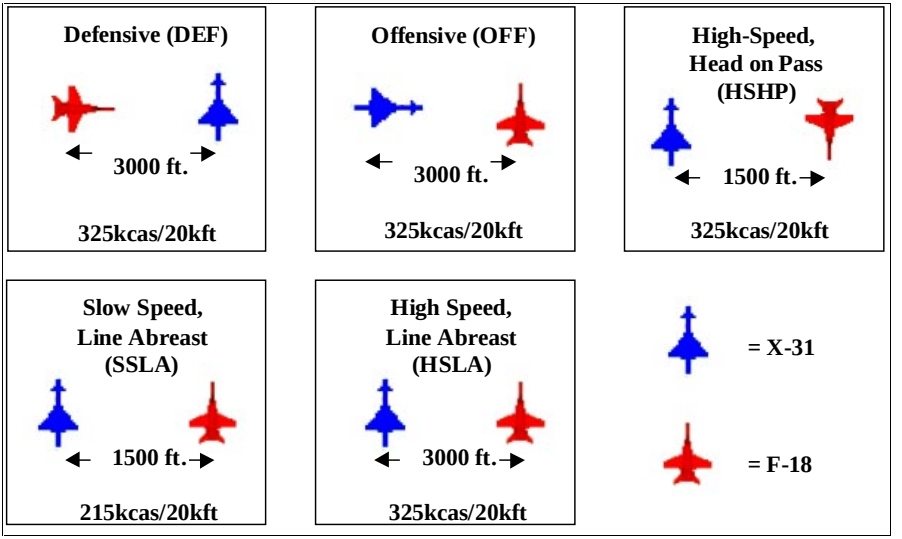


Fig. 2. Baseline test matrix of initial conditions for the combat simulations.

5 ”One-Sided Learning” Results

Results from the system outlined in Section 4 are extensively documented elsewhere [8,10]. However, results are reprinted here for clarification.

In our early efforts [8], only one of the fighter aircraft employs the genetic learning system, while the other employs fixed, but reactive, standard combat maneuvers that are embedded in AASPEM. Simply stated, these fixed maneuvers instruct the opponent to execute the fastest possible turn to point it’s own nose at the LCS-controlled aircraft, while attempting to match it’s altitude.

In this "one-sided learning" configuration, the system has found a variety of novel fighter aircraft maneuvers that were positively evaluated by actual fighter test pilots. One maneuever discovered by the LCS is shown in Fig. 3. Moreover, the system discovers maneuvers from a variety of well-recognized fighter aircraft strategies [7]. For instance, the result shown in Fig. 3 is a variant of the well-documented "Herbst maneuver"[7].

6 “Two-Sided Learning” Results

In more recent work with the fighter combat LCS, we have allowed both opponents to adapt under the action of the GA [10]. This ongoing effort complicates the fighter combat problem, and the interpretation of simulation results. These complexities are best seen from the perspective of extant literature on two player games.

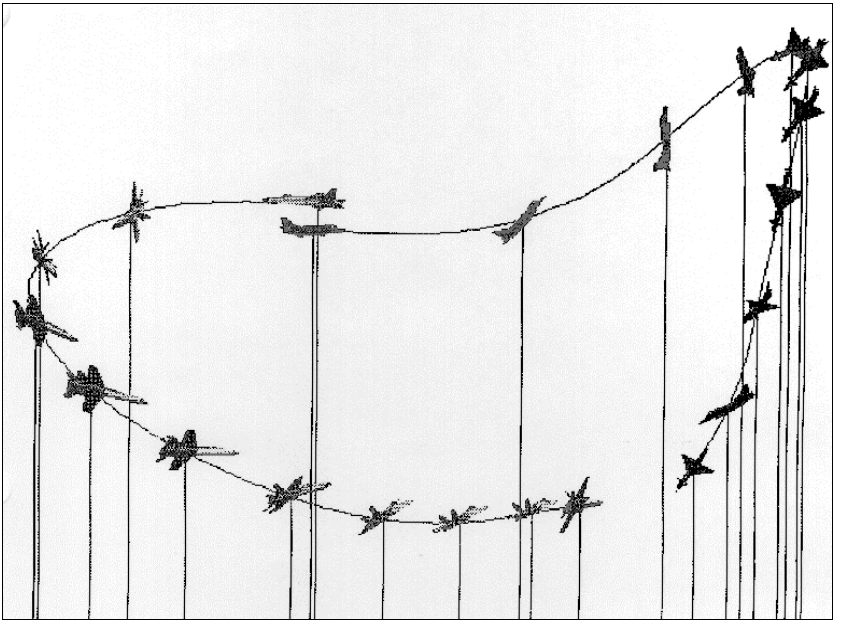


Fig. 3. A maneuver evolved by the LCS, starting from the HSHP starting position (see Fig. 2). The aircraft on the left is following a fixed, but reactive strategy, while the aircraft on the right is following a strategy evolved by the LCS [8].

In many ways, the one-versus-one fighter combat scenario is like a continuous version of the *iterated prisoners' dilemma* (IPD) [17,18]. In the IPD, the game takes place in distinct rounds, in which each player can take one of two actions: cooperate or defect. If both players cooperate, both players get a moderate reward. If one player defects while the other cooperates, the former gets a large reward, and the latter gets a large punishment (negative reward). If both defect, both get an even larger punishment.

This is similar to the situation in the one-versus-one fighter combat scenario. Each aircraft has a choice of attacking the other (analogous to defecting), or evading the other (analogous to cooperating). This analogy is useful, since IPD has been studied a great deal, both with static and adaptive players. However, it is important to note that the current work may be one of the first explorations of a real-world, continuous time analog of IPD with two adaptive players.

The IPD literature for co-adaptive players shows us several interesting behaviors. Each of these is an artifact of the *red queen effect*, so-called, because the red queen in Alice in Wonderland states that in her world you must keep running just to stand still [19]. In an analogous way, the performance of each player in the two-sided learning problem is relative to that of its opponent. In other words, when one player adapts and the other uses a static strategy (as in our previous work), the performance of the adaptive player is absolute with respect to its opponent. However, when both players are adaptive, the performance ceases to have an absolute meaning. Instead, its meaning is only relative to the state of its current opponent. This is an important effect that must be considered in the interpretation of our current results.

Also because of the red queen effect, the dynamic system created by two players has several possible attractors. These include

Fixed Points Where each player adapts a static strategy, as is willing to cope with a (possibly inferior) steady state.

Periodic Behavior Where each player cycles through a range of strategies, sometimes losing and sometimes winning.

Chaotic Behavior: Where each player visits a random series of strategies, with no eventual escalation of tactics.

Arms Races Where each player continuously ramps up the sophistication of its strategies.

The latter is clearly the behavior we want our simulations to encourage. Our current results have (qualitatively) shown promise in this area (i.e., we have seen an escalation of strategies between the two aircraft).

A number of approaches to two-sided learning have been considered. In each approach, a "run" consists of 300 simulated combat engagements. Approaches employed include:

1. **Alternate freeze learning (ALT):** In each run one side is learning while the other is frozen (not altering its rule base). The frozen player uses the population (rules) obtained from the previous run, in which it was learning. The learning player starts each new run with a random set of rules. In other words, rules are learned “from scratch” against the rules learned in the last cycle by the other player.
2. **Alternate freeze learning with memory (MEM):** This learning scheme can be viewed as an extended version of the ALT learning. At the end of each run, the results of the 300 engagements are scanned to obtain the highest measure of effectiveness. The rules from the highest scoring engagement are used for the frozen strategy in the next run. Furthermore, these rules are memorized and are added to the population in the upcoming learning sequence runs. Thus, the system has memory of its previously learned behavior.
3. **Parallel learning (PAR):** Genetic learning goes on continuously for both players.

Each of the three two-sided algorithms were tested and evaluated on all five Pinball cases (see Fig. 2). In all experiments a population of maneuvers was generated in an iterative process, beginning with an initial population of random classifiers. The maneuver population was successively improved using AASPEM air combat engagements to evaluate the maneuver.

A typical two-sided learning result (and one which indicates a learning "arms race") is shown in Fig. 4 and Fig. 5.

Fig. 4 shows the "best" maneuver (in terms of the measure of effectiveness discussed in Section 4.7) discovered for the Red Player. In this maneuver, the Red Player (which is an F-18) fails to have an advantage for more than the last half of the engagement due to agile maneuvers performed by the Blue Player (which is an X-31, a somewhat superior aircraft). The X-31 executes a nose dive maneuver followed by Herbst-type maneuver allowing it to gain an advantage. Although Red does not gain advantage, it is interesting to note its evasive behavior in reaction to blue’s maneuver.

Fig. 5 is the Blue Player's "best" maneuver (once again, in terms of the measure of effectiveness), which is evolved after the result shown in Fig. 4. It is likely that

learning from the Red Player's evasion (Fig. 4) resulted in the improved behavior seen in Fig. 5. This illustrates the advantage of two-sided learning. However, note that the meaning of "best", in terms of a static measure of effectiveness, is not entirely clear in these simulations, due to the red queen effect. Therefore, further investigation of the two-sided learning system is needed. However, the two-sided learning system is already yielding valuable maneuver information, in relationship to the system's overall goals.

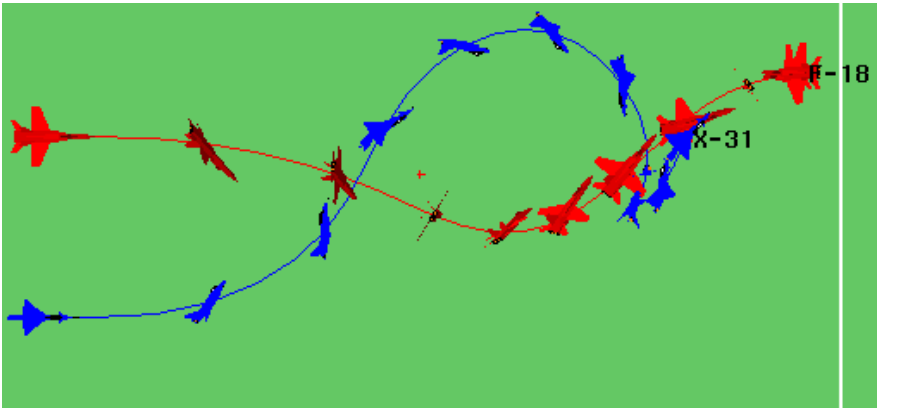


Fig. 4. A coevolved result, where both aircraft are executing maneuvers that were discovered by the separate LCSs, starting from the SSLA initial condition (see Fig. 2), and the ALT learning strategy. This is the "best" maneuver discovered by the Red Player, which is a reaction to the maneuver learned by the Blue Player [10].

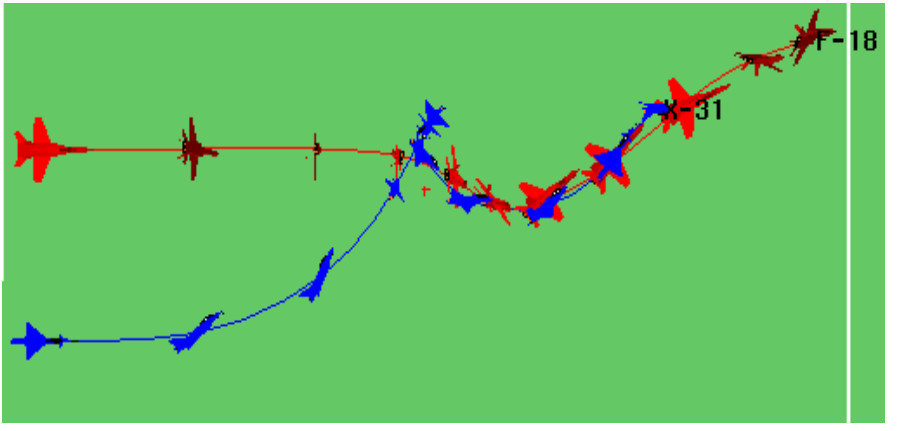


Fig. 5. A coevolved result, where both aircraft are executing maneuvers that were discovered by the separate LCSs, starting from the SSLA initial condition (see Fig. 2), and the ALT learning strategy. This is the "best" maneuver discovered by the Blue Player, which is a reaction to the "best" maneuver for the Red Player, which was discovered earlier by the Red Player's LCS (which is shown in Fig. 4) [10].

7 This System's Goal: Machine Innovation

Section 3 shows that many of the goals and techniques of the LCS have been clarified and formalized through active research in reinforcement learning. However, Sections 4, 5, and 6 show a successful LCS that has much in common with the traditional LCS. At first glance, this LCS seems to have less of a firm, reinforcement learning based foundation than many modern LCSs. Therefore, one must ask, why the LCS from Section 4, which is applied to a very complex problem, has not suffered from the typical problems of traditional LCSs, discussed in Section 2.

We believe much of the explanation lies in the goal at which the system is directed. From the outset of the fighter aircraft LCS project, the goal has not been to directly *control* fighter aircraft. Instead, the focus has been on the discovery of novel maneuvers for their own sake. The utility of this goal is revealed when one considers the complexity of the aircraft combat task [7].

In a general sense, one can understand the aircraft's *aerodynamic* characteristics before an actual prototype is generated. However, given the complexity of the aircraft combat task, one cannot map these aerodynamic characteristics directly to advantageous maneuvers. Therefore, test pilots, and extensive combat trials (mock or real), are generally needed to discover the innovative maneuvers that will convert raw aerodynamics into combat success.

Therefore, the goal of or system is simply the discovery of novel maneuvers, not optimality. There is, in general, no such thing as an optimal maneuver, and discovery of such a maneuver is not the system's goal. Discovering many *successful* combat maneuvers in this complex task has quantifiable advantages, including:

- Feedback to aircraft designers with regard to the combat advantages of various aerodynamic characteristics.
- Feedback to fighter pilots on the use of various aircraft characteristics.
- Feedback to customers on the advantages of a given aircraft's characteristics.

These advantages have real-world value. Moreover, they are advantages that could transfer to any number of other tasks where the discovery of novel strategies is difficult, and where new strategies have intrinsic values.

8 Implications of this Goal

In general, the goal of novelty discovery shifts the perception of what techniques are advantageous. For instance, the pursuit of *Q*-values that yield a Bellman optimal control strategy seems less critical in this setting. While it is true that one wants to pursue high utility maneuvers, it is not expected, that an "optimal" maneuver can be found. Moreover, enumerating all possible high-utility maneuvers is probably not possible. Finding classifier-based generalizations over the *entire* payoff landscape is probably not possible as well.

Most importantly, maintaining any *consistent* control strategy is not of great importance in this task. The frequent tendency of LCSs to have periodic failures noted in Section 2 is not a particular concern. Consider a typical result from the fighter aircraft system, shown in Fig. 6.

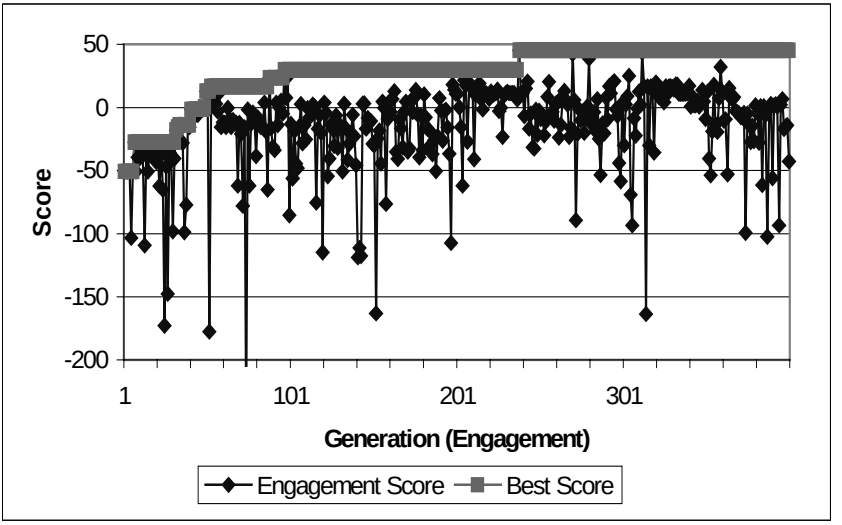


Fig. 6. Engagement Score for the LCS aircraft versus Generation Number. In this typical run, only one aircraft is using the LCS, while the other uses standard combat maneuver logic, as in the run shown in Fig. 3.

This figure shows that the fighter aircraft LCS frequently loses rules and suffers temporary losses of performance. However, the system moves to improved performance (and new maneuvers). Thus, it accomplishes the goal of *discovery*, but not necessarily the goal of control.

9 Techniques Used in this System

Like any LCS, the fighter aircraft LCS involves a complex interplay of elements. However, the authors believe there are two differences in technique that can be directly related to the particular goals of this system. The first involves conflict resolution and credit assignment, and the second involves effectors and action encoding.

9.1 Epochal Credit Allocation

Note that the system employs an *epochal* credit assignment scheme. This scheme is similar to the *profit sharing plan* [3] although it is applied in a "Michigan-style" LCS [6]. The scheme also employs deterministic conflict resolution, selecting the highest fitness matching classifier at every time step. At first, this may seem very disconnected from the reinforcement literature. However, it can be related to the Monte-Carlo reinforcement learning techniques discussed by Sutton and Barto [11].

Consider the following notation. Assume the strengths in the aircraft system are analogous to *Q*-values, environmental messages are analogous to states *s*, and action

messages are analogous to actions *a*. Given this notation, consider the following Monte Carlo reinforcement learning method, paraphrased from Sutton and Barto [11]:

1. For all states *s* and all actions *a*
   a. Initialize *Q(s,a)* to arbitrary values
   b. Initialize *Returns(s,a)* to an empty list
2. Repeat:
   a. Generate a complete engagement, *deterministically selecting* the maximum *Q(s,a)* action for each state *s* visited.
   b. For each state/action pair *s,a* that occurred in the engagement
      i. Append the engagement score to the list *Returns(s,a)*
      ii. Set *Q(s,a)* to average(*Returns(s,a)*)

Note that deterministic action selection is required in this algorithm. As Sutton and Barto indicate, this method is expected to converge to an optimal policy, but *only if* "exploring starts" for each engagement are employed. That is, each engagement must start at a randomly selected state, to ensure that the complete state-action space is explored.

Clearly, the fighter combat LCS does not employ exploring starts to ensure adequate exploration. However, we surmise that the randomness interjected by the GA, and the complexity of the fighter aircraft world, are sufficient to provide for exploration. Also note that returns are not being averaged in the fighter aircraft LCS, although such averaging is necessary in step 2.b.ii above. However, the GA itself carries out implicit schemata averages over generational time.

Therefore, although the credit allocation scheme used in the fighter combat LCS is not strictly similar to Bellman optimal reinforcement learning techniques, it does bear a relationship to these techniques. Note that this is as much as can be said for the *Q*-learning based techniques used in other LCSs. The authors are not arguing that the scheme used in the fighter aircraft LCS is superior to other techniques, simply that the scheme we have employed is not without foundation for an epochal reinforcement learning task.

Although the fighter combat LCS scheme is unlikely to have provable convergence to Bellman optimal values, recall that such convergence is not necessary for the goals of this problem. What is necessary is good evaluation of classifier schema, so ongoing exploration of the maneuver space can go on. Our empirical results indicate that this is the case in the fighter aircraft LCS.

9.2 Action Encoding

A second important aspect of the fighter combat LCS lies in the environmental effector's interpretation of classifier actions. In effect, the classifiers only *suggest* the actions to the environment, since many actions they may suggest may be aerodynamically impossible. However, because of the nature of the simulation environment (The Air-to-Air System Performance Evaluation Model, AASPEM, one of the standard models of fighter aircraft combat), the simulated aircraft responds with a "nearest feasible response" (in an aerodynamic sense) to the suggested action. This is an artifact of the AASPEM simulation environment that we have exploited in our system.

This means that a single, generalized classifier can have a variety of effects in different contexts. For instance, consider a generalized classifier (i.e., one that contains #s in its condition) that suggests the action: Relative Bank Angle = 45 degrees, Angle of Attack = 20 degrees, and Speed = 350 knots. This rule can have a variety of effects, depending on the (generalized) condition in which it is activated. If the aircraft was at a speed much lower than 350 knots, the rule is likely to result in a fast, but incremental, change *towards* 350 knots. If the aircraft was at a speed nearer to 350 knots, the rule may result in exactly the suggested value being achieved. The same holds for the other suggested action values. Moreover, the aerodynamic coupling between what values can be achieved is automatically handled by the simulation.

The authors believe this has several effects:

- It builds on the linguistic character of the classifier conditions, by giving the actions appropriate linguistic character (i.e., suggesting things like "speed up and bank steeply").
- It "smoothes" the effect of a classifier becoming less or more general due to the action of the GA, since the meaning of an action in a generalized context is likely to be similar to that in a subsumed, more specific context.
- It "smoothes" the effect of rule deletion, since one rule is likely to have a persistent effect, even if the context of the rule set in which it exists changes.

We believe this helps the system avoid the "rise and crash" behavior often observed in traditional LCSs. Moreover, it helps to continue discovery of new maneuvers after rule deletions, which is consistent with the goals discussed in the previous section.

10 Observations and Suggestions for Future LCS Efforts

There are at least three features of the current fighter aircraft LCS that deserve consideration in the design of other LCSs.

One should consider the fighter aircraft LCS credit allocation scheme. Given the efficacy and popularity of *Q*-learning and related algorithms (e.g., SARSA, which is essentially the implicit bucket brigade), it is easy to overlook that there are well-founded techniques that do not follow their form. In particular, the Monte-Carlo RL methods are just as firmly related to reinforcement learning theory. In fact, Sutton and Barto [11] indicate that such methods may be more appropriate than *Q*-learning and related methods, when one faces a non-Markovian task.

However, such methods are only appropriate for tasks with clearly defined episodes. The fighter aircraft task is clearly episodic, since an engagement takes place in a pre-defined time window. When one directs an LCS at an episodic task, epochal credit assignment schemes, like that used in the fighter aircraft LCS, may be of higher utility than methods based on *Q*-learning, SARSA, or the bucket brigade.

Like the *Q*-learning based methods used in LCSs, epochal schemes can only approximate the associated reinforcement learning techniques, since such techniques are typically based on table lookup. Given that the focus of the LCS is on finding generalized rules, associated reinforcement learning techniques, be they epochal or not, must be adapted to evolving, generalized rule sets. XCS [14] seems to have found

an appropriate adaptation of *Q*-learning for an LCS. Finding an appropriate adaptation of well founded, epochal reinforcement learning schemes to LCS use is an area worthy of future research.

One should also consider the "smoothed" interpretation of actions in the fighter combat LCS. The authors strongly suspect that this is a reason for the LCSs success on this task. Many of the action representations used in LCS research are "brittle". That is, rule deletion, and other forms of classifier set disruption, can easily cause less-than-graceful failure. In the fighter aircraft task, the actions of rules are implicitly interpreted in a linguistic, aerodynamically appropriate manner. This makes each classifier less sensitive to changes in the composition of the classifier set, which are likely to occur, given the action of the GA.

Finally, one should consider the goal of the LCS approach in the fighter aircraft LCS. Since there is a real, quantifiable value to the discovery of innovative, high utility fighter combat maneuvers, one can concentrate on the exploration and synthesis aspects of the LCS, without particular consider for the long term stability of any given rule set. While it is true that some stability is necessary for appropriate rule evaluation, and subsequent improvement of rule sets, occasional rule loss can be tolerated in this task.

Reinforcement learning control (that is, obtaining optimal strategies in reinforcement learning problems) is certainly a well-quantified goal, for which the LCS approach is useful. However, one should not overlook the utility of the LCS approach for generating novel, innovated approaches to problems. In many domains (like the fighter aircraft task), such open-ended discovery can have a real world, hard cash value. The applicability of the LCS approach to such tasks deserves further consideration.

Acknowledgments. The authors gratefully acknowledge that this work is sponsored by The United States Air Force (Air Force F33657-97-C-2035 and Air Force F33657-98-C-2045). The authors also gratefully acknowledge the support provided by NASA for the early phases of this project, under grant NAS2-13994.

References

1. Booker, L. B. (1992) Viewing Classifier Systems as an Integrated Architecture. Paper presented at The First International Conference on Learning Classifier Systems, Houston, Texas, October 1.
2. Goldberg, D. E. (1989) Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley.
3. Grefenstette, J. J. (1988) Credit assignment in rule discovery systems based on genetic algorithms. Machine Learning 3. pp. 225-246.
4. Holland, J. H. (1992) Adaptation in Natural and Artificial Systems MIT Press.
5. Holland, J. H., Holyoak, K. J., Nisbett, R. E. and Thagard, P. R. (1986) Induction: Processes of inference, learning, and discovery. MIT Press, Cambridge, MA.
6. Holland, J. H. and Reitman, J. S. (1978) Cognitive systems based on adaptive algorithms. In Waterman, D. A. and Hayes-Roth, F., " Pattern directed inference systems". Academic Press, NY.
7. Shaw, R. L. (1998) Fighter Combat : Tactics and Maneuvering. United States Naval Institute Press.

8. Smith, R. E. and Dike B. A. (1995) Learning novel fighter combat maneuver rules via genetic algorithms. International Journal of Expert Systems, 8(3) (1995) 247-276.
9. Smith, R. E., Dike, B. A. and Stegmann, S. A. (1994) Inheritance in Genetic Algorithms, in: Proceedings of the ACM 1995 Symposium on Applied Computing. ACM Press. pp. 345-350.
10. Smith , R. E., Dike, B. A., Mehra, R. K., Ravichandran, B. and El-Fallah, A. (in press). Classifier Systems In Combat: Two-Sided Learning of Maneuvers For Advanced Fighter Aircraft. Computer Methods in Applied Mechanics and Engineering, Elsevier.
11. Sutton, R. S. and Barto, A. G. (1998) Reinforcement Learning: An Introduction. MIT Press.
12. Watkins, J. C. H. (1989). Learning with delayed rewards. Unpublished doctoral dissertation. King's College, London.
13. Wilson, S. W. (1994) ZCS: A zeroth-level classifier system, Evolutionary Computation 2(1). pp. 1-18.
14. Wilson, S. W. (1995). Classifier fitness based on accuracy. Evolutionary Computation, 3(1), 149-176.
15. Wilson, S. W. (1999) State of XCS Classifier System Research. Technical Report Number 99.1.1 (Unpublished), Prediction Dynamics, Concord, MA.
16. P. M. Doane, C. H. Gay and J. A. Fligg, Multi-system integrated control (MuSIC) program. final report. Technical report, Wright Laboratories, Wright-Patterson AFB, OH., 1989.
17. R. Axelrod, The Evolution of Cooperation. (Basic Books, New York, 1984)
18. R. D. Luce and H. Raiffa, Games and Decisions. (Dover Publications, 1989).
19. D. Floriano and S. Nolfi, S., God save the red queen!: Competition in co-evolutionary robotics, in: Proceedings of the Second International Conference on Genetic Programming, (1997) 398-406.
20. Smith, R. E., Memory Exploitation in Learning Classifier Systems,(1995) Evolutionary Computation, 2 (3), pp. 199-220
21. Wilson, S. W. (1994) ZCS: A zeroth-level classifier system, Evolutionary Computation 2(1). pp. 1-18.

Latent Learning and Action Planning in Robots with Anticipatory Classifier Systems

Wolfgang Stolzmann and Martin Butz

Institute for Psychology, University of Wuerzburg, Germany
{stolzmann, butz}@psychologie.uni-wuerzburg.de

Abstract. Two applications of Anticipatory Classifier Systems (ACS) in robotics are discussed. The first one is a simulation of an experiment about latent learning in rats with a mobile robot. It shows that an ACS is able to learn latently, i.e. in the absence of environmental reward and that ACS can do action planning. The second one is about learning of the hand-eye coordination of a robot arm in conjunction with a camera. Goal-directed learning will be introduced. This combination of action planning and latent learning leads to a substantial reduction of the number of trials which are required to learn a complete model of a prototypical environment.

1 Introduction

In [10,this volume, pp. 189-208] ACS were applied to simple "woods" environments to show how they work and in which kind of environments they can be applied. In this chapter ACS will be used to control robots.

Firstly, an experiment of latent learning in rats is simulated with a mobile robot. In the field of learning classifier systems this experiment was first simulated by Riolo [6] with CFSC2. This simulation is not comparable with the robot-simulation described here, because here the internal representation of the environment is much more complicated for a robot-simulation (e.g. the robot has to deal with aliasing states).

Secondly, an experiment about learning the hand-eye coordination of a robot arm in conjunction with a camera is replicated with an ACS. This experiment was first done by Birk [1]. He used a learning algorithm called "Stimulus Response Learning".

For the simulation of the first experiment it is necessary to enable ACS to do *action planning*. In the second experiment the performance of an ACS can be improved, if *goal-directed learning* is used. Both algorithms *action planning* and *goal-directed learning* are introduced.

In this paper the same notations are used as in [10,this volume, pp. 175-194]

2 An Experiment about Latent Learning

Latent learning is defined as learning in the absence of environmental reward, i.e. $\rho(t) = 0$ for all time steps t. Therefore latent learning cannot be simulated with usual reinforcement learning techniques. In this section an experiment about latent learning

P.L. Lanzi, W. Stolzmann, and S.W. Wilson (Eds.): LCS '99, LNAI 1813, pp.301-317, 2000.

in rats is introduced, that was developed by Seward [7]. This experiment is simulated with a Khepera robot that learns by using an ACS. For this simulation the ACS must be able to learn latently and to do action planning. As shown in [10, this volume, pp. 175-194], an ACS is able to learn an internal model of its environment without getting environmental reward, i.e. it can learn latently. How action planning can be done in an ACS is shown in section 2.3.

2.1 An Experiment about Latent Learning in Rats

Seward [7] developed an experiment about latent learning in rats. He used a simple T-maze with two distinguishable goal boxes F and E shown in figure 1.

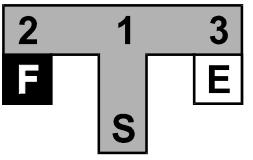


Fig. 1. A simple T-maze

During an exploration phase a rat was put into the T-maze and explored it without getting any reward. After that the rat was not fed for 24 hours. Then it was placed in one of the goal boxes and received food for one minute. Finally, the rat was placed in the box 'S'. 28 of 32 tested rats demonstrated latent learning by going to the good-goal box. A similar experiment was done by Croake [3]. During a control experiment without an exploration phase only about 50% of the rats go to the good-goal box. This experiment proves that the rats learned latently during the exploration phase, because there was no environmental reward.

2.2 A Khepera Robot

A Khepera robot is a mini mobile robot (5 cm diameter) distributed by K-Team S.A (*http://www.k-team.com*). Its features are 2 independent DC motors with encoders and 8 infra-red sensors (emitter/receiver) (cf. figure 2). The motor speeds range from -20 to +20, The infra-red reflection values range from 0 to 1023 and the infra-red ambient values range from 0 to 511.

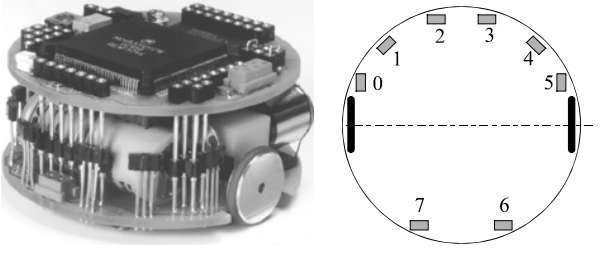


Fig. 2. A Khepera robot

This robot will be used to simulate Seward's experiment about latent learning in rats. The goal boxes of the T-maze must be distinguishable for the robot. This can be done with an infra-red light in the goal-box F. The infra-red light enables the robot to distinguish the light goal-box F from the dark goal-box E.

The Khepera robot will learn by using an ACS. So it is necessary to define an input interface and an output interface that enables the ACS to interact with the environment. It is possible to define the following 5 detectors and 3 simple actions for an ACS by using the 8 sensors and 2 motors of a Khepera-robot:

$d_1 = \begin{cases} 0, \text{if there is nothing in front of the robot} \\ 1, \text{if there is a wall in front of the robot} \end{cases}$,

d_2 as d_1, but on the left of the robot,

d_3 as d_1, but behind the robot,

d_4 as d_1, but on the right of the robot,

$d_5 = \begin{cases} 0, \text{if there is no light near to the robot} \\ 1, \text{if there is a ligth near to the robot} \end{cases}$,

l, to make a 90 degree left turn,

r, to make a 90 degree right turn,

f, to go forward until a wall or a choice-point is reached.

These detectors and simple actions are very robust. In all simulations a robot never got stuck in a corner of the T-maze.

Figure 3 shows the ACS's internal representation of the T-maze. This internal representation contains four important aliasing states: $2_N = 3_E$, $2_W = 3_N$, $2_S = 3_W$ and $2_E = 3_S$.

A simulation of Seward's experiment with a real Khepera robot would take a very long time. Therefore a real robot was only used for about 100 behavioral acts to show that the input interface and the output interface works. For the learning experiment we used a simulated robot.

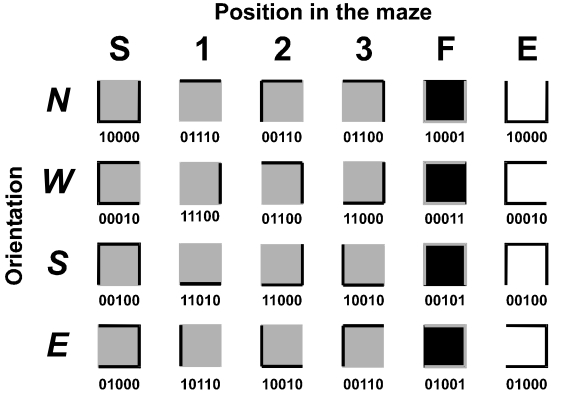


Fig. 3. The ACS's representation of the T-maze (*N=North, W=West, S=South, E=East*)

2.3 Action Planning in ACS

An ACS can use its internal model, i.e. its reliable classifiers for action planning. When doing action planning the expectation part E of a reliable classifier is used to calculate an anticipation of the next environmental state S_{t+1}^{ant}. It is assumed that S_{t+1}^{ant} is equal to the next state S_{t+1} and so it is possible to calculate all new satisfied reliable classifiers. For each new satisfied reliable classifier this can be done again. The same process can take place backwards, too. That is how a bi-directional search is used to generate an optimal sequence of classifiers (at least the system "believes" it is optimal) with action parts that lead from the actual state to a desired goal state. Action planning is done without any interactions with the environment.

Let us assume that we have a reliable classifier for each possible behavior in each state shown in figure 3, i.e. an internal model of the T-maze. (This assumption is wrong because some of the states are aliasing states, but nevertheless we make this assumption to explain action planning in ACS.) And let us assume that the current state is S_N and that the goal state is F_S. Now three algorithms are described that can be used to find a sequence of classifiers in the internal model that enables the ACS to reach the goal.

It's possible to use the *forwards-directed breadth-first search* to find such a sequence:

1. Let $c_i = C_i - A_i - E_i$ for $i = 1,\ldots,k$ be all reliable classifiers with: S_N matches C_i.
2. Calculate $S_1^{ant},\ldots,S_k^{ant}$ with $S_i^{ant} = passthrough\left(S_N, C_i^{ant}\right)$.
3. If $F_S \in \left\{S_1^{ant},\ldots,S_k^{ant}\right\}$, then stop the calculation, else repeat 1. and 2. for $S_1^{ant},\ldots,S_k^{ant}$ instead of S_N.
4. Repeat this recursively until F_S is found or a maximum number of repeats $s_{\max}$ is reached. $s_{\max}$ is called *maximum number of steps during action planning*.

If F_S is found, then execute all action-parts of the classifiers of the sequence. This execution of all action-parts can be regarded as goal-directed behavior. This is similar to lookahead planning for classifier systems that was introduced by Riolo (1991).

It is also possible to use the *backwards-directed breadth-first search* to find a path:

1. Let $c_i = C_i - A_i - E_i$ for $i = 1,\ldots,k$ be all reliable classifiers with: F_S matches E_i.
2. Calculate $S_1^{ant},\ldots,S_k^{ant}$ with $S_i^{ant} = passthrough\left(F_S, C_i\right)$.
   (Note: $passthrough\left(F_S, E_i\right)$ would be wrong here!)
3. If $S_N \in \left\{S_1^{ant},\ldots,S_k^{ant}\right\}$, then stop the calculation, else repeat 1. and 2. for $S_1^{ant},\ldots,S_k^{ant}$ instead of F_S.
4. Repeat this recursively until S_N is found or $s_{\max}$ is reached.

If S_N is found, then execute all action-parts of the classifiers of the sequence starting with the last classifier.

A third possibility is to combine both algorithms and use *bi-directional search*.

All of these three algorithms have the problem of cycles. For example, in S_N a left turn l leads to S_W and in S_W a right turn r leads back to S_N. Cycles can be avoided if only messages S_i^{ant} are considered for the *breadth-first search* that have not been calculated before.

Note: It is absolutely necessary to use a small value for $s_{\max}$ because of the exponential memory requirements of the search-algorithms.

Figure 4 shows an example for action planning:

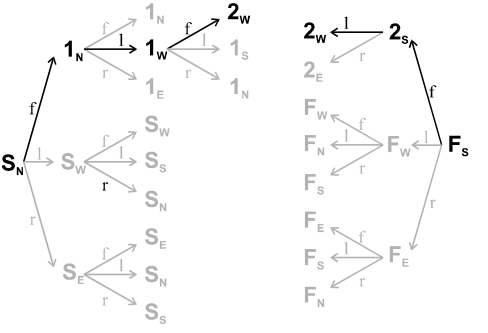


Fig. 4. An illustration of a bi-directional search while the ACS is doing" action planning

The robot is located in the position S_N. Its goal is to reach the left goal box F_S. If the *forwards-directed* search and the *backwards-directed* search reach a common state, here 2_W, then the action planning is successful and the ACS knows the actions that are necessary to reach the goal, here flflf.

2.4 Results

Two series of simulations were done, pure simulations and Webots-simulations. Webots is a open mobile robots simulator (*http://www.cyberbotics.com*). In contrast to a pure simulation the Webots-simulation has noisy detectors and effectors. Figure 5 shows two examples for behavioral and perceptual errors that happen during a Webots-simulation.

Fig. 5. Errors during a Webots-simulation

In the left example the robot has done several left and right turns without going forward in the mean time. So its orientation is south-east instead of east. If the robot

wants to go forward now it does not work. In the right example the robot is very near to the lower wall and does not detect the upper wall, i.e. the robot believes that there is no wall to its left-hand side.

One simulation consists of 100 runs. One run is organized as follows: The robot is controlled by an ACS that uses *specification of unchanging components*, *action chunking* to avoid aliasing states and *action planning*. The ACS starts without any knowledge and executes 100.000 behavioral acts. After each 10.000 behavioral acts the execution is interrupted and the bi-directional search checks whether the internal model (reliable classifiers) contains a path with a minimum number of behaviors from 1_N to F_S (or E_S). If the bi-directional search finds such a path, then we assume that the T-maze is learned.

The following parameters were used for the ACS: $\rho(t) = 0$ for all t, $b_q = 0.05$, $b_r = 0.0$, $p_x = 0.0$, $\theta_r = 0.95$, $\theta_i = 0.2$, $u_{\max} = 2$, $a_{\max} = 3$, $s_{\max} = 5$

$p_x = 0.0$ means that only roulette-wheel selection is used to select a classifier.

In Figures 6 and 7 the abscissa contains the number of behavioral acts and the ordinate the number of robots that learned the T-maze. Figure 6 shows the results of a series of 10 pure simulations and 1 Webots-simulation with food in F, i.e. a path from 1_N to F_S has to be learned. The result of the Webots-simulation is not different from the other results. This shows that an ACS is able to deal with low behavioral and perceptual errors. In [2] ACS in non-deterministic environments, e.g. in environments with noise in the actions, are discussed in further detail.

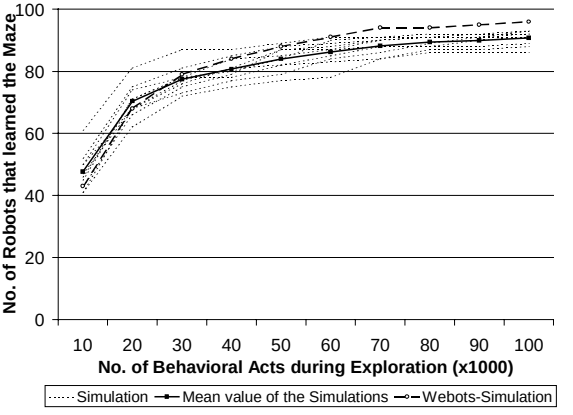


Fig. 6. Results of the simulations with food in F

In addition to the series of simulations with food in F we have done a series of 10 pure simulations with food in E, i.e. a path from 1_N to E_S has to be learned, to analyze whether there is a different learning rate in both series. Figure 7 shows the results.

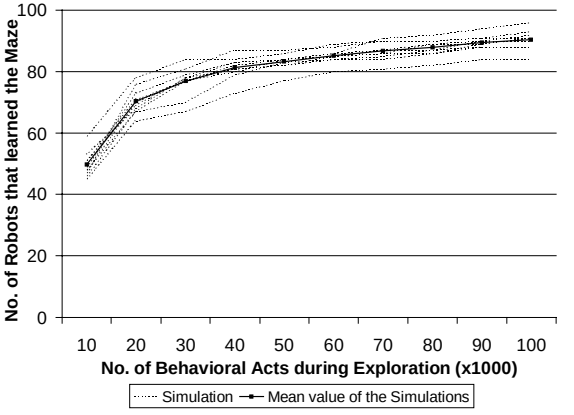


Fig. 7. Results of the simulations with food in E

A comparison of figures 6 and 7 proves that the results of the simulations do not depend on the goal during the post-eating trial because the mean values for 10 simulations with food in F and the mean values for 10 simulations with food in E are equal. A priori this is not clear because the light goal-box F is distinguishable from the position S but the dark goal-box E is not distinguishable from the position S. In other words the robot's perception of E is an aliasing state but the perception of F is not. This important difference between F and E does not have any consequences for the results of the simulation.

The performance can be improved, if also random selection is used to select a classifier. Figures 8 and 9 show the results for $p_x = 0.5$. All other parameters are the same as above.

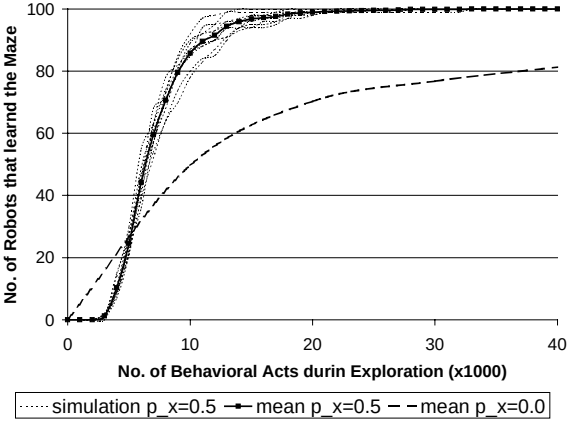


Fig. 8. Results of the simulations for $p_x = 0.5$ with food in F

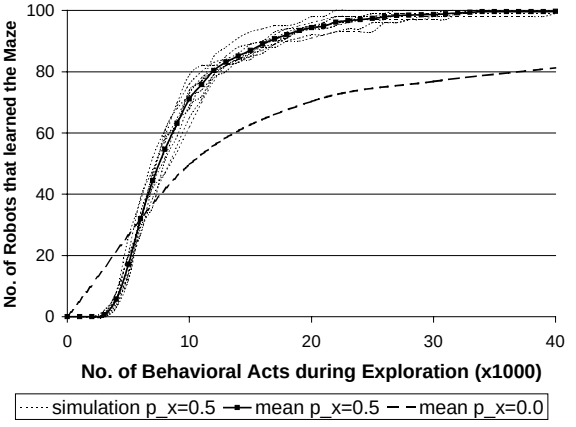


Fig. 9. Results of the simulations for $p_x = 0.5$ with food in E

A comparison of figures 8 and 9 shows that this time there is a difference between the simulations with food in F and the simulations with food in E. The performance in figure 8 is better than in figure 9.

An objection to this simulation of Seward's experiment could be that the task looks very simple and the ACS needs about 10.000 behavioral acts to reach 90% in figure 8. The task looks simple but it is not. The experiment was replicated with students: The states were presented as shown in figure 3 and the students were told to press three different keys to explore this finite state world for 15 minutes. After that only two of 11 students knew how to reach one of the goal-boxes.

3 An Experiment about the Hand-Eye Coordination of a Robot

This section focuses on the integration of action planning and latent learning in ACS, called goal-directed learning. It is merging the planning possibility with the latent learning process in ACS, building an action planning phase that will alternate with the normal exploration phase. The experiment discussed in section 2 is unsuited to discuss an integration of action planning and latent learning because first the environment is explored and then, after learning finished, action planning is used to reach a goal state. In this section an experiment about learning the hand-eye coordination of a robot in conjunction with a camera is discussed. This experiment was introduced by Birk [1]. A camera is monitoring a matrix representing a planar surface with a robot-arm on it. The robot is able to simply move in the four directions of the planar surface, grip a block, transport it to a different position and release it again. It has to learn the effects of its possible motor actions.

This scenario is especially useful to show the positive effects of goal-directed learning, because the different states do not occur with the same probability. Only on the position where the block is situated gripping will be successful, so that the system normally will move around the field without the block most of the time. Latent

learning of how to grip the block and to move it to a different position seems to be impossible. Besides, it will be possible to compare the positive effects of goal-directed learning in ACS to those Birk made in his work. Birk tested this experiment also on a real robot arm in combination with a real camera, which shows the possible application in reality.

First of all, the experiment is described. After that the method of goal-directed learning is introduced. Finally, the experimental results are presented and discussed.

3.1 The Experiment

3.1.1 The Coding

The camera (the eye) is vertically watching a black planar surface. The picture is diverted in a $n \times n$-matrix. Over this surface, the robot-arm with a red gripper is installed. It can move in the four directions North, South, East and West over the plane. Besides there is a blue block that can be gripped by the robot-arm and be moved around the planar surface. The robot-arm has got a feeling sensor that switches from 0 to 1 when it feels a block under it. That is why the possible environmental states can be coded in $\{\text{black}, \text{red}, \text{blue}\}^{n^2} \times \{0,1\}$. The possible motor actions are N = North, S = South, E = East, W = West, G = Grip and R = Release. As mentioned, Birk implemented this experiment also on a real robot. Therefore the coding was not changed. Figure 10 shows the robot and its environment for n = 5.

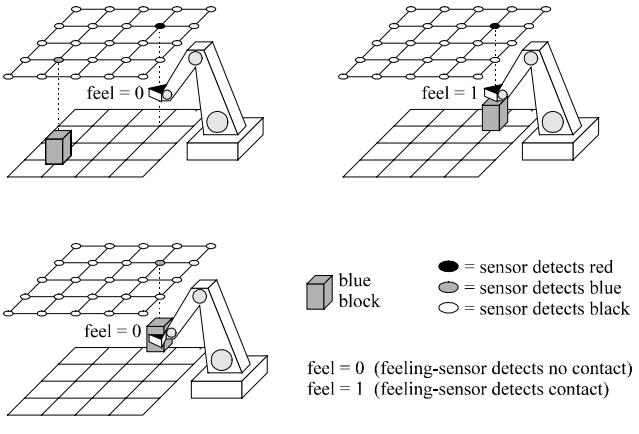


Fig. 10. : The Robot and its Environment (cf. Birk [1, p. 35])

3.1.2 The Size of the Knowledge

As in all Classifier Systems the knowledge is represented by its classifiers. The ACS has to generate in each position a classifier that moves in each direction with or without the block, onto the block or off the block. Furthermore, it needs a classifier that is able to grip and to release the block in each position (in case the block is under

it). Table 1 shows the necessary raw classifiers for a 2×2 matrix for each of this possibilities, if *specification of unchanging components* is not used..

Table 1. Sorted representation of the classifier list that consists of the knowledge generated without further specialization for *n*=2 (r: red; b: blue; s: black; N: north; S: south; W: west; E: east; G: grip; R: release)

move onto the block	move off the block	move without contact to the block	transport the block	grip the block	release the block
(b#r#0,N,r#s#1)	(s#r#1,N,r#b#0)	(s#r##,N,r#s##)	(s#b##,N,b#s##)	(r###1,G,b###0)	(b###0,R,r###1)
(#b#r0,N,#r#s1)	(#s#r1,N,#r#b0)	(#s#r#,N,#r#s#)	(#s#b#,N,#b#s#)	(#r##1,G,#b##0)	(#b##0,R,#r##1)
(r#b#0,S,s#r#1)	(r#s#1,S,b#r#0)	(r#s##,S,s#r##)	(b#s##,S,s#b##)	(##r#1,G,##b#0)	(##b#0,R,##r#1)
(#r#b0,S,#s#r1)	(#r#s1,S,#b#r0)	(#r#s#,S,#s#r#)	(#b#s#,S,#s#b#)	(###r1,G,###b0)	(###b0,R,###r1)
(br##0,W,rs##1)	(sr##1,W,rb##0)	(sr###,W,rs###)	(sb###,W,bs###)		
(##br0,W,##rs1)	(##sr1,W,##rb0)	(##sr#,W,##rs#)	(##sb#,W,##bs#)		
(rb##0,E,sr##1)	(rs##1,E,br##0)	(rs###,E,sr###)	(bs###,E,sb###)		
(##rb0,E,##sr1)	(##rs1,E,##br0)	(##rs#,E,##sr#)	(##bs#,E,##sb#)		

Therefore the size of the knowledge is n^2 each for gripping and releasing, $4(n-2)^2+3\cdot4(n-2)+2\cdot4=4n^2-4n$ for transporting the block, moving without contact to the block, moving off the block and moving onto the block. Because of the passthrough signs in ACS moving without contact to the block is independent from the actual position of the block and therefore has got the same size as moving off the block, moving onto the block or transporting the block. All in all that is $4(4n^2-4n)+2n^2=18n^2-16n$. Table 2 shows the size of the necessary knowledge for n=2 to 7.

Table 2. The number of different classifiers necessary for a complete knowledge

n=2	n=3	n=4	n=5	n=6	n=7
40	114	224	370	552	770

3.1.3 More Specific Knowledge

Considering the classifiers for transporting the block, one can realize that each of these classifiers can also match in a state where the arm and the block have no contact as each #-sign could either match with an 's' for black or with an 'r' for the robot arm. The same problem arises when looking at the classifiers for releasing the block. These classifiers have to be specialized by using *specification of unchanging components* to make sure that they are only applicable in the correct state.

Taking e.g. the first classifier for transportation in table 1: s#b## N b#s##, meaning that when transporting the block one step northwards the third component becomes black (s) and the first component becomes blue (b). It is applicable in the states ssbs0, srbs0 and ssbr0 but only successful in ssbs0, as in the other two states the robot is in a different position without the block. Once generated it will also be applied in the other states and will be marked there. Let us assume that the classifier has got a mark in state srbs0. Now it is applied again in the successful state ssbs0 and realizes that it has got the mark srbs0. That is why it will generate a new classifier that will explicitly carry an 's' (instead of the '#') in the second component to differentiate itself from the

other state: ssb## N bss##. After setting another mark in ssbr0 and again being applied successfully the classifier will be ssbs# N bsss# and will only match the correct state from now on.

The classifiers for releasing the block have to be specialized in the same manner, as they can also be applied in states where the gripper is in a different position than the block. Furthermore, the classifiers for moving without contact to the block have to be specialized, too, i.e. in the last (the feeler) component, to make sure not to be over the block. Let's consider e.g. the first classifier for moving without contact to the block in table 1: s#r## N r#s##. It is applicable in the states sbrs0, ssrb0, ssrs1, but only correct in sbrs0 and ssrb0. So eventually it will be marked by ssrs1. Let's say it is successfully applied in sbrs0 again. Now it is possible to specialize the second or the last component. While the generated classifier that is specified in the second component matches from now on only in sbrs0, the generated classifier that is specified in the last component matches in both correct states and therefore is more useful than the other. Currently this is an unsolved problem in ACS. Despite this, eventually the classifier list will contain the classifiers shown in table 3, which represent a complete and unique knowledge of the environment.

Table 3. Sorted representation of the classifier list that consists of the complete and unique knowledge (after specification of unchanging components) for n=2 (r: red; b: blue; s: black; N: north; S: south; W: west; E: east; G: grip; R: release)

move onto the block	move off the block	move without contact to the block	transport the block	grip the block	release the block
(b#r#0,N,r#s#1)	(s#r#1,N,r#b#0)	**(s#r#0,N,r#s#0)**	**(ssbs#,N,bsss#)**	(r###1,G,b###0)	**(bsss0,R,rsss1)**
(#b#r0,N,#r#s1)	(#s#r1,N,#r#b0)	**(#s#r0,N,#r#s0)**	**(sssb#,N,sbss#)**	(#r##1,G,#b##0)	**(sbss0,R,srss1)**
(r#b#0,S,s#r#1)	(r#s#1,S,b#r#0)	**(r#s#0,S,s#r#0)**	**(bsss#,S,ssbs#)**	(##r#1,G,##b#0)	**(ssbs0,R,ssrs1)**
(#r#b0,S,#s#r1)	(#r#s1,S,#b#r0)	**(#r#s0,S,#s#r0)**	**(sbss#,S,sssb#)**	(###r1,G,###b0)	**(sssb0,R,sssr1)**
(br##0,W,rs##1)	(sr##1,W,rb##0)	**(sr##0,W,rs##0)**	**(sbss#,W,bsss#)**		
(##br0,W,##rs1)	(##sr1,W,##rb0)	**(##sr0,W,##rs0)**	**(sssb#,W,ssbs#)**		
(rb##0,E,sr##1)	(rs##1,E,br##0)	**(rs##0,E,sr##0)**	**(bsss#,E,sbss#)**		
(##rb0,E,##sr1)	(##rs1,E,##br0)	**(##rs0,E,##sr0)**	**(ssbs#,E,sssb#)**		

3.1.4 Testing the Knowledge

For monitoring purpose of the evolved knowledge a testing method of the currently achieved knowledge is necessary. The testing method is taking all the reliable classifiers and checks if they are capable to solve all the tasks mentioned in 3.1.2. Although it should be irrelevant where the block is situated in the skill *moving without contact to the block* all block positions are tested for making sure, that all correct and unique classifiers are formed (see problem in 3.1.3). Meaning that once achieved a 100% test result all possible goals will be reached successfully and even in an optimal number of steps. Therefore the size of the tested knowledge is $2n^2$ for gripping and releasing, $n^2(4n^2-4n)$ for all possible movements without the block considering all possible block positions and $4n^2-4n$ for all possible movements with the block resulting in $2n^2+(4n^2-4n)\cdot(n^2+1)$.

Note the difference between the growth of the different action tests. While the complexity of the actions for the possible movements without the block is $\in O(n^4)$,

the complexity of the other, actual more difficult actions is $\in O(n^2)$. Table 4 shows the growth of the sizes.

Table 4. The size of all tested actions and in comparison the size of the possible actions with the block

n	No. of all tested actions	No. of actions with the block
2	48	16 (32%)
3	258	42 (16%)
4	848	80 (9.4%)
5	2130	130 (6.1%)
6	4512	192 (4.3%)
7	8498	266 (3.1%)

3.2 Goal-Directed Learning

In the introduced environment the probabilities of the occurrences of each state are not equal. Transporting the block is only possible if the system tried to grip the block while over the block. This will occur quite seldom as the latent learning process is not rewarded anywhere and gripping in any other position than the block position will result in weakening the quality q of the grip-classifier. Transporting needs a successful grip before it can be executed so that it is still more unlikely. As it will be shown goal-directed learning is a possibility to decrease this effect.

3.2.1 The Exploration Phase and the Action Planning Phase

Action planning is done without any interactions with the environment (cf. section 2.3). This paper will not focus on the question which goal state is desirable but the goals will simply be generated externally by a so called goal-generator, that is part of the environment and not part of the ACS itself.

In the action planning phase an ACS requests a goal from the goal-generator. It tries to find the shortest possible path from the actual state to the goal by using the bi-directional search mechanism mentioned in 2.3. If it finds such a path, then it executes the classifiers of the path like in the exploration phase. If the execution was successful, then the ACS requests a new goal from the goal-generator. If the execution of any found sequence failed or if the search mechanism did not find a sequence at all, then the ACS leaves the action planning phase and reenters the normal exploration phase. During the exploration phase the ACS explores its environment, as described in Stolzmann [10,this volume, pp. 175-194]. Every 50 steps the action planning phase is taking place.

3.2.2 The Application of Goal-Directed Learning in the Experiment

The goal-generator generates continuously the following goals:

1. Move to a randomly selected place, where the block is not situated.
2. Move to the block

3. Grip the block.
4. Release the block at a different randomly selected position.

The goal-generator and the switching between the action planning phase and the exploration phase is analogous to Birk (cf. [1, p. 61ff]).

Let's consider an example in which the ACS has already learned the complete world model (see table 3). Let's e.g. assume the following constellation of a 2x2 matrix: sbsr0, meaning the block is situated Northeast and the robot hand Southeast. Now the action planning phase starts. The goal generator in phase one creates e.g. the goal sbrs0. The search mechanism will find a sequence that consists of only one classifier: ##sr0 W ##rs0. After the execution of this classifier the environmental state will be sbrs0 and the ACS requests a new goal. The goal generator, now in phase two, will generate the goal srss1. The search mechanism generates the sequence: ##sr0 E ##rs0, #b#r0 N #r#s1 or s#r#0 N r#s#0, rb##0 E sr##1 and executes it. In phase three the goal is sbss0 and the sequence #r##1 G #b##0 is generated. After the execution the environmental state is sbss0 and the goal generator enters phase four and generates e.g. rsss1, that leads the ACS to the sequence sbss# W bsss#, bsss0 R rsss1. After the execution of these two classifiers the goal generator again enters phase one and the process starts from the beginning.

If the world model is not learned completely, then on the one hand a sequence might be longer than the optimal path and on the other hand the action planning process might fail. In this case the ACS reenters the exploration phase.

3.2.3 Advantage of Goal-Directed Learning

Due to the permanent request of the goal-generator during phase 2, to move to the block and to grip it during phase 3, the system will be much more often in a position over the block and so experiences the results of gripping much more often. That is how the unequal distribution of probabilities between the different states shown in table 4 is partly neutralized. For there is a quadratic growth of the number of fields but the number of blocks stays the same (only one block) this advantage should increase with the size of the matrix.

3.3 Results

This section shows on the one hand the positive effect of adding goal-directed learning to ACS and on the other hand the overall performance of ACS in this simulation.

3.3.1 Comparing ACS in Conjunction with and Without Goal-Directed Learning

Figure 11 shows the mean results of 100 simulations of the $n \times n$ matrix for n=2 and n=3 without and with goal-directed learning. The positive effect of goal-directed learning becomes obvious.

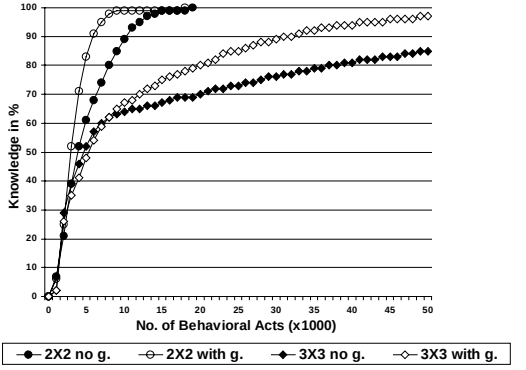


Fig. 11. Results of the simulation with n=2, 3 without and with goal-directed learning

Table 5 shows the same results in numeric form and the mean results of 40 simulations for n=4 without and with goal-directed learning.

Table 5. Average results for n=2-4 with and without goal-directed learning

	2x2 no g.	**2x2 with g.**	**3x3 no g.**	**3x3 with g.**	**4x4 no g.**	**4x4 with g.**
>75%	7200	4400	27200	15000	14500	16000
>80%	8000	4800	36400	19400	37500	24000
>85%	9000	5200	48200	24000	97500	49000
>90%	10400	6000	62400	30800	187500	75000
>95%	11800	6800	82000	40800	320500	119000
>99%	15000	8200	114800	61800	>600000	301000

Obviously the ACS with goal-directed learning needs about half the behavioral steps of the one without goal-directed learning. The effect is still increasing for increasing numbers of n. This confirms the reflections in section 3.2.3.

The result that 75% of the 4x4 matrix are learned faster than 75% of the 3x3 matrix is an effect of the measurement of the knowledge. As explained in 3.1.4 the complexity of the possible movements without the block is $O(n^4)$ while the one with the block is $O(n^2)$. During the exploration phase the movements without the block are taking place most of the time so that most of the corresponding classifiers are learned very quickly. For n=4 this is 90.6% of the world model and for n=3 only 84%. The same effect explains the slightly better performance of ACS without goal-directed learning than the performance of ACS with goal-directed learning for the 4x4-matrix in the 75% results because an ACS without goal-directed learning is always in the exploration phase.

Figures 12 and 13 show the results of 100 simulations each of a 3x3 matrix without and with goal-directed learning in detail revealing the differences between the single experiments and again the positive effect of goal-directed learning.

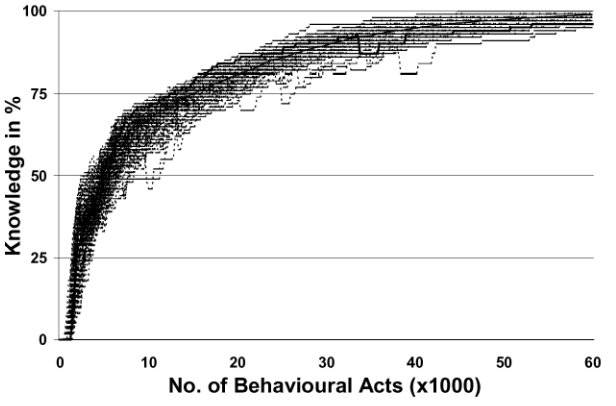


Fig. 12. 100 single experiments in a 3x3 matrix with goal-directed learning

Fig. 13. 100 single experiments in a 3x3 matrix without goal-directed learning

3.3.2 More Results with Action Planning

As shown in 3.3.1 the performance of ACS in conjunction with goal-directed learning is much better than without goal-directed learning. Therefore some more results for $n \times n$-matrixes with goal-directed learning are calculated. Table 6 shows the average performance for n=4 (for reference purposes), 5 and 6 with 10 experiments each and for n=7 one example experiment is given.

Table 6. The results for n= 4 to 7 in the learning constellation with goal-directed learning

	n=4	**n=5**	**n=6**	**n=7**
>75%	16000	16000	18000	18000
>80%	24000	23000	22000	24000
>85%	49000	42000	44000	30000
>90%	75000	99500	86000	54000
>92%	89000	141500	154000	80000
>94%	107500	194500	236000	158000
>96%	139000	276500	412000	428000
>98%	195500	479500	710000	1188000
>99%	261500	727500	1115000	1938000

3.4 Comparison with Birk's SRL

Birk [1] developed a learning algorithm called "Stimulus Response Learning" (SRL) to solve the presented experiment. In a situation S_2 the last behavior *R* and its starting situation S_1 are used to build a totally specialized rule $S_1 - R - S_2$. Such rules are generalized by using genetic algorithms. An ACS learns similar rules but it does not need genetic algorithms.

Analogous to Birk we were able to show the positive effect of goal-directed learning, although his experimental results seem to prove an even greater effect.

But first of all Birk only refers to the time his computer needed for the different tasks, so that a direct comparison is not possible. Moreover, his system is not creating simple rules but a graph whose edges are not trying to generate common results but indeed monitoring the effects completely. Last but not least the knowledge about the environment is not complete as in ACS. The junctions in SRL are too common. That is why a gripping action is possible to take place where no block is under the robot-arm.

In the current system the possibilities of planning are restricted (...) the system is normally not able to generate a successful plan for gripping a block [1, p. 71], own translation).

These restrictions are not given in ACS. Once the system has generated a complete world model it is not anymore limited and all possible tasks will be performed correctly.

4 Conclusions

This chapter showed that ACS can be used for learning robots. The performance of an ACS was examined when being applied to a mobile Khepera robot and a robot gripper that works in conjunction with a camera. Furthermore, ACS's ability of doing action planning was introduced and successfully applied.

Section 2 introduced an extended simulation of Seward's experiment about latent learning in rats. The experiment was firstly simulated by Riolo [6]. Stolzmann [8,9] discussed a first simulation with ACS. A comparison with Riolo's CFSC2 is given by

Stolzmann [9]. The here introduced extension shows that Seward's experiment can be replicated with a mobile robot (in this case a Khepera robot) that learns by using ACS. Furthermore, in contrast to the earlier simulations, the ACS that controlled the mobile robot had to handle aliasing states.

In Section 3 the ability of an ACS to generate plans was shown. The experiment of a hand-eye coordination task simulated with a robot gripper and a camera proved that an ACS is able to take advantage out of generating plans to given goals, even while learning. It is able to learn much faster in environments where the probability of entering each state is not equal.

Although the goal generating process was part of the environment in this simulation, integrating the goal generator into the ACS should be possible. A possibility would be to add a temporary memory to the ACS out of which it could select states, that were e.g. experienced quite long ago. This approach can even be psychologically confirmed. In contrary to the known "joy of satisfaction", which could be compared to the normal latent learning in an ACS, a so called "joy of function" is observable especially in young children, when they are enjoying doing something over and over again [4]. Future research will show if such a self-goal-generating process can be successfully applied in the ACS.

Finally, it was shown that the marking mechanism does not always reveal the definite position for doing specification of unchanging components. In [2] this problem is discussed in further detail. To be able to generate always most general classifiers in an ACS this issue deserves future research, as well.

References

1. Birk, Andreas (1995). *Stimulus Response Lernen*. Dissertation. University of Saarbruecken. Germany.
2. Butz, M., Goldberg, D., & Stolzmann, W. (1999). New Challenges for an Anticipatory Classifier System: Some Hard Problems and Possible Solutions. IlliGAL Report No. 99019. Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign
3. Croake, J. W. (1971). Unrewarded exploration and maze learning. *Psychological Reports*, 1971, 29, 1335-1340.
4. Hoffmann, Joachim (1993). *Vorhersage und Erkenntnis*. Göttingen: Hogrefe.
5. Michel, Olivier (1998). Webots - User Guide. Cyberbotics (*http://www.cyberbotics.com*)
6. Riolo, Rick L. (1991). Lookahead planning and latent learning in a classifier system. In Meyer, J.A., and Wilson, S. W. (editors.). *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*. Cambridge, MA: The MIT Press. 316-326.
7. Seward, John P. (1949). An Experimental Analysis of Latent Learning. *Journal of Experimental Psychology*. 39 177-186.
8. Stolzmann, Wolfgang (1997). *Antizipative Classifier Systeme*- PhD Dissertation. Fachbereich Mathematik /Informatik. University of Osnabrueck. Aachen: Shaker Verlag.
9. Stolzmann, Wolfgang (1998). Anticipatory Classifier Systems. In Koza, John R. et al. (editors). Genetic Programming 1998: *Proceedings of the Third Annual Conference, July 22-25, 1998, University of Wisconsin, Madison, Wisconsin.* San Francisco, CA: Morgan Kaufmann. 658-664.
10. Stolzmann, Wolfgang (2000). An Introduction to Anticipatory Classifier Systems. In Lanzi, Stolzmann, Wilson *Learning Classifier Systems: An Introduction to Contemporary Research.* Pagg. 175-194. LNAI 1813. Springer, Berlin(This Volume)

A Learning Classifier Systems Bibliography

Tim Kovacs[1] and Pier Luca Lanzi[2]

[1] School of Computer Science
University of Birmingham
Birmingham B15 2TT, United Kingdom
T.Kovacs@cs.bham.ac.uk
[2] Artificial Intelligence & Robotics Project
Dipartimento di Elettronica e Informazione
Politecnico di Milano
lanzi@elet.polimi.it

Abstract. We present a bibliography of all works we could find on Learning Classifier Systems (LCS) – the genetics-based machine learning systems introduced by John Holland. With over 400 entries, this is at present the largest bibliography on classifier systems in existence. We include a list of LCS resources on the world wide web.

Introduction

Learning classifier systems (LCS) have a long and rich history. We hope this bibliography will both illustrate this point and prove a useful resource for researchers. Although the first classifier system, CS-1, was reported in 1978 [235], the development of LCS was foreshadowed by some of Holland's earlier work [220,221,222] dating back as far as 1971. In the early 80's much progress was in the form of PhD theses [388,37,192,352,170] (but also see [427,428]), following which LCS papers began to appear steadily in conferences. Later landmarks include the publication of books by Holland in 1986 [233] and Goldberg in 1989 [195], and the first and second international workshops on LCS, in 1992 and 1999 respectively.

As the bibliography shows, LCS papers appear in a wide range of conferences and journals. LCS resources on the world wide web include:

The LCS Web http://www.csm.uwe.ac.uk/~ambarry/LCSWEB/
GA-list A mailing list about Genetic Algorithms. To subscribe mail: ga-list-request@aic.nrl.navy.mil with 'subscribe' in the body of the mail.
GA-list archive http://www.aic.nrl.navy.mil/galist
comp.ai.genetic A USENET news group, which provides a useful list of Frequently Asked Questions.
ENCORE (Evolutionary Computation Repository Network) http://www.cs.bham.ac.uk/Mirrors/ftp.de.uu.net/EC/clife/
The LCS Bibliography http://www.cs.bham.ac.uk/~tyk/lcs/ Contributions and corrections can be mailed to the first author at: T.Kovacs@cs.bham.ac.uk

P.L. Lanzi, W. Stolzmann, and S.W. Wilson (Eds.): LCS '99, LNAI 1813, pp. 321–347, 2000.

References

1. Emergent Computation. Proceedings of the Ninth Annual International Conference of the Center for Nonlinear Studies on Self-organizing, Collective, and Cooperative Phenomena in Natural and Artificial Computing Networks. A special issue of Physica D. Stephanie Forrest (Ed.), 1989.
2. *Collected Abstracts for the First International Workshop on Learning Classifier System (IWLCS92)*, 1992. October 6–8, NASA Johnson Space Center, Houston, Texas.
3. Jose L. Aguilar and Mariela Cerrada. Reliability-Centered Maintenance Methodology-Based Fuzzy Classifier System Design for Fault Tolerance. In Koza et al. [262], page 621. One page paper.
4. Manu Ahluwalia and Larry Bull. A Genetic Programming-based Classifier System. In Banzhaf et al. [12], pages 11–18.
5. Rudolf F. Albrecht, Nigel C. Steele, and Colin R. Reeves, editors. *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms*. Spring-Verlag, 1993.
6. Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors. *Proceedings of the 1999 Congress on Evolutionary Computation CEC99*, Washington (DC), 1999. IEEE Press.
7. W. Brian Arthur, John H. Holland, Blake LeBaron, Richard Palmer, and Paul Talyer. Asset Pricing Under Endogenous Expectations in an Artificial Stock Market. Technical report, Santa Fe Institute, 1996. This is the original version of LeBaron1999a.
8. Thomas Bäck, editor. *Proceedings of the 7th International Conference on Genetic Algorithms (ICGA97)*. Morgan Kaufmann: San Francisco CA, 1997.
9. Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors. *Handbook of Evolutionary Computation*. Institute of Physics Publishing and Oxford University Press, 1997. http://www.iop.org/Books/Catalogue/.
10. Thomas Bäck, Ulrich Hammel, and Hans-Paul Schwefel. Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, 1(1):3–17, 1997.
11. N. R. Ball. Towards the Development of Cognitive Maps in Classifier Systems. In Albrecht et al. [5], pages 712–718.
12. Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*. Morgan Kaufmann: San Francisco, CA, 1999.
13. Alwyn Barry. The Emergence of High Level Structure in Classifier Systems - A Proposal. *Irish Jounral of Psychology*, 14(3):480–498, 1993.
14. Alwyn Barry. Hierarchy Formulation Within Classifiers System – A Review. In Goodman et al. [198], pages 195–211.
15. Alwyn Barry. Aliasing in XCS and the Consecutive State Problem: 1 – Effects. In Banzhaf et al. [12], pages 19–26.
16. Alwyn Barry. Aliasing in XCS and the Consecutive State Problem: 2 – Solutions. In Banzhaf et al. [12], pages 27–34.
17. Richard J. Bauer. *Genetic Algorithms and Investment Strategies*. Wiley Finance Editions. John Wiley & Sons, 1994.
18. Richard K. Belew and Stephanie Forrest. Learning and Programming in Classifier Systems. *Machine Learning*, 3:193–223, 1988.

19. Richard K. Belew and Michael Gherrity. Back Propagation for the Classifier System. In Schaffer [354], pages 275–281.
20. Hugues Bersini and Francisco J. Varela. Hints for Adaptive Problem Solving Gleaned From Immune Networks. In Schwefel and Männer [358], pages 343–354.
21. Janine Beunings, Ludwig Bölkow, Bernd Heydemann, Biruta Kresling, Claus-Peter Lieckfeld, Claus Mattheck, Werner Nachtigall, Josef Reichholf, Bertram J. Schmidt, Veronika Straaß, and Reinhard Witt. *Bionik: Natur als Vorbild*. WWF Dokumentationen. PRO FUTURA Verlag, München, 1993.
22. J. Biondi. Robustness and evolution in an adaptive system application on classification task. In Albrecht et al. [5], pages 463–470.
23. Andrea Bonarini. ELF: Learning Incomplete Fuzzy Rule Sets for an Autonomous Robot. In Hans-Jürgen Zimmermann, editor, *First European Congress on Fuzzy and Intelligent Technologies – EUFIT'93*, volume 1, pages 69–75, Aachen, D, September 1993. Verlag der Augustinus Buchhandlung.
24. Andrea Bonarini. Evolutionary Learning of General Fuzzy Rules with Biased Evaluation Functions: Competition and Cooperation. *Proc. 1st IEEE Conf. on Evolutionary Computation*, pages 51–56, 1994.
25. Andrea Bonarini. Learning Behaviors Represented as Fuzzy Logic Controllers. In Hans Jürgen Zimmermann, editor, *Second European Congress on Intelligent Techniques and Soft Computing - EUFIT'94*, volume 2, pages 710–715, Aachen, D, 1994. Verlag der Augustinus Buchhandlung.
26. Andrea Bonarini. Extending Q-learning to Fuzzy Classifier Systems. In Marco Gori and Giovanni Soda, editors, *Proceedings of the Italian Association for Artificial Intelligence on Topics in Artificial Intelligence*, volume 992 of *LNAI*, pages 25–36, Berlin, 1995. Springer.
27. Andrea Bonarini. Delayed Reinforcement, Fuzzy Q-Learning and Fuzzy Logic Controllers. In Herrera and Verdegay [218], pages 447–466.
28. Andrea Bonarini. Delayed Reinforcement, Fuzzy Q-Learning and Fuzzy Logic Controllers. In F. Herrera and J. L. Verdegay, editors, *Genetic Algorithms and Soft Computing, (Studies in Fuzziness, 8)*, pages 447–466, Berlin, D, 1996. Physica-Verlag.
29. Andrea Bonarini. Evolutionary Learning of Fuzzy rules: competition and cooperation. In W. Pedrycz, editor, *Fuzzy Modelling: Paradigms and Practice*, pages 265–284. Norwell, MA: Kluwer Academic Press, 1996. ftp://ftp.elet.polimi.it/pub/Andrea.Bonarini/ELF/ELF-Pedrycz.ps.gz.
30. Andrea Bonarini. Anytime learning and adaptation of fuzzy logic behaviors. *Adaptive Behavior Journal*, 5(3–4):281–315, 1997.
31. Andrea Bonarini. Reinforcement Distribution to Fuzzy Classifiers. In *Proceedings of the IEEE World Congress on Computational Intelligence (WCCI) – Evolutionary Computation*, pages 51–56. IEEE Computer Press, 1998.
32. Andrea Bonarini. Comparing reinforcement learning algorithms applied to crisp and fuzzy learning classifier systems. In Banzhaf et al. [12], pages 52–59.
33. Andrea Bonarini and Filippo Basso. Learning to compose fuzzy behaviors for autonomous agents. *Int. Journal of Approximate Reasoning*, 17(4):409–432, 1997.
34. Andrea Bonarini, Marco Dorigo, V. Maniezzo, and D. Sorrenti. AutonoMouse: An Experiment in Grounded Behaviors. In *Proceedings of GAA91 – Second Italian Workshop on Machine Learning, Bari, Italy*, 1991.
35. Pierre Bonelli and Alexandre Parodi. An Efficient Classifier System and its Experimental Comparison with two Representative learning methods on three medical domains. In Booker and Belew [44], pages 288–295.

36. Pierre Bonelli, Alexandre Parodi, Sandip Sen, and Stewart W. Wilson. NEWBOOLE: A Fast GBML System. In *International Conference on Machine Learning*, pages 153–159, San Mateo, California, 1990. Morgan Kaufmann.
37. Lashon B. Booker. *Intelligent Behavior as an Adaptation to the Task Environment*. PhD thesis, The University of Michigan, 1982.
38. Lashon B. Booker. Improving the performance of genetic algorithms in classifier systems. In Grefenstette [200], pages 80–92.
39. Lashon B. Booker. Classifier Systems that Learn Internal World Models. *Machine Learning*, 3:161–192, 1988.
40. Lashon B. Booker. Triggered rule discovery in classifier systems. In Schaffer [354], pages 265–274.
41. Lashon B. Booker. Instinct as an Inductive Bias for Learning Behavioral Sequences. In Meyer and Wilson [291], pages 230–237.
42. Lashon B. Booker. Representing Attribute-Based Concepts in a Classifier System. In Rawlins [318], pages 115–127.
43. Lashon B. Booker. Viewing Classifier Systems as an Integrated Architecture. In *Collected Abstracts for the First International Workshop on Learning Classifier System (IWLCS-92)* [2]. October 6–8, NASA Johnson Space Center, Houston, Texas.
44. Lashon B. Booker and Richard K. Belew, editors. *Proceedings of the 4th International Conference on Genetic Algorithms (ICGA91)*. Morgan Kaufmann: San Francisco CA, July 1991.
45. Lashon B. Booker, David E. Goldberg, and John H. Holland. Classifier Systems and Genetic Algorithms. *Artificial Intelligence*, 40:235–282, 1989.
46. Lashon B. Booker, Rick L. Riolo, and John H. Holland. Learning and Representation in Classifier Systems. In Vassant Honavar and Leonard Uhr, editors, *Artificial Intelligence and Neural Networks*, pages 581–613. Academic Press, 1994.
47. H. Brown Cribbs III and Robert E. Smith. Classifier System Renaissance: New Analogies, New Directions. In Koza et al. [264], pages 547–552.
48. Will Browne. *The Development of an Industrial Learning Classifier System for Application to a Steel Hot Strip Mill*. PhD thesis, University of Wales, Cardiff, 1999.
49. Will Browne, Karen Holford, and Carolynne Moore. An Industry Based Development of the Learning Classifier System Technique. Submitted to: 4th International Conference on Adaptive Computing in Design and Manufacturing (ACDM 2000).
50. Will Browne, Karen Holford, Carolynne Moore, and John Bullock. The implementation of a learning classifier system for parameter identification by signal processing of data from steel strip downcoilers. In A. T. Augousti, editor, *Software in Measurement. IEE Computer and Control Division*, 1996.
51. Will Browne, Karen Holford, Carolynne Moore, and John Bullock. A Practical Application of a Learning Classifier System for Downcoiler Decision Support in a Steel Hot Strip Mill. *Ironmaking and Steelmaking*, 25(1):33–41, 1997. Engineering Doctorate Seminar '97. Swansea, Wales, Sept. 2nd, 1997.
52. Will Browne, Karen Holford, Carolynne Moore, and John Bullock. A Practical Application of a Learning Classifier System in a Steel Hot Strip Mill. In Smith et al. [372], pages 611–614.
53. Will Browne, Karen Holford, Carolynne Moore, and John Bullock. An Industrial Learning Classifier System: The Importance of Pre-Processing Real Data and Choice of Alphabet. *To appear in: Engineering Applications of Artificial Intelligence*, 1999.

54. Bryan G. Spohn and Philip H. Crowley. Complexity of Strategies and the Evolution of Cooperation. In Koza et al. [263], pages 521–528.
55. Larry Bull. *Artificial Symbiology: evolution in cooperative multi-agent environments*. PhD thesis, University of the West of England, 1995.
56. Larry Bull. Artificial Symbiogenesis. *Artificial Life*, 2(3):269–292, 1996.
57. Larry Bull. On ZCS in Multi-agent Environments. *Lecture Notes in Computer Science*, 1498:471–480, 1998.
58. Larry Bull. On Evolving Social Systems. *Computational and Mathematical Organization Theory*, 5(3):281–298, 1999.
59. Larry Bull. On using ZCS in a Simulated Continuous Double-Auction Market. In Banzhaf et al. [12], pages 83–90.
60. Larry Bull and Terence C. Fogarty. Coevolving Communicating Classifier Systems for Tracking. In Albrecht et al. [5], pages 522–527.
61. Larry Bull and Terence C. Fogarty. Evolving Cooperative Communicating Classifier Systems. In A. V. Sebald and L. J. Fogel, editors, *Proceedings of the Third Annual Conference on Evolutionary Programming*, pages 308–315, 1994.
62. Larry Bull and Terence C. Fogarty. Parallel Evolution of Communcating Classifier Systems. In *Proceedings of the 1994 IEEE Conference on Evolutionary Computing*, pages 680–685. IEEE, 1994.
63. Larry Bull and Terence C. Fogarty. Evolutionary Computing in Cooperative Multi-Agent Systems. In Sandip Sen, editor, *Proceedings of the 1996 AAAI Symposium on Adaptation, Coevolution and Learning in Multi-Agent Systems*, pages 22–27. AAAI, 1996.
64. Larry Bull and Terence C. Fogarty. Evolutionary Computing in Multi-Agent Environments: Speciation and Symbiogenesis. In H-M. Voigt, W. Ebeling, I. Rechenberg, and H-P. Schwefel, editors, *Parallel Problem Solving from Nature – PPSN IV*, pages 12–21. Springer-Verlag, 1996.
65. Larry Bull, Terence C. Fogarty, S. Mikami, and J. G. Thomas. Adaptive Gait Acquisition using Multi-agent Learning for Wall Climbing Robots. In *Automation and Robotics in Construction XII*, pages 80–86, 1995.
66. Larry Bull, Terence C. Fogarty, and M. Snaith. Evolution in Multi-agent Systems: Evolving Communicating Classifier Systems for Gait in a Quadrupedal Robot. In Eshelman [148], pages 382–388.
67. Larry Bull and O. Holland. Internal and External Representations: A Comparison in Evolving the Ability to Count. In *Proceedings of the First Annual Society for the Study of Artificial Intelligence and Simulated Behaviour Robotics Workshop*, pages 11–14, 1994.
68. Martin Butz, David E. Goldberg, and Wolfgang Stolzmann. New challenges for an ACS: Hard problems and possible solutions. Technical Report 99019, University of Illinois at Urbana-Champaign, Urbana, IL, October 1999.
69. C. L. Ramsey and John J. Grefenstette. Case-based initialization of genetic algorithms. In Forrest [173], pages 84–91. http://www.ib3.gmu.edu/gref/.
70. Alessio Camilli. Classifier systems in massively parallel architectures. Master's thesis, University of Pisa, 1990. (In Italian).
71. Alessio Camilli and Roberto Di Meglio. Sistemi a classificatori su architetture a parallelismo massiccio. Technical report, Univ. Delgi Studi di Pisa, 1989.
72. Alessio Camilli, Roberto Di Meglio, F. Baiardi, M. Vanneschi, D. Montanari, and R. Serra. Classifier System Parallelization on MIMD Architectures. Technical Report 3/17, CNR, 1990.

73. Y. J. Cao, N. Ireson, Larry Bull, and R. Miles. Design of a Traffic Junction Controller using a Classifier System and Fuzzy Logic. In *Proceedings of the Sixth International Conference on Computational Intelligence, Theory, and Applications*. Springer-Verlag, 1999.
74. A. Carbonaro, G. Casadei, and A. Palareti. Genetic Algorithms and Classifier Systems in Simulating a Cooperative Behavior. In Albrecht et al. [5], pages 479–483.
75. Brian Carse. Learning Anticipatory Behaviour Using a Delayed Action Classifier System. In Fogarty [160], pages 210–223.
76. Brian Carse and Terence C. Fogarty. A delayed-action classifier system for learning in temporal environments. In *Proceedings of the 1st IEEE Conference on Evolutionary Computation*, volume 2, pages 670–673, 1994.
77. Brian Carse and Terence C. Fogarty. A Fuzzy Classifier System Using the Pittsburgh Approach. In Davidor and Schwefel [102], pages 260–269.
78. Brian Carse, Terence C. Fogarty, and A. Munro. Evolving fuzzy rule based controllers using genetic algorithms. *International Journal for Fuzzy Sets and Systems*, 80:273–293, 1996.
79. Brian Carse, Terence C. Fogarty, and A. Munro. The Temporal Fuzzy Classifier System and its Application to Distributed Control in a Homogeneous Multi-Agent ecology. In Goodman et al. [198], pages 76–86.
80. Brian Carse, Terence C. Fogarty, and Alistair Munro. Evolving Temporal Fuzzy Rule-Bases for Distributed Routing Control in Telecommunication Networks. In Herrera and Verdegay [218], pages 467–488.
81. Brian Carse, Terence C. Fogarty, and Alistair Munro. Artificial evolution of fuzzy rule bases which represent time: A temporal fuzzy classifier system. *International Journal of Intelligent Systems*, 13(issue 10-11):905–927, 1998.
82. G. Casadei, A. Palareti, and G. Proli. Classifier System in Traffic Management. In Albrecht et al. [5], pages 620–627.
83. Keith Chalk and George D. Smith. Multi-Agent Classifier Systems and the Iterated Prisoner's Dilemma. In Smith et al. [372], pages 615–618.
84. Keith W. Chalk and George D. Smith. The Co-evolution of Classifier Systems in a Competitive Environment. Poster presented at AISB94. Authors were from the University of East Anglia, U.K.
85. Dave Cliff and S. G. Bullock. Adding 'Foveal Vision' to Wilson's Animat. *Adaptive Behavior*, 2(1):47–70, 1993.
86. Dave Cliff, Philip Husbands, Jean-Arcady Meyer, and Stewart W. Wilson, editors. *From Animals to Animats 3. Proceedings of the Third International Conference on Simulation of Adaptive Behavior (SAB94)*. A Bradford Book. MIT Press, 1994.
87. Dave Cliff and Susi Ross. Adding Temporary Memory to ZCS. *Adaptive Behavior*, 3(2):101–150, 1995. Also technical report: ftp://ftp.cogs.susx.ac.uk/pub/reports/csrp/csrp347.ps.Z.
88. Dave Cliff and Susi Ross. Adding Temporary Memory to ZCS. Technical Report CSRP347, School of Cognitive and Computing Sciences, University of Sussex, 1995. ftp://ftp.cogs.susx.ac.uk/pub/reports/csrp/csrp347.ps.Z.
89. H. G. Cobb and John J. Grefenstette. Learning the persistence of actions in reactive control rules. In *Proceedings 8th International Machine Learning Workshop*, pages 293–297. Morgan Kaufmann, 1991.
90. Philippe Collard and Cathy Escazut. Relational Schemata: A Way to Improve the Expressiveness of Classifiers. In Eshelman [148], pages 397–404.
91. Marco Colombetti and Marco Dorigo. Learning to Control an Autonomous Robot by Distributed Genetic Algorithms. In Roitblat and Wilson [342], pages 305–312.

92. Marco Colombetti and Marco Dorigo. Robot Shaping: Developing Situated Agents through Learning. Technical Report TR-92-040, International Computer Science Institute, Berkeley, CA, 1993.
93. Marco Colombetti and Marco Dorigo. Training Agents to Perform Sequential Behavior. Technical Report TR-93-023, International Computer Science Institute, Berkeley, CA, September 1993.
94. Marco Colombetti and Marco Dorigo. Training agents to perform sequential behavior. *Adaptive Behavior*, 2(3):247–275, 1994. ftp://iridia.ulb.ac.be/pub/dorigo/journals/IJ.06-ADAP94.ps.gz.
95. Marco Colombetti and Marco Dorigo. Verso un'ingegneria del comportamento. *Rivista di Automatica, Elettronica e Informatica*, 83(10), 1996. In Italian.
96. Marco Colombetti and Marco Dorigo. Evolutionary Computation in Behavior Engineering. In *Evolutionary Computation: Theory and Applications*, chapter 2, pages 37–80. World Scientific Publishing Co.: Singapore, 1999. Also Tech. Report. TR/IRIDIA/1996-1, IRIDIA, Université Libre de Bruxelles.
97. Marco Colombetti, Marco Dorigo, and G. Borghi. Behavior Analysis and Training: A Methodology for Behavior Engineering. *IEEE Transactions on Systems, Man and Cybernetics*, 26(6):365–380, 1996.
98. Marco Colombetti, Marco Dorigo, and G. Borghi. Robot shaping: The HAMSTER Experiment. In M. Jamshidi et al., editor, *Proceedings of ISRAM'96, Sixth International Symposium on Robotics and Manufacturing, May 28–30, Montpellier, France*, 1996.
99. M. Compiani, D. Montanari, R. Serra, and P. Simonini. Asymptotic dynamics of classifier systems. In Schaffer [354], pages 298–303.
100. M. Compiani, D. Montanari, R. Serra, and P. Simonini. Learning and Bucket Brigade Dynamics in Classifier Systems. In *Special issue of Physica D (Vol. 42)* [1], pages 202–212.
101. M. Compiani, D. Montanari, R. Serra, and G. Valastro. Classifier systems and neural networks. In *Parallel Architectures and Neural Networks–First Italian Workshop*, pages 105–118. World Scientific, Teaneck, NJ, 1989.
102. Y. Davidor and H.-P. Schwefel, editors. *Parallel Problem Solving From Nature – PPSN III*, volume 866 of *Lecture Notes in Computer Science*, Berlin, 1994. Springer Verlag.
103. Lawrence Davis. Mapping Classifier Systems into Neural Networks. In *Proceedings of the Workshop on Neural Information Processing Systems 1*, pages 49–56, 1988.
104. Lawrence Davis, editor. *Genetic Algorithms and Simulated Annealing*, Research Notes in Artificial Intelligence. Pitman Publishing: London, 1989.
105. Lawrence Davis. Mapping Neural Networks into Classifier Systems. In Schaffer [354], pages 375–378.
106. Lawrence Davis. Covering and Memory in Classifier Systems. In *Collected Abstracts for the First International Workshop on Learning Classifier System (IWLCS-92)* [2]. October 6–8, NASA Johnson Space Center, Houston, Texas.
107. Lawrence Davis and David Orvosh. The Mating Pool: A Testbed for Experiments in the Evolution of Symbol Systems. In Eshelman [148], pages 405–??
108. Lawrence Davis, Stewart W. Wilson, and David Orvosh. Temporary Memory for Examples can Speed Learning in a Simple Adaptive System. In Roitblat and Wilson [342], pages 313–320.
109. Lawrence Davis and D. K. Young. Classifier Systems with Hamming Weights. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 162–173. Morgan Kaufmann, 1988.

110. Bart de Boer. Classifier Systems: a useful approach to machine learning? Master's thesis, Leiden University, 1994. ftp://ftp.wi.leidenuniv.nl/pub/CS/MScTheses/deboer.94.ps.gz.
111. Kenneth A. De Jong. Learning with Genetic Algorithms: An Overview. *Machine Learning*, 3:121–138, 1988.
112. Daniel Derrig and James Johannes. Deleting End-of-Sequence Classifiers. In John R. Koza, editor, *Late Breaking Papers at the Genetic Programming 1998 Conference*, University of Wisconsin, Madison, Wisconsin, USA, July 1998. Stanford University Bookstore.
113. Daniel Derrig and James D. Johannes. Hierarchical Exemplar Based Credit Allocation for Genetic Classifier Systems. In Koza et al. [262], pages 622–628.
114. L. Desjarlais and Stephanie Forrest. Linked learning in classifier systems: A control architecture for mobile robots. In *Collected Abstracts for the First International Workshop on Learning Classifier System (IWLCS-92)* [2]. October 6–8, NASA Johnson Space Center, Houston, Texas.
115. P. Devine, R. Paton, and M. Amos. Adaptation of Evolutionary Agents in Computational Ecologies. In *BCEC-97, Sweden*, 1997.
116. Jean-Yves Donnart and Jean-Arcady Meyer. A hierarchical classifier system implementing a motivationally autonomous animat. In Cliff et al. [86], pages 144–153.
117. Jean-Yves Donnart and Jean-Arcady Meyer. Hierarchical-map Building and Self-positioning with MonaLysa. *Adaptive Behavior*, 5(1):29–74, 1996.
118. Jean-Yves Donnart and Jean-Arcady Meyer. Spatial Exploration, Map Learning, and Self-Positioning with MonaLysa. In Maes et al. [284], pages 204–213.
119. Marco Dorigo. Message-Based Bucket Brigade: An Algorithm for the Apportionment of Credit Problem. In Y. Kodratoff, editor, *Proceedings of European Working Session on Learning '91, Porto, Portugal*, number 482 in Lecture notes in Artificial Intelligence, pages 235–244. Springer-Verlag, 1991.
120. Marco Dorigo. New perspectives about default hierarchies formation in learning classifier systems. In E. Ardizzone, E. Gaglio, and S. Sorbello, editors, *Proceedings of the 2nd Congress of the Italian Association for Artificial Intelligence (AI*IA) on Trends in Artificial Intelligence*, volume 549 of *LNAI*, pages 218–227, Palermo, Italy, October 1991. Springer Verlag.
121. Marco Dorigo. Using Transputers to Increase Speed and Flexibility of Genetic-based Machine Learning Systems. *Microprocessing and Microprogramming*, 34:147–152, 1991.
122. Marco Dorigo. Alecsys and the AutonoMouse: Learning to Control a Real Robot by Distributed Classifier System. Technical Report 92-011, Politecnico di Milano, 1992.
123. Marco Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, Italy, 1992. (In Italian).
124. Marco Dorigo. Genetic and Non-Genetic Operators in ALECSYS. *Evolutionary Computation*, 1(2):151–164, 1993. Also Tech. Report TR-92-075 International Computer Science Institute.
125. Marco Dorigo. Gli Algoritmi Genetici, i Sistemi a Classificatori e il Problema dell'Animat. *Sistemi Intelligenti*, 3(93):401–434, 1993. In Italian.
126. Marco Dorigo. Alecsys and the AutonoMouse: Learning to Control a Real Robot by Distributed Classifier Systems. *Machine Learning*, 19:209–240, 1995.
127. Marco Dorigo. The Robot Shaping Approach to Behavior Engineering. Thése d'Agrégation de l'Enseignement Supérieur, Faculté des Sciences Appliquées, Université Libre de Bruxelles, pp.176, 1995.

128. Marco Dorigo and Hugues Bersini. A Comparison of Q-Learning and Classifier Systems. In Cliff et al. [86], pages 248–255.
129. Marco Dorigo and Marco Colombetti. Robot shaping: Developing autonomous agents through learning. *Artificial Intelligence*, 2:321–370, 1994. ftp://iridia.ulb.ac.be/pub/dorigo/journals/IJ.05-AIJ94.ps.gz.
130. Marco Dorigo and Marco Colombetti. The Role of the Trainer in Reinforcement Learning. In S. Mahadevan et al., editor, *Proceedings of MLC-COLT '94 Workshop on Robot Learning, July 10th, New Brunswick, NJ*, pages 37–45, 1994.
131. Marco Dorigo and Marco Colombetti. Précis of Robot Shaping: An Experiment in Behavior Engineering. *Special Issue on Complete Agent Learning in Complex Environment, Adaptive Behavior*, 5(3–4):391–405, 1997.
132. Marco Dorigo and Marco Colombetti. Reply to Dario Floreano's "Engineering Adaptive Behavior". *Special Issue on Complete Agent Learning in Complex Environment, Adaptive Behavior*, 5(3–4):417–420, 1997.
133. Marco Dorigo and Marco Colombetti. *Robot Shaping: An Experiment in Behavior Engineering*. MIT Press/Bradford Books, 1998.
134. Marco Dorigo and V. Maniezzo. Parallel Genetic Algorithms: Introduction and Overview of Current Research. In J. Stenders, editor, *Parallel Genetic Algorithms: Theory and Applications*, Amsterdam, 1992. IOS Press.
135. Marco Dorigo, V. Maniezzo, and D. Montanari. Classifier-based robot control systems. In *IFAC/IFIP/IMACS International Symposium on Artificial Intelligence in Real-Time Control*, pages 591–598, Delft, Netherlands, 1992.
136. Marco Dorigo, Mukesh J. Patel, and Marco Colombetti. The effect of Sensory Information on Reinforcement Learning by a Robot Arm. In M. Jamshidi et al., editor, *Proceedings of ISRAM'94, Fifth International Symposium on Robotics and Manufacturing, August 14–18, Maui, HI*, pages 83–88. ASME Press, 1994.
137. Marco Dorigo and U. Schnepf. Organisation of Robot Behaviour Through Genetic Learning Processes. In *Proceedings of ICAR'91 – Fifth IEEE International Conference on Advanced Robotics, Pisa, Italy*, pages 1456–1460. IEEE Press, 1991.
138. Marco Dorigo and U. Schnepf. Genetics-based Machine Learning and Behaviour Based Robotics: A New Synthesis. *IEEE Transactions on Systems, Man and Cybernetics*, 23(1):141–154, 1993.
139. Marco Dorigo and E. Sirtori. A Parallel Environment for Learning Systems. In *Proceedings of GAA91 – Second Italian Workshop on Machine Learning, Bari, Italy*, 1991.
140. Marco Dorigo and Enrico Sirtori. Alecsys: A Parallel Laboratory for Learning Classifier Systems. In Booker and Belew [44], pages 296–302.
141. Barry B. Druhan and Robert C. Mathews. THIYOS: A Classifier System Model of Implicit Knowledge in Artificial Grammars. In *Proc. Ann. Cog. Sci. Soc.*, 1989.
142. Daniel Eckert and Johann Mitlöhner. Modeling individual and endogenous learning in games: the relevance of classifier systems. In *Complex Modeling for Socio-Economic Systems, SASA, Vienna*, 1997.
143. Daniel Eckert, Johann Mitlöhner, and Makus Moschner. Evolutionary stability issues and adaptive learning in classifier systems. In *OR'97 Conference on Operations Research, Vienna*, 1997.
144. G. Enee and C. Escazut. Classifier systems evolving multi-agent system with distributed elitism. In Angeline et al. [6], pages 1740–1745.
145. Cathy Escazut and Philippe Collard. Learning Disjunctive Normal Forms in a Dual Classifier System. In Nada Lavrač and Stefan Wrobel, editors, *Proceedings of the 8th European Conference on Machine Learning*, volume 912 of *LNAI*, pages 271–274. Springer, 1995.

146. Cathy Escazut, Philippe Collard, and Jean-Louis Cavarero. Dynamic Management of the Specificity in Classifier Systems. In Albrecht et al. [5], pages 484–491.
147. Cathy Escazut and Terence C. Fogarty. Coevolving Classifier Systems to Control Traffic Signals. In John R. Koza, editor, *Late Breaking Papers at the 1997 Genetic Programming Conference*, Stanford University, CA, USA, July 1997. Stanford Bookstore.
148. Larry J. Eshelman, editor. *Proceedings of the 6th International Conference on Genetic Algorithms (ICGA95)*. Morgan kaufmann Publishers: San Francisco CA, 1995.
149. Andrew Fairley and Derek F. Yates. Improving Simple Classifier Systems to alleviate the problems of Duplication, Subsumption and Equivalence of Rules. In Albrecht et al. [5], pages 408–416.
150. Andrew Fairley and Derek F. Yates. Inductive Operators and Rule Repair in a Hybrid Genetic Learning System: Some Initial Results. In Fogarty [160], pages 166–179.
151. J. Doyne Farmer. A Rosetta Stone for Connectionism. In *Special issue of Physica D (Vol. 42)* [1], pages 153–187.
152. J. Doyne Farmer, N. H. Packard, and A. S. Perelson. The Immune System, Adaptation & Learning. *Physica D*, 22:187–204, 1986.
153. Francine Federman and Susan Fife Dorchak. Information Theory and NEXT-PITCH: A Learning Classifier System. In Bäck [8], pages 442–449.
154. Francine Federman and Susan Fife Dorchak. Representation of Music in a Learning Classifier System. In Rad and Skowron, editors, *Foundations of Intelligent Systems: Proceedings 10th International Symposium (ISMIS'97)*. Springer-Verlag: Heidelberg, 1997.
155. Francine Federman and Susan Fife Dorchak. A Study of Classifier Length and Population Size. In Koza et al. [262], pages 629–634.
156. Rhonda Ficek. Genetic Algorithms. Technical Report NDSU-CS-TR-90-51, North Dakota State University. Computer Science and Operations Research, 1997.
157. Gary William Flake. *The Computational Beauty of Nature*. MIT Press, 1998. (Contains a chapter on ZCS).
158. Peter Fletcher. Simulating the use of 'fiat money' in a simple commodity economy. Master's thesis, Schools of Psychology and Computer Science, University of Birmingham, 1996.
159. Terence C. Fogarty. Co-evolving Co-operative Populations of Rules in Learning Control Systems. In *Evolutionary Computing, AISB Workshop Selected Papers* [160], pages 195–209.
160. Terence C. Fogarty, editor. *Evolutionary Computing, AISB Workshop Selected Papers*, number 865 in Lecture Notes in Computer Science. Springer-Verlag, 1994.
161. Terence C. Fogarty. Learning new rules and adapting old ones with the genetic algorithm. In G. Rzevski, editor, *Artificial Intelligence in Manufacturing*, pages 275–290. Springer-Verlag, 1994.
162. Terence C. Fogarty. Optimising Individual Control Rules and Multiple Communicating Rule-based Control Systems with Parallel Distributed Genetic Algorithms. *IEE Journal of Control Theory and Applications*, 142(3):211–215, 1995.
163. Terence C. Fogarty, Larry Bull, and Brian Carse. Evolving Multi-Agent Systems. In J. Periaux and G. Winter, editors, *Genetic Algorithms in Engineering and Computer Science*, pages 3–22. John Wiley & Sons, 1995.
164. Terence C. Fogarty, Brian Carse, and Larry Bull. Classifier Systems – recent research. *AISB Quarterly*, 89:48–54, 1994.

165. Terence C. Fogarty, Brian Carse, and Larry Bull. Classifier Systems: selectionist reinforcement learning, fuzzy rules and communication. Presented at the First International Workshop on Biologically Inspired Evolutionary Systems, Tokyo, 1995.
166. Terence C. Fogarty, Brian Carse, and A. Munro. Artificial evolution of fuzzy rule bases which represent time: A temporal fuzzy classifier system. *International Journal of Intelligent Systems*, 13(10–11):906–927, 1998.
167. Terence C. Fogarty, N. S. Ireson, and Larry Bull. Genetic-based Machine Learning – Applications in Industry and Commerce. In *Applications of Modern Heuristic Methods*, pages 91–110. Alfred Waller Ltd, 1995.
168. David B. Fogel. *Evolutionary Computation. The Fossil Record. Selected Readings on the History of Evolutionary Computation*, chapter 16: Classifier Systems. IEEE Press, 1998. This is a reprint of (Holland and Reitman, 1978), with an added introduction by Fogel.
169. Stephanie Forrest. *A study of parallelism in the classifier system and its application to classification in KL-ONE semantic networks*. PhD thesis, University of Michigan, Ann Arbor, MI, 1985.
170. Stephanie Forrest. Implementing semantic network structures using the classifier system. In Grefenstette [200], pages 24–44.
171. Stephanie Forrest. The Classifier System: A Computational Model that Supports Machine Intelligence. In *International Conference on Parallel Processing*, pages 711–716, Los Alamitos, Ca., USA, August 1986. IEEE Computer Society Press.
172. Stephanie Forrest. *Parallelism and Programming in Classifier Systems*. Pittman, London, 1991.
173. Stephanie Forrest, editor. *Proceedings of the 5th International Conference on Genetic Algorithms (ICGA93)*. Morgan Kaufmann, 1993.
174. Stephanie Forrest and John H. Miller. Emergent behavior in classifier systems. In *Special issue of Physica D (Vol. 42)* [1], pages 213–217.
175. Stephanie Forrest, Robert E. Smith, and A. Perelson. Maintaining diversity with a genetic algorithm. In *Collected Abstracts for the First International Workshop on Learning Classifier System (IWLCS-92)* [2]. October 6–8, NASA Johnson Space Center, Houston, Texas.
176. Francesc Xavier Llorà i Fàbrega and Josep Maria Garrell i Guiu. GENIFER: A Nearest Neighbour based Classifier System using GA. In Banzhaf et al. [12], page 797. One page poster paper.
177. Francine Federman and Gayle Sparkman and Stephanie Watt. Representation of Music in a Learning Classifier System Utilizing Bach Chorales. In Banzhaf et al. [12], page 785. One page poster paper.
178. Peter W. Frey and David J. Slate. Letter Recognition Using Holland-Style Adaptive Classifiers. *Machine Learning*, 6:161–182, 1991.
179. Leeann L. Fu. The XCS Classifier System and Q-learning. In John R. Koza, editor, *Late Breaking Papers at the Genetic Programming 1998 Conference*, University of Wisconsin, Madison, Wisconsin, USA, 1998. Stanford University Bookstore.
180. Takeshi Furuhashi. A Proposal of Hierarchical Fuzzy Classifier Systems. In Forrest [173].
181. Takeshi Furuhashi, Ken Nakaoka, Koji Morikawa, and Yoshiki Uchikawa. Controlling Excessive Fuzziness in a Fuzzy Classifier System. In Forrest [173], pages 635–635.
182. Takeshi Furuhashi, Ken Nakaoka, and Yoshiki Uchikawa. A Study on Fuzzy Classifier System for Finding Control Knowledge of Multi-Input Systems. In Herrera and Verdegay [218], pages 489–502.

183. Santiago Garcia, Fermin Gonzalez, and Luciano Sanchez. Evolving Fuzzy Rule Based Classifiers with GA-P: A Grammatical Approach. In Riccardo Poli, Peter Nordin, William B. Langdon, and Terence C. Fogarty, editors, *Genetic Programming, Proceedings of EuroGP'99*, volume 1598 of *LNCS*, pages 203–210, Goteborg, Sweden, May 1999. Springer-Verlag.
184. Andreas Geyer-Schulz. Fuzzy Classifier Systems. In Robert Lowen and Marc Roubens, editors, *Fuzzy Logic: State of the Art*, Series D: System Theory, Knowledge Engineering and Problem Solving, pages 345–354, Dordrecht, 1993. Kluwer Academic Publishers.
185. Andreas Geyer-Schulz. *Fuzzy Rule-Based Expert Systems and Genetic Machine Learning.* Physica Verlag, 1995. Book review at: http://www.apl.demon.co.uk/aplandj/fuzzy.htm.
186. Andreas Geyer-Schulz. Holland Classifier Systems. In *Proceedings of the International Conference on APL (APL'95)*, volume 25, pages 43–55, New York, NY, USA, June 1995. ACM Press.
187. Antonella Giani. A Study of Parallel Cooperative Classifier Systems. In John R. Koza, editor, *Late Breaking Papers at the Genetic Programming 1998 Conference*, University of Wisconsin, Madison, Wisconsin, USA, July 1998. Stanford University Bookstore.
188. Antonella Giani, Fabrizio Baiardi, and Antonina Starita. Q-Learning in Evolutionary Rule-Based Systems. In Davidor and Schwefel [102], pages 270–289.
189. Antonella Giani, A. Sticca, F. Baiardi, and A. Starita. Q-learning and Redundancy Reduction in Classifier Systems with Internal State. In Claire Nédellec and Céline Rouveirol, editors, *Proceedings of the 10th European Conference on Machine Learning (ECML-98)*, volume 1398 of *LNAI*, pages 364–369. Springer, 1998.
190. A. H. Gilbert, Frances Bell, and Christine L. Valenzuela. Adaptive Learning of Process Control and Profit Optimisation using a Classifier System. *Evolutionary Computation*, 3(2):177–198, 1995.
191. Attilio Giordana and Filippo Neri. Search-intensive concept induction. *Evolutionary Computation*, 3:375–416, 1995.
192. David E. Goldberg. *Computer-Aided Gas Pipeline Operation using Genetic Algorithms and Rule Learning.* PhD thesis, The University of Michigan, 1983.
193. David E. Goldberg. Dynamic System Control using Rule Learning and Genetic Algorithms. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI-85)*, pages 588–592. Morgan Kaufmann, 1985.
194. David E. Goldberg. Genetic algorithms and rules learning in dynamic system control. In Grefenstette [200], pages 8–15.
195. David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, Mass., 1989.
196. David E. Goldberg. Probability Matching, the Magnitude of Reinforcement, and Classifier System Bidding. *Machine Learning*, 5:407–425, 1990. (Also TCGA tech report 88002, U. of Alabama).
197. David E. Goldberg, Jeffrey Horn, and Kalyanmoy Deb. What Makes a Problem Hard for a Classifier System? In *Collected Abstracts for the First International Workshop on Learning Classifier System (IWLCS-92)* [2]. (Also tech. report 92007 Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign). Available from ENCORE (ftp://ftp.krl.caltech.edu/pub/EC/Welcome.html) in the section on Classifier Systems.

198. E. G. Goodman, V. L. Uskov, and W. F. Punch, editors. *Proceedings of the First International Conference on Evolutionary Algorithms and their Application EVCA'96*, Moscow, 1996. The Presidium of the Russian Academy of Sciences.
199. David Perry Greene and Stephen F. Smith. Using Coverage as a Model Building Constraint in Learning Classifier Systems. *Evolutionary Computation*, 2(1):67–91, 1994.
200. John J. Grefenstette, editor. *Proceedings of the 1st International Conference on Genetic Algorithms and their Applications (ICGA85)*. Lawrence Erlbaum Associates: Pittsburgh, PA, July 1985.
201. John J. Grefenstette. Multilevel Credit Assignment in a Genetic Learning System. In *Proceedings of the 2nd International Conference on Genetic Algorithms (ICGA87)* [202], pages 202–207.
202. John J. Grefenstette, editor. *Proceedings of the 2nd International Conference on Genetic Algorithms (ICGA87)*, Cambridge, MA, July 1987. Lawrence Erlbaum Associates.
203. John J. Grefenstette. Credit Assignment in Rule Discovery Systems Based on Genetic Algorithms. *Machine Learning*, 3:225–245, 1988.
204. John J. Grefenstette. A System for Learning Control Strategies with Genetic Algorithms. In Schaffer [354], pages 183–190.
205. John J. Grefenstette. Lamarckian Learning in Multi-Agent Environments. In Booker and Belew [44], pages 303–310. http://www.ib3.gmu.edu/gref/publications.html.
206. John J. Grefenstette. Learning decision strategies with genetic algorithms. In *Proc. Intl. Workshop on Analogical and Inductive Inference*, volume 642 of *Lecture Notes in Artificial Intelligence*, pages 35–50. Springer-Verlag, 1992. http://www.ib3.gmu.edu/gref/.
207. John J. Grefenstette. The Evolution of Strategies for Multi-agent Environments. *Adaptive Behavior*, 1:65–89, 1992. http://www.ib3.gmu.edu/gref/.
208. John J. Grefenstette. Using a genetic algorithm to learn behaviors for autonomous vehicles. In *Proceedings American Institute of Aeronautics and Astronautics Guidance, Navigation and Control Conference*, pages 739–749. AIAA, 1992. http://www.ib3.gmu.edu/gref/.
209. John J. Grefenstette. Evolutionary Algorithms in Robotics. In M. Jamshedi and C. Nguyen, editors, *Robotics and Manufacturing: Recent Trends in Research, Education and Applications, v5. Proc. Fifth Intl. Symposium on Robotics and Manufacturing, ISRAM 94*, pages 65–72. ASME Press: New York, 1994. http://www.ib3.gmu.edu/gref/.
210. John J. Grefenstette and H. G. Cobb. User's guide for SAMUEL, Version 1.3. Technical Report NRL Memorandum Report 6820, Naval Research Laboratory, 1991.
211. John J. Grefenstette, C. L. Ramsey, and Alan C. Schultz. Learning Sequential Decision Rules using Simulation Models and Competition. *Machine Learning*, 5(4):355–381, 1990. http://www.ib3.gmu.edu/gref/publications.html.
212. Adrian Hartley. Genetics Based Machine Learning as a Model of Perceptual Category Learning in Humans. Master's thesis, University of Birmingham, 1998. ftp://ftp.cs.bham.ac.uk/pub/authors/T.Kovacs/index.html.
213. Adrian Hartley. Accuracy-based fitness allows similar performance to humans in static and dynamic classification environments. In Banzhaf et al. [12], pages 266–273.
214. U. Hartmann. Efficient Parallel Learning in Classifier Systems. In Albrecht et al. [5], pages 515–521.

215. U. Hartmann. On the Complexity of Learning in Classifier Systems. In Davidor and Schwefel [102], pages 280–289. Republished in: ECAI 94. 11th European Conference on Artificial Intelligence. A Cohn (Ed.), pp.438–442, 1994. John Wiley and Sons.
216. Marianne Haslev. A Classifier System for the Production by Computer of Past Tense Verb-Forms. Presented at a Genetic Algorithms Workshop at the Rowland Institute, Cambridge MA, Nov 1986, 1986.
217. Luis Miramontes Hercog. Hand-eye coordination: An evolutionary approach. Master's thesis, Department of Artificial Intelligence. University of Edinburgh, 1998.
218. F. Herrera and J. L. Verdegay, editors. *Genetic Algorithms and Soft Computing, (Studies in Fuzziness, 8)*. Physica-Verlag, Berlin, 1996.
219. M. R. Hilliard, G. E. Liepins, Mark Palmer, Michael Morrow, and Jon Richardson. A classifier based system for discovering scheduling heuristics. In Grefenstette [202], pages 231–235.
220. John H. Holland. Processing and processors for schemata. In E. L. Jacks, editor, *Associative information processing*, pages 127–146. New York: American Elsevier, 1971.
221. John H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, 1975. Republished by the MIT press, 1992.
222. John H. Holland. Adaptation. In R. Rosen and F. M. Snell, editors, *Progress in theoretical biology.* New York: Plenum, 1976.
223. John H. Holland. Adaptive algorithms for discovering and using general patterns in growing knowledge bases. *International Journal of Policy Analysis and Information Systems*, 4(3):245–268, 1980.
224. John H. Holland. Genetic Algorithms and Adaptation. Technical Report 34, University of Michigan. Department of Computer and Communication Sciences, Ann Arbor, 1981.
225. John H. Holland. Escaping brittleness. In *Proceedings Second International Workshop on Machine Learning*, pages 92–95, 1983.
226. John H. Holland. Properties of the bucket brigade. In Grefenstette [200], pages 1–7.
227. John H. Holland. A Mathematical Framework for Studying Learning in a Classifier System. In Doyne Farmer, Alan Lapedes, Norman Packard, and Burton Wendroff, editors, *Evolution, Games and Learning: Models for Adaptation in Machines and Nature*, pages 307–317, Amsterdam, 1986. North-Holland.
228. John H. Holland. A Mathematical Framework for Studying Learning in Classifier Systems. *Physica D*, 22:307–317, 1986.
229. John H. Holland. Escaping Brittleness: The possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems. In Mitchell, Michalski, and Carbonell, editors, *Machine learning, an artificial intelligence approach. Volume II*, chapter 20, pages 593–623. Morgan Kaufmann, 1986.
230. John H. Holland. Genetic Algorithms and Classifier Systems: Foundations and Future Directions. In Grefenstette [202], pages 82–89.
231. John H. Holland. Concerning the Emergence of Tag-Mediated Lookahead in Classifier Systems. In *Special issue of Physica D (Vol. 42)* [1], pages 188–201.
232. John H. Holland and Arthur W. Burks. Adaptive Computing System Capable of Learning and Discovery. Patent 4697242 United States 29 Sept., 1987.
233. John H. Holland, Keith J. Holyoak, Richard E. Nisbett, and P. R. Thagard. *Induction: Processes of Inference, Learning, and Discovery.* MIT Press, Cambridge, 1986.

234. John H. Holland, Keith J. Holyoak, Richard E. Nisbett, and Paul R. Thagard. Classifier Systems, Q-Morphisms, and Induction. In Davis [104], pages 116–128.
235. John H. Holland and J. S. Reitman. Cognitive systems based on adaptive algorithms. In D. A. Waterman and F. Hayes-Roth, editors, *Pattern-directed inference systems*. New York: Academic Press, 1978. Reprinted in: Evolutionary Computation. The Fossil Record. David B. Fogel (Ed.) IEEE Press, 1998. ISBN: 0-7803-3481-7.
236. John H. Holmes. *Evolution-Assisted Discovery of Sentinel Features in Epidemiologic Surveillance*. PhD thesis, Drexel Univesity, 1996. http://cceb.med.upenn.edu/holmes/disstxt.ps.gz.
237. John H. Holmes. Discovering Risk of Disease with a Learning Classifier System. In Bäck [8]. http://cceb.med.upenn.edu/holmes/icga97.ps.gz.
238. John H. Holmes. Differential negative reinforcement improves classifier system learning rate in two-class problems with unequal base rates. In Koza et al. [262], pages 635–642. http://cceb.med.upenn.edu/holmes/gp98.ps.gz.
239. Keith J. Holyoak, K. Koh, and Richard E. Nisbett. A Theory of Conditioning: Inductive Learning within Rule-Based Default Hierarchies. *Psych. Review*, 96:315–340, 1990.
240. Jeffrey Horn and David E. Goldberg. Natural Niching for Cooperative Learning in Classifier Systems. In Koza et al. [264], pages 553–564.
241. Jeffrey Horn and David E. Goldberg. Towards a Control Map for Niching. In *Foundations of Genetic Algorithms*, pages 287–310, 1998.
242. Jeffrey Horn, David E. Goldberg, and Kalyanmoy Deb. Implicit Niching in a Learning Classifier System: Nature's Way. *Evolutionary Computation*, 2(1):37–66, 1994. Also IlliGAL Report No 94001, 1994.
243. Dijia Huang. A framework for the credit-apportionment process in rule-based systems. *IEEE Transaction on System Man and Cybernetics*, 1989.
244. Dijia Huang. *Credit Apportionment in Rule-Based Systems: Problem Analysis and Algorithm Synthesis*. PhD thesis, University of Michigan, 1989.
245. Dijia Huang. The Context-Array Bucket-Brigade Algorithm: An Enhanced Approach to Credit-Apportionment in Classifier Systems. In Schaffer [354], pages 311–316.
246. Francesc Xavier Llorà i Fàbrega. Automatic Classification using genetic algorithms under a Pittsburgh approach. Master's thesis, Enginyeria La Salle - Ramon Llull University, 1998. http://www.salleurl.edu/~xevil/Work/index.html.
247. Francesc Xavier Llorà i Fàbrega, Josep Maria Garrell i Guiu, and Ester Bernadó i Mansilla. A Classifier System based on Genetic Algorithm under the Pittsburgh aproach for problems with real valued attributes. In Viceng Torra, editor, *Proceedings of Artificial Intelligence Catalan Workshop (CCIA98)*, volume 14–15, pages 85–93. ACIA Press, 1998. In Catalan http://www.salleurl.edu/~xevil/Work/index.html.
248. Josep Maria Garrell i Guiu, Elisabet Golobardes i Ribé, Ester Bernadó i Mansilla, and Francesc Xavier Llorà i Fàbrega. Automatic Classification of mammary biopsy images with machine learning techniques. In E. Alpaydin, editor, *Proceedings of Engineering of Intelligent Systems (EIS'98)*, volume 3, pages 411–418. ICSC Academic Press, 1998. http://www.salleurl.edu/~xevil/Work/index.html.
249. Josep Maria Garrell i Guiu, Elisabet Golobardes i Ribé, Ester Bernadó i Mansilla, and Francesc Xavier Llorà i Fàbrega. Automatic Diagnosis with Genetic Algorithms and Case-Based Reasoning. *To appear in AIENG Journal*, 1999. (This is an expanded version of Guiu98a).

250. H. Iba, H. de Garis, and T. Higuchi. Evolutionary Learning of Predatory Behaviors Based on Structured Classifiers. In Roitblat and Wilson [342], pages 356–363.
251. John H. Holmes. Evaluating Learning Classifier System Performance In Two-Choice Decision Tasks: An LCS Metric Toolkit. In Banzhaf et al. [12], page 789. One page poster paper.
252. John J. Grefenstette and Alan C. Schultz. An evolutionary approach to learning in robots. In *Machine Learning Workshop on Robot Learning*, New Brunswick, NJ, 1994. http://www.ib3.gmu.edu/gref/.
253. Kenneth A. De Jong and William M. Spears. Learning Concept Classification Rules using Genetic Algorithms. In *Proceedings of the Twelth International Conference on Artificial Intelligence IJCAI-91*, volume 2, 1991.
254. Hiroaki Kitano, Stephen F. Smith, and Tetsuya Higuchi. GA-1: A Parallel Associative Memory Processor for Rule Learning with Genetic Algorithms. In Booker and Belew [44], pages 311–317.
255. Leslie Knight and Sandip Sen. PLEASE: A Prototype Learning System using Genetic Algorithms. In Eshelman [148], pages 429–??
256. Kostyantyn Korovkin and Robert Richards. Visual Auction: A Classifier System Pedagogical and Researcher Tool. In Scott Brave and Annie S. Wu, editors, *Late Breaking Papers at the 1999 Genetic and Evolutionary Computation Conference (GECCO-99)*, pages 159–163, 1999.
257. Tim Kovacs. Evolving Optimal Populations with XCS Classifier Systems. Master's thesis, School of Computer Science, University of Birmingham, Birmingham, U.K., 1996. Also tech. report CSR-96-17 and CSRP-96-17 ftp://ftp.cs.bham.ac.uk/pub/tech-reports/1996/CSRP-96-17.ps.gz.
258. Tim Kovacs. Steady State Deletion Techniques in a Classifier System. Unpublished document – partially subsumed by Kovacs1999a 'Deletion Schemes for Classifier Systems', 1997.
259. Tim Kovacs. XCS Classifier System Reliably Evolves Accurate, Complete, and Minimal Representations for Boolean Functions. In Roy, Chawdhry, and Pant, editors, *Soft Computing in Engineering Design and Manufacturing*, pages 59–68. Springer-Verlag, London, 1997. ftp://ftp.cs.bham.ac.uk/pub/authors/T.Kovacs/index.html.
260. Tim Kovacs. XCS Classifier System Reliably Evolves Accurate, Complete, and Minimal Representations for Boolean Functions. Technical Report Version. Technical Report CSRP-97-19, School of Computer Science, University of Birmingham, Birmingham, U.K., 1997. http://www.cs.bham.ac.uk/system/tech-reports/tr.html.
261. Tim Kovacs. Deletion schemes for classifier systems. In Banzhaf et al. [12], pages 329–336. Also technical report CSRP-99-08, School of Computer Science, University of Birmingham. http://www.cs.bham.ac.uk/~tyk.
262. John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors. *Genetic Programming 1998: Proceedings of the Third Annual Conference*. Morgan Kaufmann: San Francisco, CA, 1998.
263. John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, Hitoshi Iba, and Rick Riolo, editors. *Genetic Programming 1997: Proceedings of the Second Annual Conference*. Morgan Kaufmann: San Francisco, CA, 1997.
264. John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors. *Genetic Programming 1996: Proceedings of the First Annual Conference*, Stanford University, CA, USA, 1996. MIT Press.

265. Pier Luca Lanzi. A Model of the Environment to Avoid Local Learning (An Analysis of the Generalization Mechanism of XCS). Technical Report 97.46, Politecnico di Milano. Department of Electronic Engineering and Information Sciences, 1997. http://ftp.elet.polimi.it/people/lanzi/report46.ps.gz.
266. Pier Luca Lanzi. A Study of the Generalization Capabilities of XCS. In Bäck [8], pages 418–425. http://ftp.elet.polimi.it/people/lanzi/icga97.ps.gz.
267. Pier Luca Lanzi. Solving Problems in Partially Observable Environments with Classifier Systems (Experiments on Adding Memory to XCS). Technical Report 97.45, Politecnico di Milano. Department of Electronic Engineering and Information Sciences, 1997. http://ftp.elet.polimi.it/people/lanzi/report45.ps.gz.
268. Pier Luca Lanzi. Adding Memory to XCS. In *Proceedings of the IEEE Conference on Evolutionary Computation (ICEC98)*. IEEE Press, 1998. http://ftp.elet.polimi.it/people/lanzi/icec98.ps.gz.
269. Pier Luca Lanzi. An analysis of the memory mechanism of XCSM. In Koza et al. [262], pages 643–651. http://ftp.elet.polimi.it/people/lanzi/gp98.ps.gz.
270. Pier Luca Lanzi. Generalization in Wilson's XCS. In A. E. Eiben, T. Bäck, M. Shoenauer, and H.-P Schwefel, editors, *Proceedings of the Fifth International Conference on Parallel Problem Solving From Nature – PPSN V*, number 1498 in LNCS. Springer Verlag, 1998.
271. Pier Luca Lanzi. *Reinforcement Learning by Learning Classifier Systems*. PhD thesis, Politecnico di Milano, 1998.
272. Pier Luca Lanzi. An Analysis of Generalization in the XCS Classifier System. *Evolutionary Computation*, 7(2):125–149, 1999.
273. Pier Luca Lanzi. Extending the Representation of Classifier Conditions Part I: From Binary to Messy Coding. In Banzhaf et al. [12], pages 337–344.
274. Pier Luca Lanzi. Extending the Representation of Classifier Conditions Part II: From Messy Coding to S-Expressions. In Banzhaf et al. [12], pages 345–352.
275. Pier Luca Lanzi and Marco Colombetti. An Extension of XCS to Stochastic Environments. Technical Report 98.85, Dipartimento di Elettronica e Informazione - Politecnico di Milano, 1998.
276. Pier Luca Lanzi and Marco Colombetti. An Extension to the XCS Classifier System for Stochastic Environments. In Banzhaf et al. [12], pages 353–360.
277. Pier Luca Lanzi and Stewart W. Wilson. Optimal classifier system performance in non-Markov environments. Technical Report 99.36, Dipartimento di Elettronica e Informazione - Politecnico di Milano, 1999. Also IlliGAL tech. report 99022, University of Illinois.
278. Blake Lebaron, W. Brian Arthur, and R. Palmer. The Time Series Properties of an Artificial Stock Market. *Journal of Economic Dynamics and Control*, 1999.
279. T. Lettau and H. Uhlig. Rules of Thumb and Dynamic Programming. Technical report, Department of Economics, Princeton University, 1994.
280. Gunar E. Liepins, M. R. Hillard, M. Palmer, and G. Rangarajan. Credit Assignment and Discovery in Classifier Systems. *International Journal of Intelligent Systems*, 6:55–69, 1991.
281. Gunar E. Liepins, Michael R. Hilliard, Mark Palmer, and Gita Rangarajan. Alternatives for Classifier System Credit Assignment. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, pages 756–761, 1989.
282. Gunar E. Liepins and Lori A. Wang. Classifier System Learning of Boolean Concepts. In Booker and Belew [44], pages 318–323.

283. Derek A. Linkens and H. Okola Nyongesah. Genetic Algorithms for fuzzy control - Part II: Off-line system development and application. Technical Report CTA/94/2387/1st MS, Department of Automatic Control and System Engineering, University of Sheffield, U.K., 1994.
284. Pattie Maes, Maja J. Mataric, Jean-Arcady Meyer, Jordan Pollack, and Stewart W. Wilson, editors. *From Animals to Animats 4. Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior (SAB96).* A Bradford Book. MIT Press, 1996.
285. Chikara Maezawa and Masayasu Atsumi. Collaborative Learning Agents with Strutural Classifier Systems. In Banzhaf et al. [12], page 777. One page poster paper.
286. Bernard Manderick. Selectionist Categorization. In Schwefel and Männer [358], pages 326–330.
287. Ramon Marimon, Ellen McGrattan, and Thomas J. Sargent. Money as a Medium of Exchange in an Economy with Artificially Intelligent Agents. *Journal of Economic Dynamics and Control*, 14:329–373, 1990. Also Tech. Report 89-004, Santa Fe Institute, 1989.
288. Maja J Mataric. A comparative analysis of reinforcement learning methods. A.I. Memo No. 1322, Massachusetts Institute of Technology, 1991.
289. Alaster D. McAulay and Jae Chan Oh. Image Learning Classifier System Using Genetic Algorithms. In *Proceedings IEEE NAECON '89*, 1989.
290. Chris Melhuish and Terence C. Fogarty. Applying A Restricted Mating Policy To Determine State Space Niches Using Immediate and Delayed Reinforcement. In Fogarty [160], pages 224–237.
291. J. A. Meyer and S. W. Wilson, editors. *From Animals to Animats 1. Proceedings of the First International Conference on Simulation of Adaptive Behavior (SAB90).* A Bradford Book. MIT Press, 1990.
292. Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, 1996. Contains introductory chapter on LCS.
293. John H. Miller and Stephanie Forrest. The dynamical behavior of classifier systems. In Schaffer [354], pages 304–310.
294. Johann Mitlöhner. Classifier systems and economic modelling. In *APL '96. Proceedings of the APL 96 conference on Designing the future*, volume 26 (4), pages 77–86, 1996.
295. Chilukuri K. Mohan. *Expert Systems: A Modern Overview.* Kluwer, to appear in 2000. Contains an introdutory survey chapter on LCS.
296. D. Montanari. Classifier systems with a constant-profile bucket brigade. In *Collected Abstracts for the First International Workshop on Learning Classifier System (IWLCS-92)* [2]. October 6–8, NASA Johnson Space Center, Houston, Texas.
297. D. E. Moriarty, Alan C. Schultz, and John J. Grefenstette. Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, 11:199–229, 1999. http://www.ib3.gmu.edu/gref/papers/moriarty-jair99.html.
298. Rémi Munos and Jocelyn Patinel. Reinforcement learning with dynamic covering of state-action space: Partitioning Q-learning. In Cliff et al. [86], pages 354–363.
299. Jorge Muruzábal. Mining the space of generality with uncertainty-concerned cooperative classifiers. In Banzhaf et al. [12], pages 449–457.
300. Ichiro Nagasaka and Toshiharu Taura. 3D Geometic Representation for Shape Generation using Classifier System. In Koza et al. [263], pages 515–520.
301. Filippo Neri. *First Order Logic Concept Learning by means of a Distributed Genetic Algorithm.* PhD thesis, University of Milano, Italy, 1997.

302. Filippo Neri and Attilio Giordana. A distributed genetic algorithm for concept learning. In Eshelman [148], pages 436–443.
303. Filippo Neri and L. Saitta. Exploring the power of genetic search in learning symbolic classifiers. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-18:1135–1142, 1996.
304. Volker Nissen and Jörg Biethahn. Determining a Good Inventory Policy with a Genetic Algorithm. In Jörg Biethahn and Volker Nissen, editors, *Evolutionary Algorithms in Management Applications*, pages 240–249. Springer Verlag, 1995.
305. Jae Chan Oh. Improved Classifier System Using Genetic Algorithms. Master's thesis, Wright State University, ??
306. Norihiko Ono and Adel T. Rahmani. Self-Organization of Communication in Distributed Learning Classifier Systems. In Albrecht et al. [5], pages 361–367.
307. F. Oppacher and D. Deugo. The Evolution of Hierarchical Representations. In *Proceedings of the 3rd European Conference on Artificial Life*. Springer-Verlag, 1995.
308. Alexandre Parodi and P. Bonelli. The Animat and the Physician. In Meyer and Wilson [291], pages 50–57.
309. Alexandre Parodi and Pierre Bonelli. A New Approach to Fuzzy Classifier Systems. In Forrest [173], pages 223–230.
310. Mukesh J. Patel, Marco Colombetti, and Marco Dorigo. Evolutionary Learning for Intelligent Automation: A Case Study. *Intelligent Automation and Soft Computing*, 1(1):29–42, 1995.
311. Mukesh J. Patel and Marco Dorigo. Adaptive Learning of a Robot Arm. In Fogarty [160], pages 180–194.
312. Mukesh J. Patel and U. Schnepf. Concept Formation as Emergent Phenomena. In Francisco J. Varela and P. Bourgine, editors, *Proceedings First European Conference on Artificial Life*, pages 11–20. MIT Press, 1992.
313. Rolf Pfeifer, Bruce Blumberg, Jean-Arcady Meyer, and Stewart W. Wilson, editors. *From Animals to Animats 5. Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior (SAB98)*. A Bradford Book. MIT Press, 1998.
314. Steven E. Phelan. *Using Artificial Adaptive Agents to Explore Strategic Landscapes*. PhD thesis, School of Business, Faculty of Law and Management, La Trobe University, Australia, 1997.
315. A. G. Pipe and Brian Carse. A Comparison between two Architectures for Searching and Learning in Maze Problems. In Fogarty [160], pages 238–249.
316. R. Piroddi and R. Rusconi. A Parallel Classifier System to Solve Learning Problems. Master's thesis, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy., 1992.
317. C. L. Ramsey and John J. Grefenstette. Case-based anytime learning. In Case-Based Reasoning: Papers from the 1994 Workshop, D. W. Aha (Ed.). Technical Report WS-94-07, AAAI Press: Menlo Park, CA, 1994. http://www.ib3.gmu.edu/gref/.
318. Gregory J. E. Rawlins, editor. *Proceedings of the First Workshop on Foundations of Genetic Algorithms (FOGA91)*. Morgan Kaufmann: San Mateo, 1991.
319. Robert A. Richards. *Zeroth-Order Shape Optimization Utilizing a Learning Classifier System*. PhD thesis, Stanford University, 1995. Online version available at: http://www-leland.stanford.edu/~buc/SPHINcsX/book.html.
320. Robert A. Richards. Classifier System Metrics: Graphical Depictions. In Koza et al. [262], pages 652–657.

321. Robert A. Richards and Sheri D. Sheppard. Classifier System Based Structural Component Shape Improvement Utilizing I-DEAS. In *Iccon User's Conference Proceeding*. Iccon, 1992.
322. Robert A. Richards and Sheri D. Sheppard. Learning Classifier Systems in Design Optimization. In *Design Theory and Methodology '92*. The American Society of Mechanical Engineers, 1992.
323. Robert A. Richards and Sheri D. Sheppard. Two-dimensional Component Shape Improvement via Classifier System. In *Artificial Intelligence in Design '92*. Kluwer Academic Publishers, 1992.
324. Robert A. Richards and Sheri D. Sheppard. A Learning Classifier System for Three-dimensional Shape Optimization. In H. M. Voigt, W. Ebeling, I. Rechenberg, and H. P. Schwefel, editors, *Parallel Problem Solving from Nature – PPSN IV*, volume 1141 of *LNCS*, pages 1032–1042. Springer-Verlag, 1996.
325. Robert A. Richards and Sheri D. Sheppard. Three-Dimensional Shape Optimization Utilizing a Learning Classifier System. In Koza et al. [264], pages 539–546.
326. Rick L. Riolo. Bucket Brigade Performance: I. Long Sequences of Classifiers. In Grefenstette [202], pages 184–195.
327. Rick L. Riolo. Bucket Brigade Performance: II. Default Hierarchies. In Grefenstette [202], pages 196–201.
328. Rick L. Riolo. CFS-C: A Package of Domain-Independent Subroutines for Implementing Classifier Systems in Arbitrary User-Defined Environments. Technical report, University of Michigan, 1988.
329. Rick L. Riolo. *Empirical Studies of Default Hierarchies and Sequences of Rules in Learning Classifier Systems*. PhD thesis, University of Michigan, 1988.
330. Rick L. Riolo. The Emergence of Coupled Sequences of Classifiers. In Schaffer [354], pages 256–264.
331. Rick L. Riolo. The Emergence of Default Hierarchies in Learning Classifier Systems. In Schaffer [354], pages 322–327.
332. Rick L. Riolo. Lookahead Planning and Latent Learning in a Classifier System. In Meyer and Wilson [291], pages 316–326.
333. Rick L. Riolo. Modeling Simple Human Category Learning with a Classifier System. In Booker and Belew [44], pages 324–333.
334. Rick L. Riolo. The discovery and use of forward models for adaptive classifier systems. In *Collected Abstracts for the First International Workshop on Learning Classifier System (IWLCS-92)* [2]. October 6–8, NASA Johnson Space Center, Houston, Texas.
335. Joaquin Rivera and Roberto Santana. Improving the Discovery Component of Classifier Systems by the Application of Estimation of Distribution Algorithms. In *Proceedings of Student Sessions ACAI'99: Machine Learning and Applications*, pages 43–44, Chania, Greece, July 1999.
336. Gary Roberts. A Rational Reconstruction of Wilson's Animat and Holland's CS-1. In Schaffer [354].
337. Gary Roberts. Dynamic Planning for Classifier Systems. In Forrest [173], pages 231–237.
338. George G. Robertson. Parallel Implementation of Genetic Algorithms in a Classifier System. In Grefenstette [202], pages 140–147. Also Tech. Report TR-159 RL87-5 Thinking Machines Corporation.
339. George G. Robertson. Population Size in Classifier Systems. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 142–152. Morgan Kaufmann, 1988.

340. George G. Robertson. Parallel Implementation of Genetic Algorithms in a Classifier System. In Davis [104], pages 129–140.
341. George G. Robertson and Rick L. Riolo. A Tale of Two Classifier Systems. *Machine Learning*, 3:139–159, 1988.
342. J. A. Meyer H. L. Roitblat and S. W. Wilson, editors. *From Animals to Animats 2. Proceedings of the Second International Conference on Simulation of Adaptive Behavior (SAB92)*. A Bradford Book. MIT Press, 1992.
343. S. Ross. Accurate Reaction or Reflective Action? Master's thesis, School of Cognitive and Computing Sciences, University of Sussex, 1994.
344. A. Sanchis, J. M. Molina, P. Isasi, and J. Segovia. Knowledge acquisition including tags in a classifier system. In Angeline et al. [6], pages 137–144.
345. Adrian V. Sannier and Erik D. Goodman. Midgard: A Genetic Approach to Adaptive Load Balancing for Distributed Systems. In *Proc. Fifth Intern. Conf. Machine Learning*. Morgan Kaufmann, 1988.
346. Cédric Sanza, Christophe Destruel, and Yves Duthen. Agents autonomes pour l'interaction adaptative dans les mondes virtuels. In *5ème Journées de l'Association Francaise d'Informatique Graphique. Décembre 1997, Rennes, France*, 1997. In French.
347. Cédric Sanza, Christophe Destruel, and Yves Duthen. A learning method for adaptation and evolution in virtual environments. In *3rd International Conference on Computer Graphics and Artificial Intelligence, April 1998, Limoges, France*, 1998.
348. Cédric Sanza, Christophe Destruel, and Yves Duthen. Autonomous actors in an interactive real-time environment. In *ICVC'99 International Conference on Visual Computing Feb. 1999, Goa, India*, 1999.
349. Cédric Sanza, Christophe Destruel, and Yves Duthen. Learning in real-time environment based on classifiers system. In *7th International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media'99*, Plzen, Czech Republic, 1999.
350. Cédric Sanza, Cyril Panatier, Hervé Luga, and Yves Duthen. Adaptive Behavior for Cooperation: a Virtual Reality Application. In *8th IEEE International Workshop on Robot and Human Interaction September 1999, Pisa, Italy*, 1999.
351. Andreas Schachtner. A classifier system with integrated genetic operators. In H.-P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature*, volume 496 of *Lecture Notes in Computer Science*, pages 331–337, Berlin, 1990. Springer.
352. J. David Schaffer. *Some experiments in machine learning using vector evaluated genetic algorithms*. PhD thesis, Vanderbilt University, Nashville, 1984.
353. J. David Schaffer. Learning Multiclass Pattern Discrimination. In Grefenstette [200], pages 74–79.
354. J. David Schaffer, editor. *Proceedings of the 3rd International Conference on Genetic Algorithms (ICGA89)*, George Mason University, June 1989. Morgan Kaufmann.
355. Alan C. Schultz and John J. Grefenstette. Evolving Robot Behaviors. Poster at the Artificial Life Conference. http://www.ib3.gmu.edu/gref/.
356. Alan C. Schultz and John J. Grefenstette. Improving Tactical Plans with Genetic Algorithms. In *Proceedings of the Second International Conference on Tools for Artificial Intelligence*. IEEE, 1990.
357. Dale Schuurmans and Jonathan Schaeffer. Representational Difficulties with Classifier Systems. In Schaffer [354], pages 328–333. http://www.cs.ualberta.ca/~jonathan/Papers/classifier.ps.gz.

358. Hans-Paul Schwefel and Reinhard Männer, editors. *Parallel Problem Solving from Nature: Proceedings of the First International Workshop. Dortmund, FRG, 1–3 Oct 1990*, number 496 in Lecture Notes in Computer Science, Heidelberg, 1990. Springer.
359. Tod A. Sedbrook, Haviland Wright, and Richard Wright. Application of a Genetic Classifier for Patient Triage. In Booker and Belew [44], pages 334–338.
360. Sandip Sen. Classifier system learning of multiplexer function. Dept. of Electrical Engineering, University of Alabama, Tuscaloosa, Alabama. Class Project, 1988.
361. Sandip Sen. Sequential Boolean Function Learning by Classifier System. In *Proc. of 1st International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, 1988.
362. Sandip Sen. Noise Sensitivity in a simple classifier system. In *Proc. 5th Conf. on Neural Networks & Parallel Distributed Processing*, 1992.
363. Sandip Sen. Improving classification accuracy through performance history. In Forrest [173], pages 652–652.
364. Sandip Sen. A Tale of two representations. In *Proc. 7th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pages 245–254, 1994.
365. Sandip Sen. Modeling human categorization by a simple classifier system. WSC1: 1st Online Workshop on Soft Computing. Aug 19-30, 1996. http://www.bioele.nuee.nagoya-u.ac.jp/wsc1/papers/p020.html, 1996.
366. Sandip Sen and Mahendra Sekaran. Multiagent Coordination with Learning Classifier Systems. In Gerhard Weiß and Sandip Sen, editors, *Proceedings of the IJCAI Workshop on Adaption and Learning in Multi-Agent Systems*, volume 1042 of *LNAI*, pages 218–233. Springer Verlag, 1996.
367. F. Seredynski and C. Z. Janikow. Learning nash equilibria by coevolving distributed classifier systems. In Angeline et al. [6], pages 1619–1626.
368. Sotaro Shimada and Yuichiro Anzai. Component-Based Adaptive Architecture with Classifier Systems. In Pfeifer et al. [313].
369. Lingyan Shu and Jonathan Schaeffer. VCS: Variable Classifier System. In Schaffer [354], pages 334–339. http://www.cs.ualberta.ca/~jonathan/Papers/vcs.ps.gz.
370. Lingyan Shu and Jonathan Schaeffer. Improving the Performance of Genetic Algorithm Learning by Choosing a Good Initial Population. Technical Report TR-90-22, University of Alberta, CS DEPT, Edmonton, Alberta, Canada, 1990.
371. Lingyan Shu and Jonathan Schaeffer. HCS: Adding Hierarchies to Classifier Systems. In Booker and Belew [44], pages 339–345.
372. George D. Smith, Nigel C. Steele, and Rudolf F. Albrecht, editors. *Artificial Neural Networks and Genetic Algorithms*. Springer, 1997.
373. Robert E. Smith. *Defult Hierarchy Formation and Memory Exploitation in Learning Classifier Systems*. PhD thesis, University of Alabama, 1991.
374. Robert E. Smith. A Report on The First International Workshop on Learning Classifier Systems (IWLCS-92). NASA Johnson Space Center, Houston, Texas, Oct. 6-9. ftp://lumpi.informatik.uni-dortmund.de/pub/LCS/papers/lcs92.ps.gz or from ENCORE, The Electronic Appendix to the Hitch-Hiker's Guide to Evolutionary Computation (ftp://ftp.krl.caltech.edu/pub/EC/Welcome.html) in the section on Classifier Systems, 1992.
375. Robert E. Smith. Is a classifier system a type of neural network? In *Collected Abstracts for the First International Workshop on Learning Classifier System (IWLCS-92)* [2]. October 6–8, NASA Johnson Space Center, Houston, Texas.

376. Robert E. Smith. Memory exploitation in learning classifier systems. In *Collected Abstracts for the First International Workshop on Learning Classifier System (IWLCS-92)* [2]. October 6–8, NASA Johnson Space Center, Houston, Texas.
377. Robert E. Smith. Genetic Learning in Rule-Based and Neural Systems. In *Proceedings of the Third International Workshop on Neural Networks and Fuzzy Logic*, volume 1, page 183. NASA. Johnson Space Center, January 1993.
378. Robert E. Smith. Memory Exploitation in Learning Classifier Systems. *Evolutionary Computation*, 2(3):199–220, 1994.
379. Robert E. Smith and H. Brown Cribbs. Is a Learning Classifier System a Type of Neural Network? *Evolutionary Computation*, 2(1):19–36, 1994.
380. Robert E. Smith, B. A. Dike, R. K. Mehra, B. Ravichandran, and A. El-Fallah. Classifier Systems in Combat: Two-sided Learning of Maneuvers for Advanced Fighter Aircraft. In *Computer Methods in Applied Mechanics and Engineering*. Elsevier, 1999.
381. Robert E. Smith, Stephanie Forrest, and A. S. Perelson. Searching for diverse, cooperative subpopulations with Genetic Algorithms. *Evolutionary Computation*, 1(2):127–149, 1993.
382. Robert E. Smith and David E. Goldberg. Reinforcement Learning with Classifier Systems: Adaptive Default Hierarchy Formation. Technical Report 90002, TCGA, University of Alabama, 1990.
383. Robert E. Smith and David E. Goldberg. Variable Default Hierarchy Separation in a Classifier System. In Rawlins [318], pages 148–170.
384. Robert E. Smith and David E. Goldberg. Reinforcement learning with classifier systems : adaptative default hierarchy formation. *Applied Artificial Intelligence*, 6, 1992.
385. Robert E. Smith and H. B. Cribbs III. Cooperative Versus Competitive System Elements in Coevolutionary Systems. In Maes et al. [284], pages 497–505.
386. Robert E. Smith and H. B. Cribbs III. Combined biological paradigms. *Robotics and Autonomous Systems*, 22(1):65–74, 1997.
387. Robert E. Smith and Manuel Valenzuela-Rendón. A Study of Rule Set Development in a Learning Classifier System. In Schaffer [354], pages 340–346.
388. S. F. Smith. *A Learning System Based on Genetic Adaptive Algorithms*. PhD thesis, University of Pittsburgh, 1980.
389. S. F. Smith. Flexible Learning of Problem Solving Heuristics through Adaptive Search. In *Proceedings Eigth International Joint Conference on Artificial Intelligence*, pages 422–425, 1983.
390. S. F. Smith and D. P. Greene. Cooperative Diversity using Coverage as a Constraint. In *Collected Abstracts for the First International Workshop on Learning Classifier System (IWLCS-92)* [2]. October 6–8, NASA Johnson Space Center, Houston, Texas.
391. Sotaro Shimada and Yuichiro Anzai. Fast and Robust Convergence of Chained Classifiers by Generating Operons through Niche Formation. In Banzhaf et al. [12], page 810. One page poster paper.
392. Piet Spiessens. PCS: A Classifier System that Builds a Predictive Internal World Model. In *PROC of the 9th European Conference on Artificial Intelligence, Stockholm, Sweden, Aug. 6–10*, pages 622–627, 1990.
393. Wolfgang Stolzmann. Learning Classifier Systems using the Cognitive Mechanism of Anticipatory Behavioral Control, detailed version. In *Proceedings of the First European Workshop on Cognitive Modelling*, pages 82–89. Berlin: TU, 1996. http://www.psychologie.uni-wuerzburg.de/stolzmann/.

394. Wolfgang Stolzmann. *Antizipative Classifier Systeme*. PhD thesis, Fachbereich Mathematik/Informatik, University of Osnabrueck, 1997.
395. Wolfgang Stolzmann. Two Applications of Anticipatory Classifier Systems (ACSs). In *Proceedings of the 2nd European Conference on Cognitive Science*, pages 68–73. Manchester, U.K., 1997. http://www.psychologie.uni-wuerzburg.de/stolzmann/.
396. Wolfgang Stolzmann. Anticipatory classifier systems. In *Proceedings of the Third Annual Genetic Programming Conference*, pages 658–664, San Francisco, CA, 1998. Morgan Kaufmann. http://www.psychologie.uni-wuerzburg.de/stolzmann/gp-98.ps.gz.
397. S. Tokinaga and A. B. Whinston. Applying Adaptive Credit Assignment Algorithm for the Learning Classifier System Based upon the Genetic Algorithm. *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences*, VE75A(5):568–577, May 1992.
398. Andy Tomlinson and Larry Bull. A Corporate Classifier System. In A. E. Eiben, T. Bäck, M. Shoenauer, and H.-P Schwefel, editors, *Proceedings of the Fifth International Conference on Parallel Problem Solving From Nature – PPSN V*, number 1498 in LNCS, pages 550–559. Springer Verlag, 1998.
399. Andy Tomlinson and Larry Bull. On Corporate Classifier Systems: Increasing the Benefits of Rule Linkage. In Banzhaf et al. [12], pages 649–656.
400. Andy Tomlinson and Larry Bull. A zeroth level corporate classifier system. In *Proc. 2nd International Workshop of Learning Classifier Systems (IWLCS-99)*, pages 306–313, Cambridge, MA, 1999. AAAI Press.
401. Patrick Tufts. Dynamic Classifiers: Genetic Programming and Classifier Systems. In E. V. Siegel and J. R. Koza, editors, *Working Notes for the AAAI Symposium on Genetic Programming*, pages 114–119, MIT, Cambridge, MA, USA, 1995. AAAI. http://www.cs.brandeis.edu/~zippy/papers.html.
402. Kirk Twardowski. Implementation of a Genetic Algorithm based Associative Classifier System (ACS). In *Proceedings International Conference on Tools for Artificial Intelligence*, 1990.
403. Kirk Twardowski. Credit Assignment for Pole Balancing with Learning Classifier Systems. In Forrest [173], pages 238–245.
404. Kirk Twardowski. An Associative Architecture for Genetic Algorithm-Based Machine Learning. *Computer*, 27(11):27–38, November 1994.
405. J. Urzelai, Dario Floreano, Marco Dorigo, and Marco Colombetti. Incremental Robot Shaping. *Connection Science*, 10(3–4):341–360, 1998.
406. J. Urzelai, Dario Floreano, Marco Dorigo, and Marco Colombetti. Incremental Robot Shaping. In Koza et al. [262], pages 832–840.
407. Manuel Valenzuela-Rendón. Boolean Analysis of Classifier Sets. In Schaffer [354], pages 351–358.
408. Manuel Valenzuela-Rendón. *Two analysis tools to describe the operation of classifier systems*. PhD thesis, University of Alabama, 1989. Also TCGA tech. report 89005.
409. Manuel Valenzuela-Rendón. The Fuzzy Classifier System: a Classifier System for Continuously Varying Variables. In Booker and Belew [44], pages 346–353.
410. Manuel Valenzuela-Rendón. The Fuzzy Classifier System: Motivations and First Results. *Lecture Notes in Computer Science*, 496:338–??, 1991.
411. Manuel Valenzuela-Rendón. Reinforcement learning in the fuzzy classifier system. In *Collected Abstracts for the First International Workshop on Learning Classifier System (IWLCS-92)* [2]. October 6–8, NASA Johnson Space Center, Houston, Texas.

412. Manuel Valenzuela-Rendón and Eduardo Uresti-Charre. A Non-Genetic Algorithm for Multiobjective Optimization. In Bäck [8], pages 658–665.
413. Terry van Belle. A New Approach to Genetic-Based Automatic Feature Discovery. Master's thesis, University of Alberta, 1995. http://www.cs.ualberta.ca/~jonathan/.
414. Gilles Venturini. *Apprentissage Adaptatif et Apprentissage Supervisé par Algorithme Génétique.* PhD thesis, Université de Paris-Sud., 1994.
415. Nickolas Vriend. Self-Organization of Markets: An Example of a Computational Approach. *Computational Economics*, 8(3):205–231, 1995.
416. L. A. Wang. Classifier System Learning of the Boolean Multiplexer Function. Master's thesis, Computer Science Department, University of Tennessee, Knoxville, TN, 1990.
417. Gerhard Weiss. The Action-Oriented Bucket Brigade. Technical Report FKI-156-91, Technical Univ. München (TUM), 1991.
418. Thomas H. Westerdale. The bucket brigade is not genetic. In Grefenstette [200], pages 45–59.
419. Thomas H. Westerdale. Altruism in the bucket brigade. In Grefenstette [202], pages 22–26.
420. Thomas H. Westerdale. A Defense of the Bucket Brigade. In Schaffer [354], pages 282–290.
421. Thomas H. Westerdale. Quasimorphisms or Queasymorphisms? Modeling Finite Automaton Environments. In Rawlins [318], pages 128–147.
422. Thomas H. Westerdale. Redundant Classifiers and Prokaryote Genomes. In Booker and Belew [44], pages 354–360.
423. Thomas H. Westerdale. Classifier Systems - No Wonder They Don't Work. In Koza et al. [263], pages 529–537.
424. Thomas H. Westerdale. An Approach to Credit Assignment in Classifier Systems. *Complexity*, 4(2), 1999.
425. Thomas H. Westerdale. Wilson's Error Measurement and the Markov Property – Identifying Detrimental Classifiers. In *Proc. 2nd International Workshop of Learning Classifier Systems (IWLCS-99)*, Cambridge, MA, 1999. AAAI Press.
426. Jason R. Wilcox. Organizational Learning within a Learning Classifier Systems. Master's thesis, University of Illinois, 1995. Also Tech. Report No. 95003 IlliGAL.
427. Stewart W. Wilson. Aubert processing and intelligent vision. Technical report, Polaroid Corporation, 1981.
428. Stewart W. Wilson. On the retino-cortical mapping. *International Journal of Man-Machine Studies*, 18:361–389, 1983.
429. Stewart W. Wilson. Adaptive "cortical" pattern recognition. In Grefenstette [200], pages 188–196.
430. Stewart W. Wilson. Knowledge Growth in an Artificial Animal. In Grefenstette [200], pages 16–23. Also appeared in Proceedings of the 4th Yale.
431. Stewart W. Wilson. Knowledge Growth in an Artificial Animal. In *Proceedings of the 4th Yale Workshop on Applications of Adaptive Systems Theory*, pages 98–104, 1985.
432. Stewart W. Wilson. Classifier System Learning of a Boolean Function. Technical Report RIS 27r, The Rowland Institute for Science, 1986.
433. Stewart W. Wilson. Knowledge Growth in an Artificial Animal. In K. S. Narenda, editor, *Adaptive and learning systems: Theory and applications*, pages 255–264. Plenum Press: New York, 1986.

434. Stewart W. Wilson. Classifier Systems and the Animat Problem. *Machine Learning*, 2:199–228, 1987. Also Research Memo RIS-36r, the Rowland Institute for Science, Cambridge, MA, 1986.
435. Stewart W. Wilson. Hierarchical Credit Allocation in a Classifier System. In *Proceedings Tenth International Joint Conference on AI (IJCAI-87)*, pages 217–220. Morgan Kaufmann Publishers, 1987. Also Research Memo RIS-37r, the Rowland Insitute for Science, Cambridge, MA, 1986.
436. Stewart W. Wilson. Quasi-Darwinian Learning in a Classifier System. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 59–65. Morgan Kaufmann, 1987.
437. Stewart W. Wilson. The genetic algorithm and biological development. In Grefenstette [202], pages 247–251.
438. Stewart W. Wilson. Bid Competition and Specificity Reconsidered. *Complex Systems*, 2(6):705–723, 1988.
439. Stewart W. Wilson. Hierarchical Credit Assignment in a Classifier System. In M. Elzas, T. Oren, and B. P. Zeigler, editors, *Modelling and Simulation Methodology: Knowledge Systems Paradigms*. North Holland, 1988.
440. Stewart W. Wilson. Hierarchical Credit Allocation in a Classifier System. In Davis [104], pages 104–115.
441. Stewart W. Wilson. Hierarchical credit allocation in a classifier system. In M. S. Elzas, T. I. Oren, and B. P. Zeigler, editors, *Modelling and simulation methodology*, pages 351–357. North-Holland: New York, 1989.
442. Stewart W. Wilson. Perceptron redux: Emergence of structure. In *Special issue of Physica D (Vol. 42)* [1], pages 249–256. Republished in Emergent Computation, S. Forrest (ed.), MIT Press/Bradford Books.
443. Stewart W. Wilson. The Genetic Algorithm and Simulated Evolution. In Chris Langton, editor, *Artificial Life: Proceedings of an Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems*, volume VI of *Santa Fe Institute Studies in the Sciences of Complexity*. Addison-Wesley: Reading, MA, 1989.
444. Stewart W. Wilson. The Animat Path to AI. In Meyer and Wilson [291], pages 15–21. http://prediction-dynamics.com/.
445. Stewart W. Wilson. Classifier System mapping of real vectors. In *Collected Abstracts for the First International Workshop on Learning Classifier System (IWLCS-92)* [2]. October 6–8, NASA Johnson Space Center, Houston, Texas.
446. Stewart W. Wilson. Toward a GA solution of the discovery problem. In *Collected Abstracts for the First International Workshop on Learning Classifier System (IWLCS-92)* [2]. October 6–8, NASA Johnson Space Center, Houston, Texas.
447. Stewart W. Wilson. ZCS: A zeroth level classifier system. *Evolutionary Computation*, 2(1):1–18, 1994. http://prediction-dynamics.com/.
448. Stewart W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995. http://prediction-dynamics.com/.
449. Stewart W. Wilson. Explore/exploit strategies in autonomy. In Maes et al. [284], pages 325–332.
450. Stewart W. Wilson. Generalization in XCS. Unpublished contribution to the ICML '96 Workshop on Evolutionary Computing and Machine Learning. http://prediction-dynamics.com/, 1996.
451. Stewart W. Wilson. Generalization in evolutionary learning. Presented at the Fourth European Conference on Artificial Life (ECAL97), Brighton, UK, July 27-31. http://prediction-dynamics.com/, 1997.
452. Stewart W. Wilson. Generalization in the XCS classifier system. In Koza et al. [262], pages 665–674. http://prediction-dynamics.com/.

453. Stewart W. Wilson. Get real! XCS with continuous-valued inputs. In L. Booker, Stephanie Forrest, M. Mitchell, and Rick L. Riolo, editors, *Festschrift in Honor of John H. Holland*, pages 111–121. Center for the Study of Complex Systems, 1999. http://prediction-dynamics.com/.
454. Stewart W. Wilson. State of XCS classifier system research. Technical Report 99.1.1, Prediction Dynamics, Concord MA, 1999. http://prediction-dynamics.com/.
455. Stewart W. Wilson and David E. Goldberg. A Critical Review of Classifier Systems. In Schaffer [354], pages 244–255. http://prediction-dynamics.com/.
456. Ian Wright. Reinforcement Learning and Animat Emotions. In Maes et al. [284], pages 272–281.
457. Ian Wright. Reinforcement learning and animat emotions. Technical Report CSRP-96-4, School of Computer Science. University of Birmingham, 1996. ftp://ftp.cs.bham.ac.uk/pub/tech-reports/1996/CSRP-96-04.ps.gz.
458. Derek F. Yates and Andrew Fairley. An Investigation into Possible Causes of, and Solutions to, Rule Strength Distortion Due to the Bucket Brigade Algorithm. In Forrest [173], pages 246–253.
459. Derek F. Yates and Andrew Fairley. Evolutionary Stability in Simple Classifier Systems. In Fogarty [160], pages 28–37.
460. Takahiro Yoshimi and Toshiharu Taura. Hierarchical Classifier System Based on the Concept of Viewpoint. In Koza et al. [262], pages 675–678.
461. Takahiro Yoshimi and Toshiharu Taura. A Computational Model of a Viewpoint-Forming Process in a Hierarchical Classifier System. In Banzhaf et al. [12], pages 758–766.
462. Zhaohua Zhang, Stan Franklin, and Dipankar Dasgupta. Metacognition in Software Agents Using Classifier Systems. In *AAAI-98. Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 83–88, Madison (WI), 1998. AAAI-Press and MIT Press.
463. Hayong Harry Zhou. Classifier systems with long term memory. In Grefenstette [200], pages 178–182.
464. Hayong Harry Zhou. *CSM: A genetic classifier system with memory for learning by analogy.* PhD thesis, Department of Computer Science, Vanderbilt University, Nashville, TN, 1987.
465. Hayong Harry Zhou. CSM: A Computational Model of Cumulative Learning. *Machine Learning*, 5(4):383–406, 1990.
466. Hayong Harry Zhou and John J. Grefenstette. Learning by Analogy in Genetic Classifier Systems. In Schaffer [354], pages 291–297.
467. Raed Abu Zitar and Mohammad H. Hassoun. Regulator Control via Genetic Search Assisted Reinforcement. In Forrest [173], pages 254–263.

Author Index