Rule-based Evolutionary Online Learning Systems: Learning Bounds, Classification, and Prediction

Martin V. Butz

IlliGAL Report No. 2004034
2004

Illinois Genetic Algorithms Laboratory (IlliGAL)
Department of General Engineering
University of Illinois at Urbana-Champaign
117 Transportation Building
104 S. Mathews Avenue, Urbana, IL 61801
http://www-illigal.ge.uiuc.edu

RULE-BASED EVOLUTIONARY ONLINE LEARNING SYSTEMS: LEARNING BOUNDS, CLASSIFICATION, AND PREDICTION

BY

MARTIN VOLKER BUTZ

Dipl.-Inf., Bayerische Julius-Maximilians Universität Würzburg, 2001

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2004

Urbana, Illinois

Abstract

Rule-based evolutionary online learning systems, often referred to as *Michigan-style learning classifier systems* (LCSs), were proposed nearly thirty years ago (Holland, 1976; Holland, 1977) originally calling them *cognitive systems*. LCSs combine the strength of reinforcement learning with the generalization capabilities of genetic algorithms promising a flexible, online generalizing, solely reinforcement dependent learning system. However, despite several initial successful applications of LCSs and their interesting relations with animal learning and cognition, understanding of the systems remained somewhat obscured. Questions concerning learning complexity or convergence remained unanswered. Performance in different problem types, problem structures, concept spaces, and hypothesis spaces stayed nearly unpredictable. This thesis has the following three major objectives: (1) to establish a facetwise theory approach for LCSs that promotes system analysis, understanding, and design; (2) to analyze, evaluate, and enhance the XCS classifier system (Wilson, 1995) by the means of the facetwise approach establishing a fundamental XCS learning theory; (3) to identify both the major advantages of an LCS-based learning approach as well as the most promising potential application areas. Achieving these three objectives leads to a rigorous understanding of LCS functioning that enables the successful application of LCSs to diverse problem types and problem domains. The quantitative analysis of XCS shows that the interactive, evolutionary-based online learning mechanism works machine learning competitively yielding a low-order polynomial learning complexity. Moreover, the facetwise analysis approach facilitates the successful design of more advanced LCSs including Holland's originally envisioned cognitive systems.

Acknowledgments

I am grateful to many people for supporting me not only intellectually but also mentally and socially in my work and life besides work. These acknowledgments can only give a glimpse on how much I benefited and learned from all my friends, family, and colleagues. Thank you so much to all of you.

I am in debt to my adviser David E. Goldberg, who supplied me with invaluable advise and guidance throughout my time at UIUC concerning my research, writing, organization and life, and who supported me in all my plans and encouraged me to pursue thoughts and ideas that may have partially sounded very unconventional at first. Thank you so much for trusting and believing in me.

I am also very grateful to my other thesis committee members including Gerald DeJong, Sylvian Ray, and Dan Roth. Thank you for all the useful comments, suggestions, and additional work. I am also grateful to many other faculty members of UIUC including Gul Agha, Thomas Anastasio, Kathryn Bock, Gary Dell, Steven LaValle, Edward Reingold, and many others. Thank you also for the support from the automated learning group at the national center for supercomputing applications (NCSA) and Michael Welge and Loretta Auvil, in particular.

In the mean time, I am very grateful to Stewart W. Wilson, who was always available for additional advise, support, and help in my writing. My visit to Boston in April 2000 was more than inspiring and certainly shaped a large part of the basis of this thesis and the thoughts beyond it including the first complexity derivations for XCS.

I am also very grateful for all the support I received during my studies from Joachim Hoffmann, head of the department of cognitive psychology at the Universität Würzburg. His cognitive perspective of things is always stimulating and many of the parts of this thesis related to cognition are shaped by the numerous discussions we had during my time in Germany.

I am also grateful to Wolfgang Stolzmann, who made my involvement into learning classifier systems and anticipatory systems possible in the first place and encouraged me to

visit UIUC. Thank you for all the trust and encouragement.

I am also grateful to Pier Luca Lanzi, who grew to be more than a research partner and who showed me many aspects of XCS that I did not imagine in that way before—in particular in the reinforcement learning and function approximation side of things. His visit to UIUC was inspiring in many ways resulting in an additional research boost as well as an even closer lab community.

At the Illinois genetic algorithms laboratory (IlliGAL), I was blessed with a research environment of really smart people that were always available for advise and discussions but also knew when to talk about other issues in life. Particularly, I am very grateful to Kumara Sastry, student lab director and the best person to ask and discuss any problem that arises. I am similarly grateful to Xavier Llorà, who was a similar strong support, discussion partner, and inspiration besides the many other activities we shared. On my way to the PhD also Martin Pelikan was always an inspiration and a great help showing me new understandings and perspectives on many research issues and other concerns—unfortunately we could not meet as often anymore after Martin left IlliGAL two years ago. Thank you so much to all other current and former lab partners including Chang-Wook Ahn, Chen-Ju Chao, Ying-Ping Chen, Dimitri Knjazew, Fiepeng Li, Fernando Lobo, Jonathan Loveall, Naohiro Matsumura, Abhishek Sinha, Kurian Tharakunnel, Tian-Li Yu and the lab visitors Hussein Abbass, Jaume Bacardit, Clarissa Van Hoyweghen, Pier Luca Lanzi, Claudio Lima, Kei Onishi, Gerulf Pedersen, and Franz Rothlauf. The close cooperations with Hussein Abbass, Jaume Bacardit and Pier Luca Lanzi were particularly fruitful. Thank you all for being so good friends and taking care of the lab and other things together.

The thesis significantly improved due to comments from many people including David Goldberg and the whole thesis committee, Martin Pelikan, Kumara Sastry, Samarth Swarup, and Stewart Wilson. Thank you for the help proofreading this thesis.

Thank you to all my other friends and research mates at UIUC and beyond including Jason Bant, Jacob Borgerson, Amanda Hinkle, John Horstman, Alex Kosorukov, Kiran Lakkaraju, Samarth Swarup, Dav Zimak and many others. I was also blessed with and greatly supported by my girlfriend Marjorie Kinney, who always listens to my thoughts and is an inspiration in many ways—thank you!

I am also grateful to my colleagues and friends back in Germany. The team at the department of cognitive psychology was a great help and made me feel always welcome and happy when I came back during the summers. Thank you in particular to Andrea Kiesel. Andrea was always helpful and more than simply supportive she made me feel home—thank you! Thank you for all the cooperation and help from all the others in the department

including Andrea Albert, Oliver Herbort, Silvia Jahn, Wilfried Kunde, Alexandra Lenhard, Georg Schüssler, Albrecht Sebald, Armin Stock, and Annika Wagener. I am looking forward to the next few years!

Thank you to all my other friends back home that were always available and always welcoming me including Marianne Greubel, Heiko and Frank Hofmann, Gregor Kopka, Stefan Merz, Rainer Munzert, André Ponsong, Matthias Reiher, Franz Rothlauf, Peter Scheerer, Marion Schmidt, Daniel Schneider and Johannes Schunk. I don't know what I would do without you.

I am also very grateful to my whole family back in Germany. My brother Christoph is an inspiration on his own. My parents, Susanne and Teja, helped me keep the balance between the US and Germany and supported me in various ways. I am glad we grew so close together again despite these long distances for such long times.

Finally, I would like to acknowledge all the financial support I received during my studies including money from the German research foundation (Deutsche Forschungsgemeinschaft, DFG), the National Center for Supercomputing Applications (NCSA) at UIUC, as well as the CSE fellowship I received from the Computational Science and Engineering department at UIUC. Thank you also for the additional money from other resources available to the IlliGAL lab that supplied me with computer power and an optimal working environment. I will miss this place.

Table of Contents

List of Tables . . . . . **xii**

List of Figures . . . . . **xiii**

Introduction **1**
Thesis Objectives 4
Road Map 4

Chapter 1 Prerequisites **7**
1.1 Problem Types 8
1.1.1 Optimization Problems 8
1.1.2 Classification Problems 10
1.1.3 Reinforcement Learning Problems 12
1.2 Reinforcement Learning 15
1.2.1 Model-Free Reinforcement Learning 16
1.2.2 Model-Based Reinforcement Learning 18
1.3 Genetic Algorithms 19
1.3.1 Basic Genetic Algorithm 19
1.3.2 Facetwise GA Theory 20
1.3.3 Selection and Replacement Techniques 21
1.3.4 Adding Mutation and Recombination 24
1.4 Summary and Conclusions 26

Chapter 2 Simple Learning Classifier Systems **28**
2.1 Learning Architecture 29
2.1.1 Knowledge Representation 29
2.1.2 Reinforcement Learning Component 31
2.1.3 Evolutionary Component 32
2.2 Simple LCS at Work 33
2.3 Towards a Facetwise LCS Theory 35
2.3.1 Problem Solutions and Fitness Guidance 35
2.3.2 General Problem Solutions 37
2.3.3 Growth of Promising Subsolutions 38
2.3.4 Neighborhood Search 39

2.3.5 Solution Sustenance . . . 45
2.3.6 Additional Multistep Challenges . . . 45
2.3.7 Facetwise LCS Theory . . . 46
2.4 Summary and Conclusions . . . 47

Chapter 3 The XCS Classifier System . . . 49
3.1 System Introduction . . . 49
3.1.1 Knowledge Representation . . . 50
3.1.2 Learning Interaction . . . 51
3.1.3 Rule Evaluation . . . 52
3.1.4 Rule Evolution . . . 55
3.1.5 XCS Learning Intuition . . . 56
3.2 Simple XCS Applications . . . 57
3.2.1 Simple Classification Problem . . . 57
3.2.2 Performance in Maze1 . . . 60
3.3 Summary and Conclusions . . . 62

Chapter 4 How XCS Works: Ensuring Effective Evolutionary Pressures . 64
4.1 Evolutionary Pressures in XCS . . . 65
4.1.1 Generalization due to Set Pressure . . . 66
4.1.2 Mutation's Influence . . . 68
4.1.3 Deletion Pressure . . . 70
4.1.4 Subsumption Pressure . . . 70
4.1.5 Fitness Pressure . . . 71
4.1.6 Pressure Interaction . . . 72
4.1.7 Steady State Specificity Distribution . . . 74
4.2 Validation of the Specificity Equation . . . 75
4.2.1 Fixed Fitness . . . 76
4.2.2 Constant Function . . . 78
4.2.3 Random Function . . . 79
4.3 Improving Fitness Pressure . . . 81
4.3.1 Proportionate vs. Tournament Selection . . . 81
4.3.2 Limitations of Proportionate Selection . . . 82
4.3.3 Tournament Selection . . . 84
4.3.4 Specificity Guidance Exhibited . . . 88
4.4 Summary and Conclusions . . . 90

Chapter 5 When XCS Works: Towards Computational Complexity . . . 92
5.1 Proper Population Initialization: The Covering Bound . . . 93
5.2 Ensuring Supply: The Schema Bound . . . 95
5.2.1 Population Size Bound . . . 96
5.2.2 Specificity Bound . . . 97
5.2.3 Extension in Time . . . 98
5.3 Making Time for Growth: The Reproductive Opportunity Bound . . . 99

5.3.1 General Population Size Bound . . . 100
5.3.2 General Reproductive Opportunity Bound . . . 102
5.3.3 Sufficiently Accurate Values . . . 104
5.3.4 Bound Verification . . . 105
5.4 Estimating Learning Time . . . 107
5.4.1 Time Bound Derivation . . . 108
5.4.2 Experimental Validation . . . 111
5.5 Assuring Solution Sustenance: The Niche Support Bound . . . 114
5.5.1 Markov Chain Model . . . 116
5.5.2 Steady State Derivation . . . 118
5.5.3 Evaluation of Niche Support Distribution . . . 122
5.5.4 Population Size Bound . . . 130
5.6 Towards PAC Learnability . . . 131
5.6.1 Problem Bounds Revisited . . . 132
5.6.2 PAC-Learning with XCS . . . 135
5.7 Summary and Conclusions . . . 136

Chapter 6 Effective XCS Search: Building Block Processing . . . 138
6.1 Building Block Hard Problems . . . 139
6.1.1 Fitness Guidance Exhibited . . . 140
6.1.2 Building Blocks in Classification Problems . . . 141
6.1.3 The Need for Effective BB Processing . . . 142
6.2 Building Block Identification and Processing . . . 148
6.2.1 Structure Identification Mechanisms . . . 150
6.2.2 BB-identification in the ECGA . . . 150
6.2.3 BB-Identification in BOA . . . 152
6.2.4 Learning Dependency Structures in XCS . . . 155
6.2.5 Sampling from the Learned Dependency Structures . . . 157
6.2.6 Experimental Evaluation . . . 160
6.3 Summary and Conclusions . . . 164

Chapter 7 XCS in Binary Classification Problems . . . 167
7.1 Multiplexer Problem Analyses . . . 167
7.1.1 Large Multiplexer Problems . . . 168
7.1.2 Very Noisy Problems . . . 170
7.1.3 Model Learning Mechanisms in the Multiplexer . . . 171
7.2 The xy-Biased Multiplexer . . . 173
7.3 Count Ones Problems . . . 176
7.4 Carry Problem . . . 179
7.5 Summary and Conclusions . . . 181

Chapter 8 XCS in Multi-Valued Datamining Problems 183
8.1 XCS Enhancements 184
8.2 Theory Modifications 185
8.2.1 Evolutionary Pressure Adjustment 185
8.2.2 Population Initialization: Covering Bound 186
8.2.3 Schema Supply and Growth: Schema and Reproductive Opportunity Bound 187
8.2.4 Solution Sustenance: Niche Frequencies 188
8.2.5 Obliqueness 189
8.3 Datamining 189
8.3.1 Datasets 189
8.3.2 Results 190
8.4 Summary and Conclusions 192

Chapter 9 Reinforcement Learning Problems 197
9.1 Q-learning and Generalization 198
9.2 Ensuring Accurate Reward Propagation: XCS with Gradient Descent 201
9.2.1 Q-Value Estimations and Update Mechanisms 201
9.2.2 Adding Gradient Descent to XCS 201
9.3 Experimental Validation 202
9.3.1 Multistep Maze Problems 203
9.3.2 Blocks-World Problems 209
9.4 Summary and Conclusions 211

Chapter 10 From Facetwise LCS Theory Towards Competent Cognitive Systems 213
10.1 General Facetwise LCS Analysis 213
10.1.1 Which Fitness for Which Solution? 214
10.1.2 Parameter Estimation 215
10.1.3 Generalization Mechanisms 216
10.1.4 Which Selection For Solution Growth? 218
10.1.5 Niching for Solution Sustenance 220
10.1.6 Effective Evolutionary Search 220
10.1.7 Conclusions 222
10.2 Towards LCS-based Cognitive Learning Structures 222
10.2.1 Predictive and Associative Representations 223
10.2.2 Incremental Learning 226
10.2.3 Hierarchical Architectures 227
10.2.4 Anticipatory Behavior 230
10.2.5 Conclusions 232

Chapter 11 Summary and Conclusions 233
11.1 Summary 233
11.2 Conclusions 237
11.2.1 Expansion of Facetwise System Analysis and Design Approach 237
11.2.2 LCS-based Future Research Directions 238

Appendix A Notation and Parameters 240

Appendix B Algorithmic Description 243

Appendix C Boolean Function Problems 256
Multiplexer 256
Layered Multiplexer 257
xy-Biased Multiplexer 258
Hidden Parity 259
Count Ones 259
Layered Count Ones 260
Carry Problem 260
Hierarchically Composed Problems 261

Bibliography 263

Vita 277

List of Tables

1.1 Typical optimization problem structures 9
1.2 Converged Q-values in the Maze1 problem 18

2.1 Conditions and the problem instances they match 31

3.1 The optimal population [O] in the 6-multiplexer problem. 58
3.2 The path to an optimal solution from the overgeneral side in the 6-multiplexer problem. 59
3.3 The optimal population [O] in the Maze1 problem. 61

4.1 Converged specificities in theory and in empirical results in a random function. 75
4.2 Mutation settings for desired specificities 75

6.1 Expected reward prediction and reward prediction error estimates in several typical Boolean functions 141
6.2 Exemplar reward prediction and reward prediction error estimates in the hierarchical 3-parity, 6-multiplexer problem 143
6.3 Illustrated formation of a marginal product model 151
6.4 Exemplar Binary encodings of classifiers 156

8.1 Structural properties of the investigated datasets 191
8.2 Performance of XCS with proportionate selection and tournament selection as well as with general and specific population initialization in the investigated datasets. 193
8.3 Comparisons of XCSTS with several other typically used machine learning algorithms in the investigated datasets 194
8.4 Continued comparisons of XCSTS with several other typically used machine learning algorithms in the investigated datasets. 195

List of Figures

1.1 Reinforcement learning problem framework 13
1.2 A simple MDP problem 14
1.3 A simple POMDP problem 14

2.1 General LCS learning architecture 30
2.2 Simple LCS learning in the Maze1 problem 34
2.3 Mutation's effect on classifier condition structure 40
2.4 Effect of crossover when recombining two overlapping classifier conditions . . 42
2.5 Effect of crossover when recombining two non-overlapping classifier conditions 43
2.6 Unbalanced problem space complexity 44

3.1 XCS's accuracy function 53
3.2 Prediction error estimate derivation 55
3.3 XCS learning iteration 56

4.1 Expected average specificity in an action set 69
4.2 Evolutionary pressure interaction in XCS 74
4.3 Specificity equation verification with niche and free mutation 77
4.4 Specificity equation verification with higher GA threshold and with empty population 78
4.5 Specificity equation verification in a constant function with random deletion and action set size based deletion 79
4.6 Specificity equation verification in a random function with random deletion and action set size and fitness-based deletion. 80
4.7 Proportionate selection causes learning deficiencies when decreasing the learning rate 83
4.8 Proportionate selection causes learning deficiencies when altering parameter initialization values or mutation rate. 83
4.9 Proportionate selection causes learning deficiencies when adding Gaussian noise to the payoff 84
4.10 Proportionate selection causes learning deficiencies when adding alternating noise to the payoff 84
4.11 Tournament selection makes XCS learning more robust with respect to its learning rate. 86

4.12 Tournament selection makes XCS learning more robust to offspring initialization and mutation rate . . . 86
4.13 Tournament selection makes XCS learning more robust to Gaussian noise. . 86
4.14 Tournament selection makes XCS learning more robust to alternating noise. 86
4.15 Fixed tournament sizes are inappropriate in XCS. . . . 87
4.16 Also in noisy problems, fixed tournament sizes do not support XCS learning appropriately. . . . 87
4.17 Small tournament set size proportions may hinder XCS learning. . . . 88
4.18 Large tournament set size proportions allow learning but prevent effective recombination. . . . 88
4.19 Proportionate selection does not detect the available accuracy signal reliably. 89
4.20 Tournament selection immediately detects the accuracy signal. . . . 89

5.1 The covering bound induces constraints on population size and initial specificity. 95
5.2 The schema bound requires a minimal population size and specificity. . . . 97
5.3 The reproductive opportunity bound requires a minimal population size dependent on specificity. . . . 102
5.4 Combining schema bound and reproductive opportunity bound results in a control map that bounds XCS's learning success. . . . 103
5.5 The reproductive opportunity bound is confirmed experimentally. . . . 106
5.6 The time extension of the schema bound affects performance in the case of low mutation rates. . . . 107
5.7 The general reproductive opportunity bound is confirmed experimentally. . . 108
5.8 Experiments confirm the derived time bound. . . . 112
5.9 Variations in problem length further confirm the time bound. . . . 113
5.10 Increasing problem length, the reproductive opportunity bound affects performance. . . . 113
5.11 Decreasing specificity (mutation rate) or problem length makes the time bound again most relevant. . . . 114
5.12 Several other genetic operators and parameters can influence the time bound. 115
5.13 A Markov chain model can simulate niche support. . . . 117
5.14 The niche support distribution is confirmed experimentally in the layered count ones problem. . . . 123
5.15 Proportionate selection hardly alters niche support distribution in the layered count ones problem. . . . 124
5.16 Larger GA thresholds cause an additional niching effect. . . . 125
5.17 Random deletion results in worse niching. . . . 125
5.18 Overlapping solutions may decrease niche sizes. . . . 126
5.19 Proportionate selection allows a more balanced competition between overlapping problem subsolutions. . . . 127
5.20 Overlapping solutions distort the niche support model. . . . 128
5.21 Also in overlapping problem solutions, a higher GA threshold decreases niche support deviation. . . . 129

5.22 Combining the overlapping niches to two macro niches results in a distribution that confirms the theory. . . . 130
5.23 A higher GA threshold focuses the combined niche support distributions. . . 130
5.24 Population size is bounded by niche occurrence frequency. . . . 132

6.1 The hierarchical 3-parity, 6-multiplexer problem demands effective BB processing. . . . 145
6.2 Learning success without crossover strongly depends on the chosen mutation rate in the hierarchical 3-parity, 6-multiplexer problem. . . . 146
6.3 Also the hierarchical 2-parity, 11-multiplexer problem demands effective BB processing. . . . 147
6.4 Learning success without crossover strongly depends on the chosen mutation rate in the hierarchical 2-parity, 11-multiplexer problem. . . . 148
6.5 Also the hierarchical 3-parity, 5-count ones problem demands effective BB processing. . . . 149
6.6 Bayesian decision trees and decision graphs are very suitable to model local dependency structures. . . . 155
6.7 The probabilistic model structure can be learned and used for offspring generation in various ways. . . . 158
6.8 Combining XCS with the models used in ECGA or BOA solves the hierarchical 3-parity, 6-multiplexer problem effectively. . . . 162
6.9 Combining XCS with the models used in ECGA or BOA also solves the hierarchical 2-parity, 11-multiplexer problem effectively. . . . 163
6.10 Also the hierarchical 3-parity, 5-count ones problem is effectively solvable with either model-based offspring generation method. . . . 165

7.1 Specificity guidance exhibited in the-11 multiplexer. . . . 168
7.2 Performance in the 70-multiplexer 169
7.3 Performance in the layered 70- and 135-multiplexer. . . . 170
7.4 Performance in the 20-multiplexer with $\sigma = 500$ Gaussian noise in the reward 172
7.5 Performance in the 20-multiplexer problem with large Gaussian noise. . . . 172
7.6 Performance of XCS combined with the model building mechanisms from ECGA or BOA in the 20-multiplexer 173
7.7 XCSECGA or XCSBOA performance in the 37-multiplexer. . . . 174
7.8 Performance in the xy-biased multiplexer 175
7.9 Performance in larger xy-biased multiplexer problem instances. . . . 177
7.10 In the count ones problems, specificity guidance is strong. . . . 178
7.11 Performance in (layered) count ones problem instances also comparing proportionate with tournament selection. . . . 179
7.12 Performance in the 100/7 (layered) count ones problems 180
7.13 Performance in the carry problem 181

8.1 The checker board problem 188
8.2 An unequally complex problem 188

9.1 Maze5, Maze6, and Wodds14 204
9.2 Performance without and with gradient-based updates in Maze5 and Maze6 . 205
9.3 Performance comparisons in runs with additional random bits in Maze5 and Maze6 206
9.4 Performance of the gradient approach is also Superior in Woods14. 207
9.5 Q-value estimation comparisons in Maze5 and Woods14 208
9.6 Performance in Maze6 with noisy actions 209
9.7 The investigated blocks world problem 210
9.8 Performance without and with gradient-based updates in the blocks world problem 211

10.1 Proposed modular, predictive, XCS-based learning architecture 225
10.2 Proposed modular, hierarchical, XCS-based learning architecture 230

C.1 Illustration of hierarchical 3-parity, 6-multiplexer problem 262

Introduction

Rule-based evolutionary learning systems, often referred to as *learning classifier systems* (LCSs), were originally inspired by the general principles of Darwinian evolution and cognitive learning. In fact, when John Holland proposed the basic LCS framework (Holland, 1976; Holland, 1977; Holland & Reitman, 1978), he actually referred to LCSs as *cognitive systems*. Inspired by stimulus-response principles in cognitive psychology, the systems are designed to evolve a set of production rules that convert given input into useful output. Additionally, temporary memory in the form of a *message list* was proposed to simulate inner mental states situating the system in the current environmental context.

Early work on LCSs confirmed the potential of the systems with respect to animal learning and cognition as well as application. In the first classifier system implementation, Holland and Reitman (1978) confirmed that LCSs can simulate animal behavior successfully. They evolved a representation that resulted in goal-directed, stimulus-response-based behavior satisfying multiple goals represented in resource reservoirs. Lashon Booker (1982) extended Holland's approach experimenting with an agent that needs to avoid aversive stimuli and reach attractive stimuli. Stewart Wilson (Wilson, 1985; Wilson, 1987a) confirmed the potential of LCSs to simulate artificial animals, or *animats*—triggering the animat approach to artificial intelligence (Wilson, 1991). In brief, the approach suggests to simulate animats in simulated environments to understand learning in organisms as well as to develop highly adaptive autonomous robotic systems. Goldberg (1983) successfully applied an LCS to the control of a simulated pipeline system confirming that LCSs are valuable learning systems for real-world applications as well.

Many of these publications were far reaching and somewhat visionary. The LCS framework predated and inspired the now well-established reinforcement learning field (Kaelbling, Littman, & Moore, 1996; Sutton & Barto, 1998). The *bucket-brigade* algorithm, originally used in LCSs, is very similar to other temporal difference learning techniques such as TD(λ) or SARSA (Sutton & Barto, 1998). The ambitious scenarios and the relation to animal learning, cognition, and robotics pointed towards research directions that remain mind chal-

lenging even today. Thus, most early LCS work was ahead of its time pointing towards interesting future research directions.

Despite these promising factors, no complete learning theory was developed for an LCS system. (1) Neither learning nor convergence could be assured mathematically. (2) The learning interactions in the system appeared to be too complex and remained not well-understood. (3) The learning biases of the system were only explained intuitively. (4) Competitive applications were restricted to a somewhat limited set of problems. Thus, LCSs did not receive general acceptance in the artificial intelligence or machine learning literature.

In their *critical review of classifier systems*, Wilson and Goldberg (1989) pointed out several of the most important problems in the available LCSs at that time. First, it appeared that successful reward chains were hard to learn as well as to maintain by means of the *bucket-brigade* algorithm. Second, inappropriate bidding and payment schemes obstructed generalization, enabled overgeneralization, or prevented the formation of default hierarchies. Third, the limitations of simple classifier syntax remained obscured with respect to noisy input features, continuous problem spaces, or larger binary problem spaces. Besides these challenges, Wilson and Goldberg (1989) also mentioned the importance of developing and understanding planning and lookahead mechanisms, representations for expectations, implementations of a short-term memory, and population sizing equations.

During the subsequent LCS winter, Stewart Wilson and few others continued to work in the LCS field. And it was Stewart Wilson who heralded an LCS renaissance with the publication of the two most influential LCS systems to date: (1) the zeroth level classifier system ZCS (Wilson, 1994) and (2) the accuracy-based classifier system XCS (Wilson, 1995).

Both classifier systems overcome many of the previously encountered challenges. The credit assignment mechanism in ZCS and XCS is directly related to the then well-understood Q-learning algorithm (Watkins, 1989) in the reinforcement learning (RL) literature ensuring appropriate reward estimation and propagation. Overgeneralization problems are overcome by proper fitness sharing techniques (ZCS) or the new accuracy-based fitness approach (XCS). Additionally, in XCS generalization is achieved by a niched reproduction combined with population-wide deletion, as stated in Wilson's generalization hypothesis (Wilson, 1995).

Published results suggested the competitiveness of the new LCSs (Wilson, 1994; Wilson, 1995). Solutions were found in previously unsolved interesting maze problems that require proper generalization as well as hard Boolean function problems, such as the multiplexer problem.

Later, research focused further on the XCS system solving larger Boolean function problems (Wilson, 1998) suggesting the scalability of the system. Others focused on performance investigations in larger maze problems considering action noise and generalization (Lanzi, 1997; Lanzi, 1999a; Lanzi, 1999c).

In addition to the promising experimental results, the growth of qualitative and quantitative theoretical insights and understanding slowly gained momentum. Tim Kovacs investigated Wilson's generality hypothesis in more detail and showed that XCS strives to learn complete, accurate, and minimal representations of Boolean function problems (Kovacs, 1997). Later, Kovacs investigated the appropriate fitness approach in LCSs contrasting a purely strength-based approach with XCS's accuracy-based approach (Kovacs, 2000; Kovacs, 2001). Finally, one of the most important questions was asked: *What makes a problem hard for XCS* (Kovacs & Kerber, 2001)? This question led to some insights on problem difficulty with respect to the optimal solution representation $[O]$. However, how XCS evolves this optimal solution as well as which computational requirements are necessary to successfully evolve and maintain such a solution remained obscured.

Besides the new direct insights into LCSs, genetic algorithms (GAs) are now much better understood than back in the late 1980s. The comprehensive introduction in Goldberg (1989) as well as the suggested *facetwise approach* to GA theory and design (Goldberg, 1991; Goldberg, Deb, & Clark, 1992) enabled GA researchers to understand system aspects and to combine the aspects effectively, while obeying their interactions. The design decomposition led to various theoretical advancements including a rigorous understanding of scale-up and convergence behavior, among others (Goldberg, 2002).

In addition to the *quantitative* advancements, the design decomposition also led to a rigorous *qualitative* understanding of what GAs are really searching for. Holland (1975) already hypothesized that GAs are processing schemata. However, Holland's original schema theory mainly showed the potential failure of schema processing instead of focusing on the best way to identify and propagate useful *schemata*, or *building blocks* (BBs) in the problem. BBs may be characterized as lower level dependency structures that result in a fitness increase when set to the correct values. Features in a BB structure usually interact nonlinearly with respect to their fitness influence.

It should be noted that Goldberg's facetwise analysis approach does not only facilitate system analysis and modeling but also leads to a more general system understanding and enables more effective system design (Goldberg, 2002). In the pure GA realm, for example, the GA design decomposition led to the creation of *competent GAs*—GAs that solve boundedly difficult problems quickly, accurately and reliably—including the extended compact GA

(ECGA) (Harik, 1999) and the Bayesian optimization algorithm (BOA) (Pelikan, Goldberg, & Cantu-Paz, 1999).

This thesis proposes and follows a similar decomposition approach in LCSs. With Wilson's powerful XCS system at hand, the thesis strives for a rigorous understanding of XCS functioning, computational requirements, convergence properties, and generalization capabilities. The design decomposition enables us to consider evolutionary components independently so that a precise and general system analysis is possible. Along the way, the analysis leads us to several successfully integrated system improvements. Moreover, the proposed decomposition points towards many interesting future research directions including further LCS analyses as well as the modular and hierarchical design of more advanced LCSs.

Thesis Objectives

This thesis aims for establishing a rigorous understanding of learning classifier systems, and the XCS classifier system in particular. We show which learning mechanisms can be identified, which learning biases the mechanisms cause, and how the mechanisms interact. The undertaken facetwise analysis enables us to establish a fundamental theory for population sizing, problem difficulty, and learning speed. It is shown that the derived problem bounds can be used to confirm (restricted) PAC-learning capabilities of the XCS system. Moreover, the analysis leads us to the identification and analysis of BB-hard problems in the LCS realm. We consequently integrate competent GA recombination operators solving the BB-hard problems by making evolutionary search more effective.

Besides the theoretical and mechanism-based enhancements, this thesis provides a body of experimental results from various problem domains including binary, nominal, and real-valued classification problems as well as multistep RL problems. Learning behavior is analyzed with respect to typical problem structures and problem properties.

Future research perspectives evaluate the lessons learned from the XCS analysis providing a broader understanding of LCSs and their interactive learning mechanisms. With this understanding at hand, we propose the creation of a cognitive learning system that may learn interactively and incrementally a modular, distributed, and hierarchical predictive problem representation and use the representation to pursue anticipatory, cognitive behavior.

Road Map

The remainder of this thesis is structured as follows.

The next chapter provides an overview of required background knowledge. First, we introduce optimization, classification and RL problems and discus most important structural properties, differences, and problem difficulties. Next, we provide an overview over relevant RL mechanisms. Finally we introduce GAs focusing on Goldberg's facetwise GA decomposition approach and the aspects therein most relevant for the remainder of this thesis.

Chapter 2 first gives a gentle introduction to a basic LCS system. A simple toy problem application illustrates the general functioning. Next, we discuss LCS theory and analysis. In particular, we propose a facetwise LCS theory approach decomposing the LCS architecture into relatively independent system facets. Each facet needs to be implemented appropriately to ensure the successful system application.

Chapter 3 introduces the system under investigation, that is, the accuracy-based classifier system XCS (Wilson, 1995). We illustrate XCS's learning behavior on exemplar toy problems, including classification and RL problems, revealing basic intuition behind XCS functioning. We then proceed to our XCS analysis.

XCS's major learning biases are investigated in Chapter 4. We show that fitness propagates accurate rules whereas generalization is achieved by a combination of subset-based reproduction and population-wide deletion. We derive a specificity equation that models the behavior of specificity in a population not influenced by fitness. Finally, we replace the previously applied proportionate selection with a subset-size dependent tournament selection mechanism ensuring reliable fitness pressure towards better classifiers.

Chapter 5 analyzes the computational requirements for solution growth and sustenance. We show that initial specificity and population size needs to be chosen adequately to ensure learning startup, minimal structural supply, relevant structural growth, and solution sustenance. With the additional learning time estimate, we can show that the computational effort scales in a low-order polynomial in problem length and solution complexity.

Next, we address solution search. Chapter 6 confirms that also in the classification and RL realm, effective BB structure identification and processing is necessary. We introduce statistical techniques to extract evolved lower level problem structure. The gained knowledge about dependency structures is then used to mutate and recombine offspring rules more effectively, consequently solving previously hard problems successfully.

Chapter 7 applies the resulting XCS system to diverse Boolean function problems. We investigate performance in large problems, the impact of irrelevant problem features, overlapping problem subsolutions, unequally distributed subsolution complexities, and external noise. As a whole, the chapter experimentally confirms the theoretic learning bounds and supports the derived mathematical learning robustness and scalability results.

Chapter 8 applies the XCS system to datamining problems. We compare XCS's performance with several other machine learning systems. The comparison further confirms XCS's learning competence and machine learning competitiveness. Moreover, we enhance the facetwise theory to the real-valued problem domain.

Chapter 9 then investigates multistep RL problems addressing the additional challenges of reward backpropagation and distribution. The chapter shows that XCS is a competent online generalizing RL system that is able to ignore additional irrelevant problem features with additional computational effort that is linear in the number of features. The results confirm that XCS offers a robust alternative to purely neural-based RL approaches.

With the pieces of the LCS puzzle in place, Chapter 10 finally outlines how a similar facetwise problem approach may be helpful in the analysis of other similar learning systems. Moreover, we outline how the analysis may carry over to the design of further competent and flexible LCS systems targeted to the problem at hand. In particular, we put most important LCS learning mechanisms in the light of the facetwise theory and then propose the integration of these mechanisms into cognitive learning structures.

Chapter 11 summarizes the major findings of the thesis. The conclusions outline the next steps necessary for further LCS analysis and more competent LCS design. With the facetwise perspective on LCSs at hand, we believe that the design of Holland's originally envisioned cognitive systems is finally within our grasp.

Chapter 1

Prerequisites

LCSs are designed to solve classification as well as more general *reinforcement learning* (RL) problems. LCSs solve these problems by evolving a rule-base of classifiers by the means of an RL-based critic for rule evaluation and a GA for rule evolution. Before jumping directly into the LCS arena, we first look at these prerequisites.

Since optimization and learning is comparable to a *search for expected structure*, we first look at the problem types and problem structures we are interested in. We differentiate between *optimization problems*, *classification problems*, and *reinforcement learning problems*. Each problem causes different but related challenges. Thus, successful learning architectures need to be endowed with different but related learning mechanisms and learning biases. LCSs are facing RL problems but might also be applied to classification or even optimization problems.

Apart from the necessary understanding of LCS-relevant problem types, the second major prerequisite is a general understanding of RL techniques and *genetic algorithms* (GAs). Section 1.2 introduces RL including the most relevant Q-learning algorithm (Watkins, 1989). We show that RL is well-suited to serve as an online learning actor/critic system that is capable of evaluating rules, distributing reward, and making action (or classification) decisions. Section 1.3 introduces GAs, which are well-suited to learn relevant rule structures given a fitness measure. Additionally, we highlight the importance of a *facetwise analysis* approach taken in the GA literature to promote understanding of GA functioning, scale-up behavior, and parameter settings as well as to enable the design of more elaborate, *competent GAs* (Goldberg, 2002).

Summary and conclusions summarize the most important issues addressed in this chapter pointing towards the integration of the addressed issues in LCSs.

1.1 Problem Types

LCSs may be applied in two major problem domains. One is the world of classification problems. The second is the one of RL problems usually defined by a Markov decision process (MDP) or the more general partially observable Markov decision process (POMDP).

In classification problems, feedback is provided instantly and successive problem instances are independent from each other as well as from the chosen classification. On the other hand, in RL problems feedback may be delayed in time and successive problem input depends on the underlying problem structure as well as on the chosen actions. Thus, internal reinforcement propagation becomes necessary, which poses an additional learning challenge.

Before introducing classification and RL problems, we give a short introduction to optimization problems emphasizing similarities and differences as well as typically expectable problem structures and properties.

1.1.1 Optimization Problems

An optimization problem is a problem in which a particular structure or solution needs to be optimized. Thus, given a particular solution space, an optimization algorithm searches for the best solution in the solution space. Optimization problems cover a huge range of problems including the optimization of a particular object, such as an engine, the optimization of a particular method, such as a construction process, the optimization or detection of a state of lowest energy, or the optimization of a solution to any search problem, such as the problem of satisfiability or the traveling salesman problem.

More formally, a simple binary optimization problem is defined for a problem space S that is characterized by a bit string of certain length l: $\mathcal{S} = \{0, 1\}^l$. Each bit string represents a particular problem solution. Feedback is provided in terms of a scalar reward (fitness) value that rates the solution quality. An optimization algorithm should be designed to *effectively* search the solution space for the global optimum.

Given for example the problem of optimizing a smoothie drink with a choice of additional ingredients mango, banana, and honey available, the problem may be coded by three bits indicating the absence (0) or presence (1) of each ingredient. The fitness is certainly very subjective in this example, but assuming that we like all three ingredients equally well, but prefer any combination of two of the ingredients, and like the combination of all three ingredients the most, we constructed a *one-max problem*.

Table 1.1 (first numeric column) gives possible numerical values for a four bit one-max problem. The best solution to the problem is denoted by `1111` and the fitness is determined

Table 1.1: Typical optimization problems in tabular form show in which way the quality measure can lead towards the global optimal solution (here: 1111). In the one-max problem, the closer the solution is to the optimal solution, the higher its quality. In the royal-road problem, the path leads to the optimal solution step-wise. In the needle in the haystack problem, fitness gives no hints about the optimal solution. Finally, in the trap problem, the quality measure is actually misleading since the more a solution differs from the optimal solution, the higher its fitness.

	One-Max	Royal-Road	Needle i.H.	Trap
0000	0	0	0	3
0001	1	0	0	2
0010	1	0	0	2
0100	1	0	0	2
1000	1	0	0	2
0101	2	0	0	1
1001	2	0	0	1
0110	2	0	0	1
1010	2	0	0	1
0011	2	2	0	1
1100	2	2	0	1
0111	3	2	0	0
1011	3	2	0	0
1101	3	2	0	0
1110	3	2	0	0
1111	4	4	4	4

by the number of ones in the problem. Certainly, the one-max problem is a very easy problem. There is only one (global) optimum and the closer to the global optimum, the higher the fitness. Thus, the problem provides strong *fitness guidance* towards the global optimum.

However, problems can be *misleading* in that the fitness measure may give incorrect clues in which direction to search for more promising solutions. Table 1.1 shows progressively more misleading types of problems. While the *royal-road* function still provides the steps that lead to the best solution (Mitchell, Forrest, & Holland, 1991), the *needle in the haystack* problem provides no direction clues whatsoever—only the optimum results in high fitness. The *trap* problem provides quality clues in the opposite direction: Fitness leads towards a local optimum away from the global optimum. An example of a trap problem would be a problem in which only the combination of a number of factors (e.g. ingredients) makes the solution better but the usage of only part of those factors makes the solution actually worse than the base solution that uses none of those factors.

The reader should not fall into the trap of thinking that a local optimum (if existent) and the global optimum must always be the exact inverse of a binary string. In longer, more complex problems, this certainly does not need to be the case. Finally, the meaning of zeroes and ones may be (partially) swapped so that the global optimum does not need to be all ones but the problem structure may still be identical to one of the types outlined in the table. Thus, although the examples are very simplified, they characterize important problem structures that pose different challenges to the learning algorithm.

These problem substructures may then be combined to form larger, more complex problems. Several combinations are possible. One is to simply add the fitness contribution of each substructure yielding a simple *uniformly scaled* problem. Another method is to exponentially scale the utility of each substructure, so that the fitness of the second block matters only once the optimum of the first block is found and so forth, yielding an *exponentially scaled* problem. It may be noted that in an uniformly scaled problem the blocks may be solved in parallel. On the other hand, in an exponentially scaled problem, the BBs need to be solved sequentially since the exponentially scaled fitness nearly eliminates quality influences from later blocks that are not yet most relevant.

Regardless of the problem structure, the problem may actually have multiple optimal solutions, the quality measure may be noisy, or the provision of several near-optimal solutions may be more desirable than the detection of one (completely) optimal solution. Often, an expert may want to choose from such a set of (near-) optimal solutions. In this case, a learner would be required to find not only one globally optimal solution but rather a set of several different (near-) optimal solutions.

To summarize, optimization problems are problems in which a best solution must be found given a solution space. Feedback is available that rates solutions proposed by the learner. The feedback may or may not provide hints where to direct the further search for the optimal solution. Finally, the number of optimal solutions may vary and, dependent on the problem, one or many optimal (or near optimal) solutions may need to be found.

1.1.2 Classification Problems

A classification problem poses further difficulties to the learning algorithm. Although a classification problem may be reduced to an optimization problem, the reduction is tedious and destroys much of the available problem structure and information inherent in a classification problem.

We define a classification problem as a problem that consists of problem instances $s \in \mathcal{S}$.

Each problem instance belongs to one class (traditionally in LCSs an action) $a \in \mathcal{A}$. In machine learning terms, s may be termed a feature vector and a a concept class. The mapping from $\mathcal{S}$ to $\mathcal{A}$ is represented by a *target concept* belonging to a set of concepts (that is, the *concept space*). The goal of a classification system is to learn the target concept. Thus, the classification system learns to which class a each problem instance s belongs. Most desirable properties of such a learning system are that the learner learns an *accurate* problem solution, measured usually by the percentage of correct problem instance classifications and a *general* problem solution, that is, a solution that generalizes well to other (unseen) problem instances. Given that the learner has a certain *hypothesis space* of expressible solutions, the learner looks for the accurate, maximally general hypothesis with respect to the target concept.

Like in optimization, problem decomposition can be expected to be relevant in classification problems. However, in this case BBs may not be directly related to fitness but may increase accuracy, generality, or both. Additionally, solution hypotheses may be represented in a more distributed fashion in that different subsolutions may be responsible for different problem subspaces. In this case, different BBs may be relevant in different subspaces. Thus, in contrast to optimization problems, BB propagation may need to be biased on the relevant problem subspaces.

Boolean Function Problems

In most of this work, we focus on Boolean function problems. In these problems, the problem instance space is restricted to the binary space, that is, $\mathcal{S} \subseteq \{0,1\}^l$ where l denotes the fixed problem length. Similarly, a Boolean function problem has only two output classes $\mathcal{A} = \{0,1\}$. Consequently, any Boolean function can be represented by a logical formula and consequently also by a logical formula in disjunctive normal form (DNF). Appendix C introduces the Boolean function problems investigated in this thesis showing an exemplar DNF representation and discussing their general structure and problem difficulty.

As an example, let us consider the well-known multiplexer problem, which is widely studied in LCS research (De Jong & Spears, 1991; Wilson, 1995; Wilson, 1998; Butz, Kovacs, Lanzi, & Wilson, 2001). It has been shown that LCSs are superior compared to standard machine learning algorithms, such as *C4.5*, in the multiplexer task (De Jong & Spears, 1991). The problem is of particular interest due to its dependency structure and its distributed niches, or subsolutions. The problem is defined for binary strings of length $l = k + 2^k$. The output of the multiplexer function is determined by the bit situated at the position

referred to by k position bits (usually but not necessarily located at the first k positions). The disjunctive normal form of the 6-multiplexer is

$$6MP(x_1, x_2, x_3, x_4, x_5, x_6) = \neg x_1 \neg x_2 x_3 \vee \neg x_1 x_2 x_4 \vee x_1 \neg x_2 x_5 \vee x_1 x_2 x_6, \tag{1.1}$$

for example, $f(100010) = 1$, $f(000111) = 0$, or $f(110101) = 1$. It is interesting to see that the DNF form of the multiplexer problem consists of conjunctions that are *non-overlapping*. That is, any problem instance belongs, if at all, to only one of the conjunctive terms in the problem. Later, we will see that the amount of overlap in a problem is an important problem property.

Datamining Problems

Boolean function problems are a rather restricted class of classification problems. In the general case, a problem instance s may be represented by a feature vector. Each feature may be a binary attribute, a nominal attribute, an integer attribute, or a real valued attribute. Mixed representations are possible.

We refer to such real-world classification problems as datamining problems. The problem is represented by a set of problem instances with their corresponding class. A problem instance may consist of a mixture of features and there may be more than two classification classes. Since the target concept is generally unknown in datamining problems, performance of the learner is oven evaluated by the means of stratified ten-fold cross-validation (Mitchell, 1997) that trains the system on a subset of the data set and tests it on the remaining problem instances. The data is partitioned into ten subsets and the learner is trained on nine of the ten subsets and tested on the remaining subset. To avoid sampling biases, this procedure is repeated ten times, each time training and testing on different subsets. Stratification assures that the class distribution is approximately equal in all folds. Ten-fold cross-validation is very useful in evaluating the generalization capabilities of the learner since performance is tested on previously unseen data instances.

1.1.3 Reinforcement Learning Problems

In contrast to optimization and classification problems, in RL problems feedback might not be available immediately. That is, given a problem instance there is no corresponding class or immediate feedback available. Rather, feedback is provided in terms of a scalar reinforcement value that indicates the quality of a chosen action (or classification). Additionally, successive

Figure 1.1: In RL problems, an adaptive agent interacts with an environment receiving state information and reinforcement feedback and executing actions.

problem instances may depend on each other. That is, the next input may depend on current input and on the executed action. RL problems are thus more difficult but also more natural, simulating the interaction with an actual outside world. Figure 1.1 shows the agent-environment interaction typical in RL problems.

Despite the environmental interaction metaphor, a classification problem may be redefined as an RL problem providing reward feedback about the accuracy of the chosen class, for example, a reward of 1000 for the correct class and a reward of 0 for the incorrect class. In this case reward is not delayed. We refer to such redefined classification problems as *single-step RL problems*. On the other hand, *multistep RL problems* refer to RL problems where reward is delayed and successive states depend on each other and on the chosen action. In the latter case, reward (back-)propagation is necessary.

Later, we see that LCSs are online generalizing RL mechanisms. Classification problems are usually converted into single-step RL problems when learned by LCSs. For convenience reasons, we usually refer to these single-step RL problems as classification problems. However, the reader should keep in mind that when referring to classification problems in conjunction with an LCS application, the LCS actually faces a single-step RL problem.

Two types of multistep RL problems need to be distinguished: Markov decision processes (MDPs) and partially observable Markov decision processes (POMDP).

Markov Decision Processes

We define a multistep problem as a Markov decision process (MDP) reusing notation from the classification problems where appropriate. An MDP problem consists of a set of possible sensory inputs $s \in \mathcal{S}$ (i.e. the states in the MDP), a set of possible actions $a \in \mathcal{A}$, a state transition function $f : \mathcal{S} \times \mathcal{A} \to \Pi(\mathcal{S})$, and a reinforcement function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \Re$. The transition function defines a probability distribution over next states given a current state and a current action. The reinforcement function defines the resulting reward, which

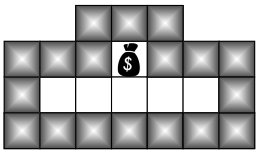


Figure 1.2: A simple MDP problem

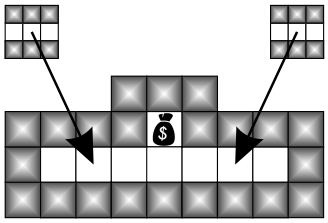


Figure 1.3: The two identical looking states turn the problem into a POMDP problem.

depends on the current state transition. At a certain point in time t, for example, given state s_t and deciding on the execution of action a_t, reward r_t and the consequent state s_{t+1} is perceived.

An MDP is called a *Markov* decision process because it satisfies the Markov property: Future probabilities and thus the optimal action decision can be determined from the current state information alone.

A simple example of a (multistep) MDP problem is a simple Maze environment shown in Figure 1.2. The learning system, or *agent*, may reside in one of the five positions in the maze perceiving the eight surrounding positions. An empty position may be coded by 0, a blocked position by 1. The money bag indicates an empty position where reward is received and the agent is reset to a randomly chosen empty position. Note that each position has a unique perception code so that the problem can be modeled by an MDP process.

Partially Observable Markov Decision Processes

More difficult than the MDP problems, are POMDP problems where current sensory input may not be sufficient to determine the optimal action. Formally, a POMDP can be defined by a state space $\mathcal{X}$, a set of possible sensations $\mathcal{S}$, a set of possible actions $\mathcal{A}$, a state transition function $f : \mathcal{X} \times \mathcal{A} \to \Pi(\mathcal{X})$. In contrast to an MDP problem, however, states are not perceived directly but are converted into a sensation using an observation function $O : \mathcal{X} \to \Pi(\mathcal{S})$ that converts a particular state into a sensation. Similarly, the reward function does not rely on the sensations but on the underlying (unobservable) states $R : \mathcal{X} \times \mathcal{A} \times \mathcal{X} \to \Re$. In contrast to the MDP, a POMDP might violate the Markov property in that optimal action decisions cannot be made solely based on current sensory input.

A simple example of a POMDP problem is shown in Figure 1.3. Although only slightly larger than the maze in Figure 1.2, the maze does not satisfy the Markov property anymore since the second empty position on the left looks identical to the second empty position

on the right. Thus, given that the agent is currently in either of the two positions, it is impossible to know if the fastest way to the reward is to the left or to the right. Only an internal state or a short-term memory (that may for example indicate that the agent came from the left-most empty position) can disambiguate the two states and allow the agent to act optimally in the position in question.

This thesis focuses on MDP problems. Applications of the learning classifier system XCS to POMDP problems can be found in (Lanzi & Wilson, 2000; Lanzi, 2000), in which the system is enhanced with internal memory.

1.2 Reinforcement Learning

Facing an MDP or POMDP problem, an RL system is the most appropriate system to solve the problem. In essence, the investigated rule-based evolutionary systems are RL systems that use GAs to evolve their state-action-value representation. Excellent introductions to RL are available (Kaelbling, Littman, & Moore, 1996; Dietterich, 1997; Sutton & Barto, 1998) and the following overview can only give a short glance at RL.

The task of an RL system is to learn an optimal behavioral policy interacting with an MDP (or POMDP) problem. A behavioral policy is a policy that makes action decisions, that is, given current sensory input (and possibly further internal state information) the behavioral policy decides on the action to execute. A behavioral policy is optimal if it results in the maximum expectable reward in the long run. The most often used expression to formalize the expected reward is the *cumulative discounted reward*:

$$E(\sum_t \gamma^t r_t), \tag{1.2}$$

where $\gamma \in [0, 1]$ denotes the discount factor that weighs the importance of more distant rewards. Setting γ to zero results in a single-step problem in which only current reward is important. Setting γ to one results in a system in which cumulative reward needs to be optimized. Usually, γ is set to values close to one such as 0.9. RL essentially searches for a behavioral policy that maximizes the cumulative discounted reward.

Looking back at our small maze problem in Figure 1.2, we can see how much reward can be expected in each state executing an optimal behavioral policy. Assuming that the environment triggers a reward of 1000 when the rewarding position is reached, the reward position itself has an expected reward of 1000, whereas the three positions that are one step away have an expected reward of 900 and the two outermost positions have an expected

reward of 810.

RL systems learn state value or state-action value representations using temporal difference learning techniques to estimate and maximize the expected discounted reward expressed in Equation 1.2. Two approaches can be distinguished: (1) *Model-free* learners learn an optimal behavioral policy directly without learning the state-transition function; (2) *Model-based* learners learn the state-transition function using it to learn or improve their behavioral policy.

1.2.1 Model-Free Reinforcement Learning

Two major approaches comprise the model-free RL realm: (1) $TD(\lambda)$ and (2) *Q-learning*. While the former needs an interactive mechanism that updates the RL-based critic and the behavioral policy in turn, the latter is doing the same more naturally. After a short overview of $TD(\lambda)$ we focus on Q-learning due to its off-policy learning and its close similarity with the RL mechanism implemented in the learning classifier system XCS.

$TD(\lambda)$ methods interactively, or in turn, update their current behavioral policy π and the critic V^π that evaluates the policy π. Given the current critic V^π, a k-armed bandit optimization mechanism may be used to optimize π. Given current policy π, V^π may be updated using the $TD(\lambda)$ strategy. Essentially, each state value $V(s)$ is updated using

$$V(s) \leftarrow V(s) + \beta(r + \gamma V(s') - V(s))e(s), \tag{1.3}$$

$$e(s) = \sum_{k=1}^{t} (\lambda\gamma)^{t-k} \delta_{s,s_k}, \quad \texttt{where } \delta_{s,s_k} = \begin{cases} 1 & \texttt{if } s = s_k \\ 0 & \texttt{otherwise} \end{cases}$$

where $e(s)$ denotes the eligibility of state s meaning its involvement in the achievement of the current reward. Parameter β denotes the learning rate somewhat reflecting the belief in the new information vs. the old information. A large β assumes very low prior knowledge resulting in a large change in the value estimate whereas a small β assumes solid knowledge, changing the value estimates only slightly. Parameter λ controls the importance of states in the past and δ monitors the occurrences of the states. With $\lambda = 0$, past states are not considered and only the currently encountered state transition is updated. Similarly, setting λ to one, all past states are considered as equally relevant. Note that the update is still discounted by γ so that the λ influence basically determines the belief in the relevancy of the current update for the states encountered in the past (Kaelbling, Littman, & Moore, 1996).

A somewhat more natural approach is Watkins' Q-learning mechanism (Watkins, 1989). Instead of learning state values, Q-learning learns state-action values effectively combining

policy and critic in one. A *Q-value* of a state action pair (s, a) essentially estimates the expected discounted future reward if executing action a in state s and pursuing the optimal policy thereafter. Q-values are updated by

$$Q(s,a) \leftarrow Q(s,a) + \beta(r + \gamma \ \texttt{max}_{a'} Q(s',a') - Q(s,a)). \tag{1.4}$$

Due to the `max` operator, the Q-value predicts the average discounted reward of the optimal policy when executing a in state s reaching state s'. Q-learning is guaranteed to converge to the optimal values if it is guaranteed that in the long run all state-action pairs are executed infinitely often and learning rate β is decayed appropriately. If the optimal Q-values are determined, the optimal policy is determined by

$$\pi^*(s) = \texttt{arg max}_a Q(s,a), \tag{1.5}$$

which choose the action that is expected to maximize the consequent Q-value if executed. To ensure an infinite exploration of the state-action space, an ϵ-greedy exploration strategy may be used:

$$\pi(s) = \begin{cases} \texttt{arg max}_a Q(s,a) & \texttt{with probability } 1-\epsilon \\ \texttt{rand}(a) & \texttt{otherwise} \end{cases}, \tag{1.6}$$

which chooses a random action with probability ϵ.

In our running Maze1 example, the Q-table learned by a Q-learner is shown in Table 1.2. The environment provides a constant reward of 1000 when the money position is reached. Additionally, after reward reception, the agent is reset to a random position in the maze. The reward is propagated backward through the maze yielding lower reward to more distant positions. In effect, in this simple setting, Q-learning is an online distance learning mechanism to a reward source where the reward prediction, that is, the *Q-value*, indicates the worthiness or value of an action given a current situation. Note that worthiness in Q-learning is defined as the expected discounted future reward using Equation 1.4. Other discount mechanisms often work just as well dependent on the task setup. For example, parameter γ could be set to one but actions may have an associated, potentially fixed cost value which will be deducted from the expected future reward. In this case, explicit discounting would be unnecessary because the environment—actually reflecting the inner architecture of the agent—would take care of the discounting, making it a potentially body-specific effort dependency.

To summarize, Q-learning learns a Q-function that determines state-dependent value estimates for each available action in each state of the environment. Due to its well-designed

Table 1.2: The Q-values in the shown Q-table reflect the value of each available action in each possible state of Maze1 (shown in Figure 1.2) where value is defined as the immediate reward plus the expected discounted future reward. The discount level is set to $\gamma = .9$.

state	sensation	$\uparrow$	$\nearrow$	$\rightarrow$	$\searrow$	$\downarrow$	$\swarrow$	$\leftarrow$	$\nwarrow$
A	11011111	.81	.81	.90	.81	.81	.81	.81	.81
B	10011101	.90	1.0	.90	.90	.90	.90	.81	.90
C	01011101	1.0	.90	.90	.90	.90	.90	.90	.90
D	11011100	.90	.90	.81	.90	.90	.90	.90	1.0
E	11111101	.81	.81	.81	.81	.81	.81	.90	.81

interactive learning mechanism and proven convergence properties, it is commonly used in RL. As we will see, Q-learning also forms a fundamental basis of the RL component used in the investigated LCSs.

1.2.2 Model-Based Reinforcement Learning

In addition to the reward prediction values, model-based RL techniques learn (potentially an approximation of) the state-transition function of the underlying MDP problem. Once the state-transition function is learned sufficiently accurately, an optimal behavioral policy can be learned by simply simulating state-transitions offline using basic dynamic programming techniques (Bellman, 1957).

In dynamic programming, the state transition function is known to the system and used to estimate the payoff for each state-action pair. The literature on dynamic programming is broad, conveying many convergence results concerning different environmental properties such as circles, probabilistic state transitions, and probabilistic reward (see e.g. Bellman, 1957; Gelb, Kasper, Nash, Price, & Sutherland, 1974; Sutton & Barto, 1998).

In model-based RL, the state-transition function needs to be learned as suggested in Sutton's DYNA architecture (Sutton, 1990). DYNA uses state-transition experiences in two ways: (1) to learn a behavioral policy (using $TD(\lambda)$ or Q-learning); (2) to learn a predictive model of the state-transition function. Additionally, DYNA executes offline policy updates (independent of the environmental interactions) using the learned predictive model. Moore and Atkeson (1993) showed that the offline learning mechanism can be sped up significantly if the offline learning steps are executed in a prioritized fashion favoring updates that promise to result in large state(-action) value changes. DYNA and related techniques usually represent their knowledge in tabular form.

Later, we will see that GAs can be applied in LCSs to learn a more generalized rep-

resentation of Q-values and predictions. LCSs essentially try to cluster the state space so that each cluster accurately predicts a certain value. GAs are used to learn the appropriate clusters.

1.3 Genetic Algorithms

As LCSs, GAs were proposed by John Holland (Holland, 1971; Holland, 1975). Somewhat concurrently ,evolution strategies (Rechenberg, 1973; Bäck & Schwefel, 1995) were proposed, which are very similar to GAs but are not discussed any further herein. Goldberg (Goldberg, 1989) provides a comprehensive introduction to GAs including LCSs. Recently, Goldberg's new book (Goldberg, 2002) provides a much more detailed analysis of GAs including scale-up behavior as well as problem and parameter/operator dependencies.

This section gives a short overview over the most important results and basic features of GAs. The interested reader is referred to (Goldberg, 1989) for a comprehensive introduction and to (Goldberg, 2002) for a detailed analysis of GAs leading to the design of *competent GAs*—GAs that solve boundedly difficult problems quickly, accurately, and reliably.

1.3.1 Basic Genetic Algorithm

GAs are evolutionary-based search or optimization techniques. GAs evolve a *population* (or set) of individuals, which are represented by a specific *genotype*. The decoded individual, or *phenotype*, specifies the meaning of the individual. For example, when facing the problem of optimizing the ingredients of a certain dish, an individual may code the presence (1) or absence (0) of all available ingredients. The decoded individual, or phenotype, would then be the actual ingredients put together.

Basic GAs face an optimization problem as specified in Section 1.1.1. Feedback is provided in form of a fitness value that specifies the quality of an individual, usually in the form of a scalar value. If we face a maximization problem, a high fitness value denotes high quality. GAs are designed to progressively generate (that is, evolve) individuals that yield higher fitness.

Given a population of evaluated individuals, *evolutionary pressures* are applied to the population generating offspring and deleting old individuals. A basic GA comprises the following steps executed in each iteration given a population:

1. selection of high fitness individuals from current population,

2. mutation and recombination of selected individuals,

3. generation of new population.

Intuitively, selection focuses the current population on more fit individuals, that is, on better solutions. Mutation is a diversification operator that searches in the syntactic, genotypic neighborhood of an individual by slightly changing its structure. Recombination, or *crossover*, recombines the structure of parental individuals in the hope of generating offspring that combines the positive structural properties of both parents. Finally, the generation of the new population decides which individuals are considered in the new population. In the simplest case, the number of reproduced classifiers equals the population size and the offspring individuals replace the old individuals. The remainder of this chapter analyzes the different methods in more detail.

1.3.2 Facetwise GA Theory

To understand a system as complex as a GA, it is helpful to partition the system into its most relevant components and investigate those components in separation. Once the single components are sufficiently well understood, they may then be combined appropriately respecting interaction constraints. This is the essential idea behind Goldberg's facetwise approach to GA theory (Goldberg, 1991; Goldberg, Deb, & Clark, 1992; Goldberg, 2002).

Before the partitioning, though, it is necessary to understand *how* a GA is supposed to work and *what* it is supposed to learn.

Since GAs are targeted to solve optimization problems evolving the solution that yields highest fitness, fitness is the crucial factor along the way to an optimal solution. GAs are assuming that there is some *fitness guidance* towards the optimal solution. However, fitness may be misleading as illustrated in the trap problem (Table 1.1). Thus, the structural assumption made in GAs is that of *bounded difficulty*, which means that the overall optimization problem is composed of substructures, or BBs, that are of bounded length. The internal structure of one BB might be misleading in that a BB might for example resemble a trap problem. However, the overall problem is assumed to be not misleading in that the best combination of BBs is expected to yield the optimal solution.

With this objective in mind, it is clear that GAs should detect and propagate subproblems effectively. The goal is to design *competent GAs* (Goldberg, 2002)—GAs that solve boundedly difficult problems quickly, accurately, and reliably. Hereby, quickly means to solve the problems in low-order polynomial time (ideally subquadratic) with respect to the

problem length, accurately means to find a solution in small δf fitness distance from the optimal solution, and reliably means to find this solution with a low error probability ϵ.

The actual GA design theory (Goldberg, 1991; Goldberg, Deb, & Clark, 1992; Goldberg, 2002) then stresses the following points for GA success:

1. Know what GAs process: Building blocks (BBs).
2. Know the GA challenge: BB-wise difficult problems.
3. Ensure an adequate supply of raw BBs.
4. Ensure increased market share for superior BBs.
5. Know BB takeover and convergence times.
6. Make decisions well among competing BBs.
7. Mix BBs well.

While we defined and discussed the BB-wise difficult problems above, we elaborate on the latter two points in the subsequent paragraphs. First, we focus on supply and increased market share. Next, we look at diversification, BB decision making, and effective BB mixing.

1.3.3 Selection and Replacement Techniques

As in real-live Darwinian evolution (Darwin, 1859), selection, reproduction, and deletion decide on life and death. Guided by fitness, individuals are evolved that are more fit for the problem at hand. Several selection and deletion techniques exist, each with certain advantages and disadvantages. For our purposes most important are (1) *proportionate selection* (also often referred to as *roulette-wheel selection*) and (2) *tournament selection*. Most other commonly used selection mechanisms are comparable to either of the two. More detailed comparisons of different selection schemes can be found in (Goldberg, 1989; Goldberg & Deb, 1991; Goldberg & Sastry, 2001).

Proportionate Selection Proportionate selection is the most basic and most natural selection mechanism: Individuals are selected for reproduction proportional to their fitness. That is, the higher the fitness of an individual, the higher its probability of being selected. In effect, high fitness individuals are evolved. Given a certain individual with fitness f_i and

an overall average fitness in the population $\overline{f}$, the proportion of individual p_i is expected to change as

$$p_i \leftarrow \frac{f_i}{\overline{f}} p_i \tag{1.7}$$

Thus, proportionate selection strongly depends on fitness scaling and the current fitness distribution in the population.

The dependency on the current fitness distribution has a strong impact on the convergence to the global best individual of a population. The more similar the fitness values in a population, the less selection pressure is encountered by the individuals. Thus, once the population has nearly converged to the global optimum, fitness values tend to be similar so that selection pressure due to proportionate selection is weak. This effect is undesirable if a single global solution is searched for.

However, complete convergence may not necessarily be desired or may even be undesired if multiple solutions are being searched for. In this case, proportionate selection mechanisms might actually be appropriate given a reasonable fitness value estimate. For example, as investigated elsewhere (Horn, 1993; Horn, Goldberg, & Deb, 1994), proportionate selection can guarantee that all similarly good solutions (or subsolution niches) can be maintained with a population size that grows linearly in the number of niches and logarithmically in the time they are assured to be maintained as long as fitness sharing is used and the niches are sufficiently non-overlapping.

Tournament Selection In contrast to proportionate selection, tournament selection does not depend on fitness scaling (Goldberg & Deb, 1991; Goldberg, 2002). In tournament selection, tournaments are held among randomly selected individuals. The tournament size s specifies how many individuals take part in a tournament. The individual with the highest fitness wins the tournament consequently undergoing mutation and crossover being a candidate for the next generation.

If replacing the whole population by individuals selected by tournament selection, the best individuals can be expected to be selected s times so that the proportion p_i of the best individual i can be expected to grow with s, that is:

$$p_i \leftarrow s p_i \tag{1.8}$$

That means that the best individuals are expected to take over the population quickly where the proportion of the best individuals grows exponentially in s. Thus, in contrast to proportionate selection, which naturally stalls late in the run, tournament selection pushes

the better individuals until the best available individual takes over the whole population.

Supply Note that before selection can actually be successful, BBs need to be available in the population. This leads to the important issue of initial *supply* of better individuals. If the initial population is too small to guarantee the presence of better individuals, a GA relies on mutation to generate better individuals by chance. However, an accidental successful mutation is very unlikely (exponentially decreasing in the number of features necessary for a better individual). Thus, a sufficiently large population with an initially sufficiently large diversity is mandatory for GA success.

Niching Very important for a successful application of GAs in the LCS realm is the parallel maintenance of equally important subsolutions. Usually, *niching* techniques are applied to accomplish this maintenance. Hereby, two techniques reached significant impact in the literature: (1) crowding and (2) sharing.

In crowding (De Jong, 1975; Mahfoud, 1992; Harik, 1994) the replacement of classifiers is restricted to classifiers that are (usually syntactically) similar. For example, in the restricted tournament selection technique (Harik, 1994), offspring is compared with a subset of other individuals in the population. The offspring competes with the (syntactically) closest individual, replacing it, if its fitness is larger.

In sharing techniques (Goldberg & Richardson, 1987) fitness is shared among similar individuals where similarity is defined by an appropriate distance measure (e.g. Hamming distance in the simplest case). The impact of sharing was investigated in detail elsewhere (Horn, 1993; Horn, Goldberg, & Deb, 1994; Mahfoud, 1995). Horn, Goldberg, and Deb (1994) highlight the importance of fitness sharing in the realm of LCSs showing the important impact of sharing on the distribution of the population and potentially near infinite niche maintenance due to the applied sharing technique.

Although sharing can be beneficial in non-overlapping (sub-)solution representations, the more the solutions overlap, the less beneficial the fitness sharing techniques become. Horn proposes that sharing works successfully as long as the overlap proportion is smaller than the fitness ratio between the competing individuals. Thus, if the individuals have identical fitness, only a complete overlap eliminates the sharing effect. In general, the higher the degree of overlap, the smaller the sharing effect and thus the higher the probability of losing important BB structures due to genetic drift.

1.3.4 Adding Mutation and Recombination

Selection alone certainly does not do much good. In essence, selection uses a complicated method to find the best individuals in a given set of individuals. Certainly, this can be done much faster by simple search mechanisms. Thus, selection needs to be combined with other search techniques that search in the neighborhood of the current best individuals. The two basic operators that accomplish such a search in GAs are *mutation* and *crossover*.

Simple mutation takes an individual as input and outputs a slight variation of the individual. In the simplest form when coding an individual in binary, mutation randomly flips bits in the individual with a certain probability μ. Effectively, mutation searches in the syntactic neighborhood of the individual where the distance of the neighborhood is defined by the Hamming distance.

Crossover is designed to recombine current best individuals. Thus, rather than searching in the syntactic neighborhood of one individual, crossover searches in the neighborhood defined by two individuals resulting in a certain type of knowledge exchange among the crossed individuals. In the simplest case when coding individuals in binary, *uniform crossover* exchanges each bit with a 50% probability, whereas one-point or two-point crossover choose one or two positions in the bit strings and exchange the right or the inner part of the resulting partition, respectively.

It should be noted that uniform crossover does not assume any relationship among bit positions whereas one- and two-point crossover implicitly assume that bits that are close to each other depend on each other since longer substrings are exchanged. One-point crossover additionally assumes that beginning and ending are unrelated to each other whereas two-point crossover assumes a more circular coding structure. For more detailed analyses on simple crossover operators see for example (Bridges & Goldberg, 1987; Booker, 1993).

It is important to recognize the effects of mutation and crossover alone, disregarding selection for a moment. If selecting randomly and simply mutating individuals, mutation causes a general *diversification* in the individuals. In the long run, each attribute in an individual will be set independently uniformly distributed resulting in a population with maximum entropy in its individuals. In combination with selection, mutation causes a search in the syntactic neighborhood of an individual where the spread of the neighborhood is controlled by the mutation rate. Mutation may be biased incorporating potentially available problem knowledge to improve the neighborhood search as well as to obey problem constraints.

Recombination, on the other hand, exchanges information among individuals syntactically dependent on the structural bias in the applied crossover operator. Selecting randomly,

random crossover results in a randomized shuffling of the individual genotypes. In the long run, crossover results in a random distribution of attribute values over the classifiers but does not affect the proportion of each value in the population.

In conjunction with selection, crossover is designed to process BBs. That is, crossover is meant to exchange BBs among individuals in order to generate better offspring. Goldberg (2002) compares this very important exchange of individual substructures with innovation. Since innovation essentially refers to a successful (re-) combination of available knowledge in a novel manner, GAs are essentially designed (or should be designed) to do just that—being innovative in a certain sense.

Unfortunately, standard crossover operators are not guaranteed to propagate BBs effectively because they may also be significantly disruptive, destroying important BB structures when recombining individuals. To prevent such disruption and design a more directed form of innovation, recently, estimation of distribution algorithms (EDA) were introduced to GAs (Pelikan, Goldberg, & Lobo, 2002; Larrañaga, 2002). These algorithms estimate the current solution distribution of the current best individuals in a problem and use this distribution estimation to constrain the crossover operator or to generate better offspring directly from the distribution. In Chapter 6, we incorporate mechanisms from the extended compact GA (ECGA) (Harik, 1999), which learns a non-overlapping BB structure, as well as the Bayesian optimization algorithm (BOA), which learns a Bayes model of the BB structure, into the investigated XCS classifier system.

Certainly the choice of an appropriate selection method for the task at hand is very important. Additionally to the impact on speed of growth and convergence, selection is very relevant with respect to mutation and recombination. In essence, the two methods interact in that selection propagates better individuals, and mutation and crossover search in the neighborhood of these individuals for even better solutions. Consequently, the growth of the better individuals, often characterized by their *take over time* (Goldberg, 2002), needs to be balanced with the search in the neighborhood of the current best solutions. Too strong selection pressure may result in a collapse of the population to one only locally optimal individual preventing effective search and innovation via mutation and crossover. On the other hand, too weak selection pressure may allow *genetic drift* that can cause the loss of important BB structure by chance.

These ideas led to the proposition of a *control map* for GAs (Goldberg, Deb, & Thierens, 1993; Thierens & Goldberg, 1993; Goldberg, 1999) that characterizes a region of selection and recombination parameter settings in which a GA can be expected to work. The region is bounded by *drift*, when selection pressure is too low, *cross-competition*, when selection is

too high, and *mixing*, when knowledge exchange is too slow with respect to the selection pressure. The mixing bound essentially characterizes the boundary below which knowledge exchange caused by crossover is not strong enough with respect to the selection pressure applied. Essentially the time until the expected generation of a better individual needs to be shorter than the mentioned take over time of the current best individual. For further details on these important factors for a successful GA design and application, the interested reader is referred elsewhere (Goldberg, 2002).

1.4 Summary and Conclusions

This chapter introduced the three major problem types relevant to this thesis: (1) optimization problems, (2) classification problems, and (3) reinforcement learning problems. Optimization problems require effective search techniques to find the best solution to the problem at hand. Classification problems require a proper structural partition into different problem classes. RL problems additionally require reward backpropagation.

RL techniques are methods that solve Markov Decision Process (MDP) problems online applying dynamic programming techniques in the form of temporal difference learning to estimate discounted future reward. The behavioral policy is optimized according to the estimated reward values. In the simplest case, RL techniques use tables to represent the expected cumulative discounted reward with respect to a state or a state-action tuple.

The most prominent RL technique is Q-learning, which is able to learn an optimal behavioral policy online without learning the underlying state transition function in the problem. Q-learning learns off-policy meaning that it does not need to pursue its optimal policy in order to learn the optimal policy.

Genetic algorithms (GAs) are optimization techniques derived from the idea of Darwinian evolution. GAs combine fitness-based selection with mutation and recombination operators to search for better individuals with respect to the current problem. Individuals usually represent complete solutions to a problem.

Effective building block (BB) processing is mandatory in order to solve problems of bounded difficulty. Effective BB processing was recently successfully accomplished using statistical modeling techniques that estimate the dependency structures in the current population and bias recombination towards this dependency.

Niching techniques are very important when the task is to maintain a subset of equally good solutions or different subsolutions for different subspaces (niches) in the problem space. Fitness sharing and crowding are the most prominent niching methods in GAs.

Goldberg's *facetwise analysis* approach to GA theory significantly improved GA understanding and enabled the design of competent GAs. Although the facetwise approach has the drawback that the found models may need to be calibrated with respect to a problem, the advantages of the approach are invaluable for the analysis and design of highly interactive systems. First, crude models of system behavior are derivable for cheap. Second, analysis is more effective and more general since it is adaptable to the actual problem at hand and focuses only on most relevant problem characteristics. Finally, the approach enables more effective system design and system improvement due to the consequently identified rather independent aspects of problem difficulty.

The remainder of this thesis investigates how RL and GA techniques are combined in LCSs to solve classification problems and RL problems effectively. Similar to the facetwise decomposition of GA theory and design, we propose a facetwise approach to LCS theory and design in the next chapter. We then pursue the facetwise approach to analyze the XCS classifier system qualitatively and quantitatively. The analysis also leads to the design of improved XCS learning mechanisms and to the proposition of more advanced LCS-based learning architectures.

Chapter 2

Simple Learning Classifier Systems

Learning Classifier Systems (LCSs) (Holland, 1976; Booker, Goldberg, & Holland, 1989) are rule-based evolutionary learning systems. A basic LCS consists of a population of rules, an apportionment of credit system, which generally applies adapted reinforcement learning (RL) (Kaelbling, Littman, & Moore, 1996; Sutton & Barto, 1998) techniques, and a rule evolution mechanism, which is usually implemented by a genetic algorithm (GA) (Holland, 1975). The classifier population codes the current knowledge of the LCS. The apportionment of credit system estimates rule utility. Based on the estimated utilities, the evolutionary mechanism generated offspring classifiers.

LCSs can be distinguished between *online learning* LCSs and *offline learning* LCSs. Moreover, they can be distinguished between LCSs that evolve a single solution, often referred to as Michigan-style LCSs, and LCSs that evolve a set of solutions, often referred to as Pittsburgh-style LCSs. Pittsburgh-style LCSs are usually applied in offline learning scenarios only.

This thesis proposes and pursues a facetwise approach to LCS analysis and design. We propose the modular analysis of LCSs focusing on appropriate identification, propagation, sustenance, and search of a complete and accurate problem solution. While most of this thesis focuses on one particular online-learning Michigan-style LCS, that is, the accuracy-based learning classifier system XCS (Wilson, 1995), the basic analysis and comparisons as well as the drawn conclusions should readily carry over to other types of LCSs neither restricted to online-learning LCSs nor to Michigan-style LCSs.

This chapter first gives a general introduction to a simple LCS in tutorial form assuming knowledge about both the basic functioning of a GA as well as basic RL principles. An illustrative example provides more details on basic LCSs. Section 2.3 introduces our facetwise theory approach. Summary and conclusions wrap up the most important lessons of this

chapter.

2.1 Learning Architecture

LCSs have a rather simple but interactive learning structure combining the strengths of GAs in search, pattern recognition, pattern propagation, and innovation with the value estimation capabilities of RL. The result are learning systems that generate online a generalized state-action value representation. Depending on the complexity of the problem and the number of different states, the generalization capability is able to save space as well as time to learn an optimal behavioral policy. Similarly, in a classification problem scenario, LCSs may be able to detect distributed dependencies in data focusing on the most relevant ones. Conveniently, the dependencies are usually directly reflected in the emerging rules allowing not only statistical datamining but also more qualitatively oriented datamining.

The basic interaction of the three major components of a learning classifier system and the environment is illustrated in Figure 2.1. While the RL component controls the interaction with the environment, the evolutionary component evolves the problem representation, that is, classifier condition and action parts. Thus, the learning mechanism interacts not only with the environment but also within itself in that the evolutionary component relies on appropriate evaluation measures from the RL component and, vice versa, the RL component relies on appropriate classifier structure generated by the GA component to be able to estimate future reinforcement accurately. The interaction between the two components is the key to LCS success although proper interaction alone does not assure success. This will become particularly evident in our later analyses. The following paragraphs provide a more concrete definition of a simple learning classifier system LCS1.

2.1.1 Knowledge Representation

To be more concrete, we define a learning classifier system LCS1. LCS1 consists of a population of maximum size N of classifiers. Each classifier i consists of a condition part C_i, an action part A_i and a reward prediction value R_i. A classifier i predicts reward $R_i \in \Re$ given its condition C_i is satisfied and given further that action $A_i \in \mathcal{A}$ is executed.

Depending on the representation of the problem space $\mathcal{S}$ (e.g. binary, nominal, real...), conditions may be defined in various ways from simple exact values, over value ranges, to more complex, kernel-based conditions such as radial basis functions. Each classifier condition defines a problem subspace. The population of classifiers as a whole usually covers the

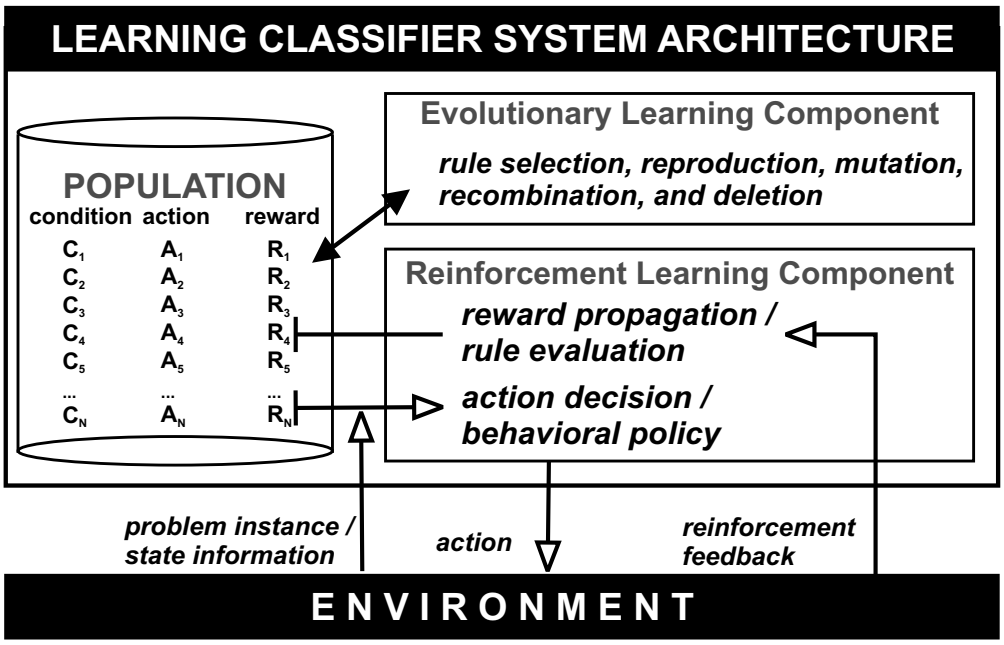


Figure 2.1: The major components in a learning classifier system are the knowledge base, which is expressed by a population of rules, the evolutionary component, which evolves classifier structure based on their reward estimation values, and the RL component, which evaluates rules and makes action (or classification) decisions.

complete problem space. Classifiers with non-overlapping conditions (specifying completely different subspaces) are independent with respect to the representation and may be considered as implicitly connected by an `or` operator. Overlapping classifiers compete for activity if their actions differ.

Let's consider the binary case corresponding to our definition of a Boolean function problem (Chapter 1) as well as our example of Maze1 (Figure 1.2). The binary input string $\mathcal{S} = \{0,1\}^l$ is matched with the conditions that specify the attributes it requires to be correctly set. Traditionally, in its simplest form a condition is represented by the ternary alphabet $C \in \{0,1,\#\}^l$ where the *don't care* symbol # matches both zero and one.[1] If the condition part is satisfied by the current problem instance, the classifier is said to *match*. Table 2.1 shows an example of a potential problem instance and all conditions that would match this problem instance.

Introducing a little more notation, a classifier cl may be said to have a certain *specificity* $\sigma(cl)$. In the binary case, we may define specificity as the ratio of the number of specified

[1]Note that the hash symbol might not be expressed explicitly representing a condition part by a set of position-value tuples corresponding to the attributes in the traditional representation that are set to zero or one. This representation has significant computational advantages when the rules only specify few attributes.

Table 2.1: All classifier conditions in which all specified attributes are identical to the corresponding values in the problem instance match the current problem instance. The more general a condition part, the more problem instances it matches.

instance	matching conditions				matching problem instances					condition
1001	1001								1001	1001
	100#	10#1	1#01	#001				1001	1000	100#
	10##	1#0#	#00#			1011	1010	1001	1000	10##
	1##1	#0#1	##01		1111	1101	...	0011	0001	###1
	###1	##0#	#0##	1###	1111	1110	...	0001	0000	####
	####									

attributes to the overall number of attributes. For example, given a problem of problem length l and a classifier with k specialized (not don't care) attributes, the classifier has a specificity of $\frac{k}{l}$. Similar definitions may be used for other problem domains and other condition representations. Essentially, specificity is a measure that characterizes how much of the problem space a classifier covers. A specificity of one means that only one possible problem instance is covered whereas a specificity of zero means that all problem instances (the whole problem space) is covered. Thus, the larger the specificity of a classifier, the less of the problem space is covered by the classifier. Specificity is an important measure in LCSs useful for deriving and quantifying evolutionary pressures, problem bounds, and parameter values, among others. Subsequent chapters derive several problem bounds and performance measures based on specificity.

2.1.2 Reinforcement Learning Component

Given a current problem instance, an LCS forms a *match set* $[M]$ of all classifier in the population $[P]$ whose conditions are satisfied by the current problem instance. The match set reflects the knowledge in the current state (given the current problem instance). An LCS uses the match set $[M]$ to make its action decisions.

The action decision is made by the behavioral policy π controlled by the RL component. In the simple case, we can use an adapted epsilon-greedy action selection mechanism that averages over the expected reward predicted by all matching classifiers to predict action values. The behavior policy may be written as

$$\pi_{LCS1}(s) = \begin{cases} \texttt{arg max}_a \sum_{\{i \in [M] | A_i = a\}} \frac{R_i}{|\{i \in [M] | A_i = a\}|} & \texttt{with probability } 1-\epsilon \\ \texttt{rand}(a) & \texttt{otherwise} \end{cases}, \qquad (2.1)$$

where s denotes the current problem instance, a the chosen action and $\{i \in [M]|A_i = a\}$ the set of all classifiers in the match set $[M]$ whose action part specifies action a. As a result of the action decision $a' = \pi_{LCS1}(s)$ a corresponding action set $[A]$ is formed that consists of all classifiers in the current match set $[M]$ that specify action a' ($[A] = \{i \in [M]|A_i = a'\}$).

After the reception of the resulting immediate reward r and the next problem instance s_{+1} yielding match set $[M]_{+1}$, all classifier reward predictions in $[A]$ are updated using the adapted Q-learning equation:

$$R_i \leftarrow R_i + \beta(r + \gamma \texttt{max}_a \sum_{\{i \in [M]_{+1}|A_i = a\}} \frac{R_i}{|\{i \in [M]|A_i = a\}|} - R_i), \tag{2.2}$$

estimating the expected discounted future reward by the average over all participating classifiers. Thus, there is not only one value that estimates a Q-value, as in Q-learning, but a set of classifiers together estimate the resulting Q-value. If all conditions were completely specific, LCS1 would do Q-learning predicting each Q-value by the means of a single, fully specific classifier.

2.1.3 Evolutionary Component

At this point, we know how reward prediction values are updated and how they are propagated in an LCS. What remains to be addressed is how the underlying conditional structure evolves. Two components are responsible for classifier structure generation and evolution: (1) a covering mechanism and (2) a GA.

The covering mechanism is mostly applied early in a run to ensure that all problem instances are covered by at least one rule. Given a problem instance, a rule may be generated that sets the value of each attribute with probability $(1-P_{\#})$ to the current value. Note that covering may be mainly avoided by initializing sufficiently general classifiers. Particularly, if adding classifiers for all possible actions with completely general conditions (all don't care symbols) to the population, covering will not be necessary because the completely general classifiers always match. In this case, the GA will take care of structure evolution starting from completely (over-) general classifiers.

In its simplest form, we use a stead-state GA which is similar to an $(N + 2)$ evolution strategy mechanism (Rechenberg, 1973; Bäck & Schwefel, 1995). In each learning iteration, the evolutionary component selects two offspring classifiers using for example proportionate selection based on the reward predictions R. The selected two classifiers are reproduced, mutated and recombined yielding two offspring classifiers. For example, mutation can change

a condition attribute with a certain probability μ to one of the other possible values. Additionally, the action part may be mutated with probability μ. Recombination combines the condition parts with a probability χ applying for example uniform crossover. The two offspring classifiers replace two other classifiers, which can be selected using proportionate selection on the inverse of their fitness (e.g. $\frac{1}{1+R}$).

In the case of such a simple GA mechanism, the GA searches in the syntactic (genotypic) local neighborhood of the current population. Selection is biased towards selecting higher reward offspring consequently propagating classifier structures that predict high reward on average and deleting classifiers that expect low reward on average.

In combination with the RL component, the GA should evolve structures that receive high reward on average. Unfortunately though, this is not enough to ensure successful learning. Section 2.3 introduces a general theory of learning in LCSs that reveals the drawbacks of this simple LCS system.

2.2 Simple LCS at Work

Let's do a hypothetical run of our simple LCS1 on the Maze1 problem (see Figure 1.2 on page 14). Perceptions are coded starting north coding clockwise indicating an obstacle by 1 and a free position by 0. The money position is perceived as a free position. For example, consider the population shown in Figure 2.2 (generated by hand). Classifier 1 is a classifier that identifies a move to the north whenever there is no obstacle to the east whereas Classifier 2 considers a move to the north whenever there is no obstacle on the west side. The shown reward values reflect the expected reward received if all situations were equally likely and the correct Q-values were propagated (effectively an approximation of the actual values).

In the illustrated learning iteration, the provided problem instance `01011101` indicates that there is a free space north, east, and west (as a result of residing in the position just south of the money position). The problem instance triggers the formation of a match set as indicated in Figure 2.2. In the example, the classifiers shown in the match set predict a reward of 950.75 for action $\uparrow$ and 900 for action $\downarrow$. When action $\uparrow$ is executed, the money position is reached, a reward r of 1000 is received, and the reward predictions of all classifiers in the current action set are updated applying Equation 2.2 (shown are updates using learning rate $\beta = 0.2$). It can be seen how the reward estimation values of all classifiers increase towards 1000.

Finally, a GA is applied that selects two classifiers from the population, reproduces,

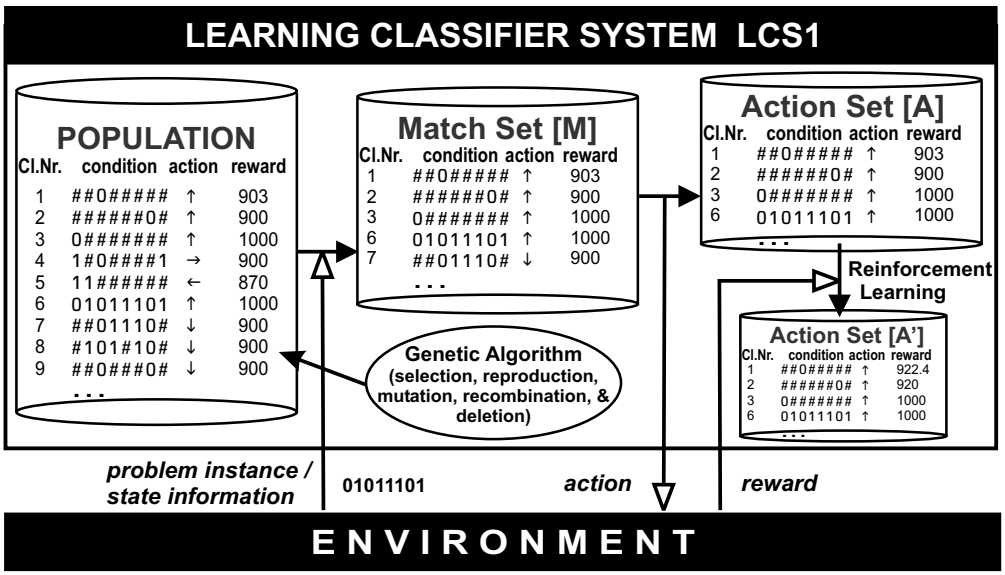


Figure 2.2: In a typical learning iteration, an LCS receives a current problem instance consequently forming the match set $[M]$. Next, an action is chosen (in this case action $\uparrow$) and executed in the environment. Since the resulting reward equals one thousand, the reward estimates are increased (shown is an increase using learning rate $\beta = 0.2$). Finally, the genetic algorithm may be applied on the update population.

mutates, and recombines them, and replaces two existing classifiers by the new classifiers. For example, the GA may select classifiers three and six, reproducing them, mutating them to e.g. $3'$=(0####1###,$\uparrow$) and $6'$=(0101#101,$\uparrow$), recombining them using one-point crossover to e.g. 3^*=(0####1101,$\uparrow$) and 6^*=(0101####,$\uparrow$), and finally reinserting 3^* and 6^* into the population replacing two other classifiers (e.g. the lower reward classifiers two and five).

We can see that the evolutionary process propagates classifiers that specify how to reach the rewarding position. Due to the bias of reproducing classifiers that predict higher reward, on average higher-reward classifiers will be reproduced more often and will be deleted less often. Mutation and crossover serve as the randomized search operators that are looking for better solutions in the (syntactically) local neighborhood of the reproduced classifiers.

Our example exhibits already several fundamental challenges for simple learning classifier systems: the problem of *strong overgenerals* (Kovacs, 2000), investigated in detail elsewhere (Kovacs, 2003), the problem of generalization, and the problem of local vs. global competition. These issues are the subject of the following section, in which we develop a facetwise theory approach for LCS analysis and design.

2.3 Towards a Facetwise LCS Theory

The introduction of LCS1 may have clarified several important properties of the general LCS learning architecture such as its online learning property and its general rule evaluation (using reinforcement methods) and rule learning (using evolutionary computation methods) procedure in each iteration. However, it remains to be understood, if and how the interactions may assure that a complete problem solution is learned.

It is clear that classifiers that receive a high reward on average will be selected for reproduction more often and will be deleted less often. Thus, the GA mechanism propagates their structure by searching in the local neighborhood of these structures. However, can we make specific learning projections? How general will the evolved solution be? How big can the problem be given a certain population size? How distributed will the final population be?

This section addresses these issues and develops a facetwise theory for LCSs. The section first starts with a general outlook of which solution LCSs evolve and *how* this may be accomplished. Next, a theory that addresses *when* this may be accomplished is outlined.

2.3.1 Problem Solutions and Fitness Guidance

The above sections showed that LCSs are designed to evolve a distributed problem solution represented by a population of classifiers. The condition of each classifier defines the subspace for which the classifier is responsible. The action specifies the proposed solution in the defined subspace. Finally, the reward measure estimates the consequent payoff of the chosen action. It is desired that the action corresponding to the highest estimated payoff equals to the best action in the current problem state.

To successfully evolve such a distributed problem solution we need to prevent overgeneralization and to have sufficient fitness guidance (that is, a fitness gradient) towards better classifiers. The two issues are discussed next starting with the problem of overgenerality and especially strong overgenerals.

The problem of *strong overgenerals* (Kovacs, 2001) concerns a particular generality vs. specificity dilemma. The problem is that a general classifier cl_g (one whose conditions are satisfied in many problem instances or states) may have a higher reward prediction value on average than a more specialized classifier cl_s that may match in a subset of cl_g. Consequently, the GA will propagate cl_g. Additionally, given that the actions are different in cl_g and cl_s, action A_{cl_g} has preference over action A_{cl_s}. However, action A_{cl_s} may yield a higher reward in the situations in which also cl_s matches. Thus, although clearly action A_{cl_s} would be more

appropriate in the described scenario, action A_{cl_g} will be chosen by the behavioral policy and will be propagated by the GA. An example clarifies the problem.

Considering classifiers one and four in Figure 2.2, it can be seen that both classifiers match in the left-most position in Maze1 (Figure 1.2), which is perceived as `11011111`. The best action in this position is certainly to move to the east which yields a discounted reward of 900 (as correctly predicted by Classifier 4). However, Classifier 1 predicts a slightly higher reward for action $\uparrow$ since its reward reflects the average reward encountered when executing action $\uparrow$ in its matching states A, B, C, and D assuming that all states are visited equally often and that action north is executed equally often in each of the states. Thus, the incorrect action $\uparrow$ will be executed more often although the action $\rightarrow$ would be correct.

The basic problem shows that good classifiers cannot be distinguished from bad classifiers as easily as initially thought. In effect, it needs to be questioned if the fitness approach—deriving fitness directly from the absolute reward received—is appropriate, or rather, in which problems a direct reward-based fitness approach is appropriate. Kovacs (Kovacs, 2001; Kovacs, 2003) analyzes this problem in detail. Essentially, strong overgenerals are possible in any problem in which more than two reward values may be perceived (and thus essentially in all but the most trivial multistep problems). The severeness of this problem consequently demands that the fitness approach itself should be changed.

Solutions to this problem are the *accuracy-based* fitness approach in XCS (Wilson, 1995), investigated in detail in later chapters, and the application of *fitness sharing* techniques as applied in the ZCS system (Wilson, 1994). In the former case, local fitness is modified requiring explicitly that the reward prediction of a classifier is accurate (that is, it has low variance). In the latter case, reward competition in the problem subspaces (defined by the current problem instance) can cause the extinction of strong overgeneral classifiers.

The problem of strong overgenerals illustrates how important it is to exactly define (1) the structure of the problem addressed (to know which challenges are expected) and (2) the objective of the learning system (to know how the system may "misbehave"). Our strength-based system LCS1 for example will work fine (with respect to strong-overgenerals) in all classification problems since only two reward values are received and reward is not propagated. However, other challenges may have to be faced as investigated below.

Once we can assure that classifiers in the optimal solution will have the highest fitness values, we need to ensure that fitness itself guides the learning process towards these optimal classifiers. This is accomplished by acknowledging that overgeneral classifiers have lower fitness values by definition of the optimal solution. To what degree fitness guides towards higher fitness values depends on the fitness definition and problem properties. Later chapters

address this problem in detail with respect to the XCS classifier system and typical problem properties.

2.3.2 General Problem Solutions

While the problem of strong overgenerals is concerned with a particular phenomenon resulting from the interaction of a classifier structure, reinforcement component, and evolutionary computation component, the problem also points to a much more fundamental problem: the problem of generalization. When we introduced LCSs above, we claimed that they can be characterized as online generalizing RL systems. And in fact, as the classifier structure suggests, rules are often matching in several potential problem instances or states. However, until now it was not addressed at all why and how the evolutionary component may propagate more general classifiers instead of more specific ones.

Considering again our Maze1 (Figure 1.2) and the exemplar population shown in Figure 2.2, classifiers 6 and 3 are actually containing the same amount of information: Both classifiers only match in maze position C and both classifiers predict that a move to the north yields a reward of 1000. Clearly, Classifier 6 is syntactically much more specialized. The general concept of Classifier 3, which only requires a free space to the north and does not care about any other position, appears more appealing and might be the best concept in the addressed environment. In general, the aim is to stay as general as possible identifying the minimal set of attributes necessary to predict a Q-value correctly.

On the other hand, consider classifiers 7 and 8 in Figure 2.2, both classifiers predict that moving south will result in a reward of 900. Both classifiers are syntactically equally specific, that is, both have an order of five (five specified attributes). However, Classifier 7 is semantically more general than Classifier 8 because it matches in more states than Classifier 8 (all three states below the money vs. only the states south and south east of the money). Classifier 9 is semantically as general as Classifier 7 but it is syntactically more general. Later, we will see that XCS biases its learning towards syntactic and semantic generality using different mechanisms.

In the general case, the quest for generality leads us to a multi-objective problem in which the objectives are to learn the underlying function as accurately as possible and to represent the function with the most general classifiers and the least number of classifiers possible. This problem is addressed explicitly elsewhere (Llorà & Goldberg, 2003; Llorà, Goldberg, Traus, & Bernadó, 2003) in which a Pareto-front of high fitness, high generality classifiers is propagated. Other approaches, including the mechanism in XCS, apply a somewhat

constant generalization pressure that is overruled by the fitness pressure if higher fitness is still achievable. Yet another approach, recently proposed by (Bull, 2003), is applied in the ZCS system. In this case, reproduction causes fitness deduction. The lost fitness can only be regained by reapplication. More general classifiers will be reapplied faster and thus do not suffer as much from the reproduction penalty and eventually they take over the population.

In Chapter 4, we analyze the effect of different generalization mechanisms in more detail.

2.3.3 Growth of Promising Subsolutions

Once we know which solution we intend to evolve with our LCS system, how fitness may guide us to the solution, and how the solution will tend to be general, we need to ensure that our intentions can be put into practice. Thus, we need to ensure the growth of higher fitness classifiers.

To do this, it is necessary to ensure that classifiers with higher fitness are available in the population. Once we can assure that better classifiers are present in the population, we need to assure that the RL component and the genetic component can interact successfully to reproduce higher fitness classifiers. Thus, we need to assure that the RL component has enough *evaluation time* to detect higher fitness classifiers reliably. Moreover, the genetic component needs to reproduce and thus propagate those better classifiers before they tend to be deleted.

The first aspect is related to the *BB supply* issue in GAs. However, due to the distributed problem representation, a more diverse supply may need to be ensured and the definition of supply differs. In essence, the initial population needs to be general enough to cover the whole problem space but it also needs to be diverse enough to have better solutions available for identification and propagation. The diversification and specialization effect of mutation may support the supply issue. These ideas become much more concrete when investigating the XCS classifier system in Chapter 5.

Note that supply is not only relevant in the beginning of a run but it is actually relevant at all stages of the learning progress continuously requiring the supply or generation of better offspring classifiers. However, the issue is most relevant in the beginning because later in the run, the currently found distributed problem solution usually significantly restricts the search space to the immediate surrounding of these solutions. In the beginning of a run, the whole search space is the surrounding and any randomized search operator such as mutation can be expected—dependent on the problem—to have a hard time to find better classifiers by chance.

Once better classifiers are available, we need to ensure that they are identified. Since the RL component requires some time to identify better classifiers (iteratively updating the reward estimates), better classifiers need to have a sufficiently long survival time. Thus, offspring classifiers need to undergo several evaluations before they are deleted.

Finally, if better classifiers are available and the RL component has enough time to identify them, it is necessary to ensure that the genetic component propagates them. Thus, the survival time needs to be also long enough to ensure reproduction of better classifiers faster. Additionally, genetic search operators may need additional time to effectively detect important problem substructures and subspaces. Due to the potentially unequally distributed problem complexity in problem space (see, for example, the problem in Figure 2.6), different time may need to be available for different problem subspaces. Chapter 5 investigates these issues in detail with respect to the XCS classifier system.

2.3.4 Neighborhood Search

Once we can assure that higher fitness classifiers undergo reproduction and thus grow in the population, we need to implement effective neighborhood search in order to detect even better problem solutions. These problem solutions can be expected to lie in the neighborhood of the currently best subsolution or in further partitions of the current subsolution subspace defined by the classifier conditions. Especially the neighborhood search is very problem dependent and thus it is generally impossible to define an optimal neighborhood search operator. We now first look at simple mutation and crossover and their impact on genetic search. Next, we discuss the issue of local vs. global search bias in somewhat more detail.

Mutation

Mutation generally searches in the syntactic neighborhood of a selected classifier. A simple mutation operator changes some attributes in the condition part of a classifier as well as the class of the classifier. If the class is changed, the new classifier basically considers the possibility that the relevant attributes for one class might also be appropriate for a reward prediction in another class. This might be helpful especially in multistep problems where classifiers often develop a condition that identifies a certain state in the environment.

Mutation of the condition part can have three types of effects that may apply in combination if several attributes of one condition part are mutated: (1) generalization, (2) specialization, (3) knowledge transfer. Considering the ternary alphabet $C \in \{0, 1, \#\}^l$, given an attribute with value 0, mutation may change the attribute to #. In this case the

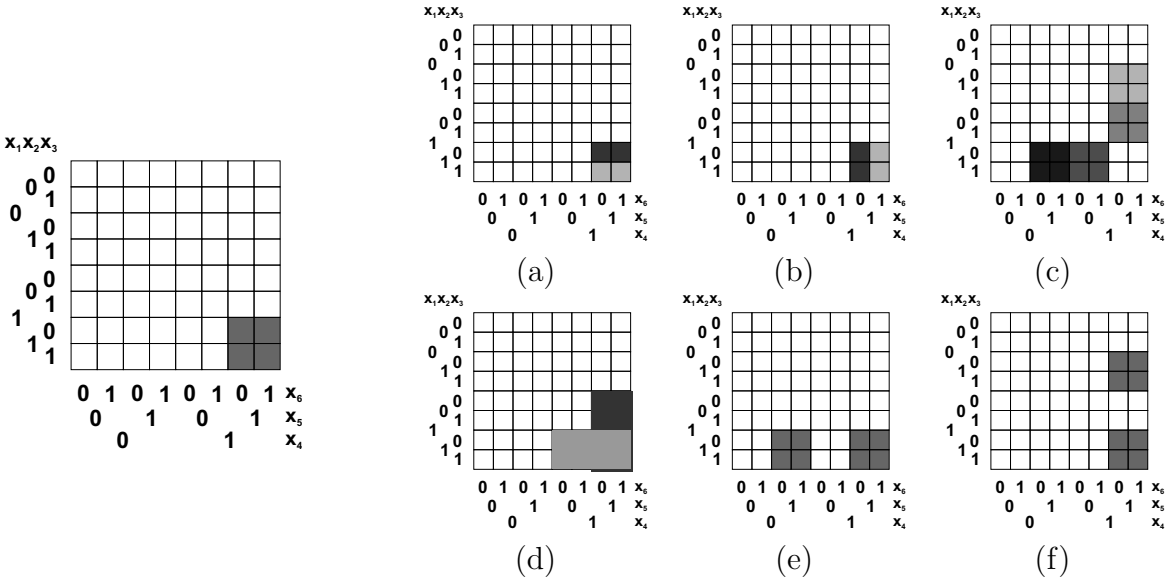


Figure 2.3: Mutation in action: The parental classifier condition on the left-hand side can be either specialized by one mutation (a,b), projected into a neighboring subspace (c), or generalized including a neighboring subspace (d,e,f). Different grays represent different classifier conditions.

classifier is generalized (its specificity decreases) since its condition covers a larger problem subspace (in the binary case, double the space). On the other hand, if the attribute actually was a don't care symbol before and it is mutated to 0 or 1, the classifier is specialized (its specificity is increased) covering a smaller portion of the problem space (in the binary case, half of the space). Finally, a specified attribute (e.g. 0) may be changed to another specific value (e.g. 1) effectively shifting the subspace structure of the rest of the classifier condition to another part of the search space.

Figure 2.3 illustrates the three mutation cases. Given the parental classifier condition `11#11#`, cases (a) and (b) show the potential cases for specialization, that is `11011#`, `11111#` and `11#110`, `11#111`, respectively. Case (c) shows knowledge transfer when a specialized attribute is changed to the other value. In our example, the classifier condition may change to `11#10#`, `11#01#`, `10#11#`, or `01#11#`, effectively moving the hypercube to other subspaces in the problem space. Cases (d), (e), and (f) show how generalization by mutation may change the condition structure. Note that in each case, the original hypercube is maintained and another hypercube with a similar structure is added. The shown cases cover all possible mutation cases of one attribute in the parental classifier. Depending on the mutation probability μ, additional mutations are exponentially less probable but may result in a more extended neighborhood search.

In effect, mutation searches in the general/specific neighborhood of the current solution as well as transfers structure from one subspace to another (close) subspace. The effectiveness of mutation consequently depends on the complexity distribution over the search space. If syntactic neighborhoods are structurally similar, mutation can be very effective. If there are strong differences between syntactic neighborhoods, mutation can be quite ineffective. The specialization effect of mutation is always present and should always be helpful in identifying proper problem subspaces.

Regardless of the problem search effect, mutation is a general diversification operator that causes the evolutionary search procedure to search in the local (syntactic) neighborhood of currently promising solutions. Doing this, mutation tends to generate an equal number of each available value for an attribute. In the ternary alphabet, mutation consequently pushes the population towards an equal amount of zeros, ones, and #-symbols in classifier conditions. With respect to specificity, mutation pushes towards a 2:1 specific:general distribution. In Chapter 4, we show in detail how mutation influences specificity in XCS and how the specificity influence interacts with the other search operators in the system.

Crossover

The nature of crossover strongly differs from mutation in that crossover does not search in the local neighborhood of one classifier but it combines classifier structures. The result are offspring classifiers that specify substructures of the parental classifiers. In contrast to mutation, simple crossover does not affect overall specificity since the combined specificity of the two offspring classifiers equals the combined specificity of the parents. Thus, although the specificity of individual offspring classifiers might differ from the parental specificity, average specificity is not affected.

For example, consider the overlapping classifier conditions `11####` and `1##00#` shown on the left-hand side of Figure 2.4. The maximal space crossover searches in is restricted to the maximal general offspring that can be generated from the two parental classifiers, that is `1#####`. Other offspring structures are possible that are progressively closer to the parental structure as indicated in Figure 2.4 (a) showing progressively more specialized classifier conditions as well as in (b) and (c) showing the four other possible offspring cases. It can be seen that crossover consequently searches in the maximal problem subspace defined by the two classifier conditions. Structure of the two classifiers is exchanged, projected onto other subspaces included in the maximal subspace defined by the two classifiers.

If the parental classifiers are not overlapping, crossover searches in the maximum sub-

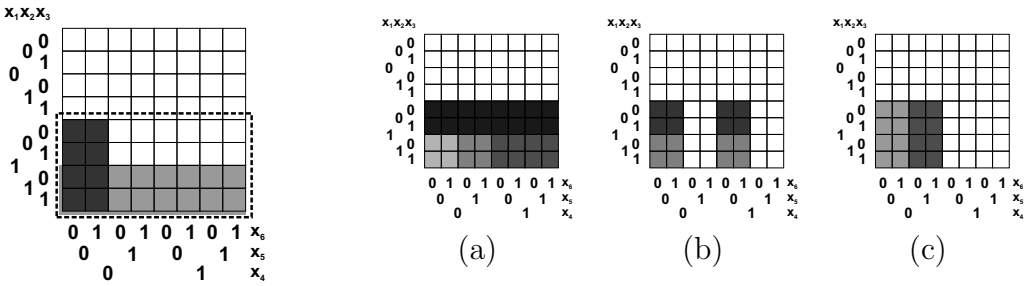


Figure 2.4: Crossover of two overlapping parental conditions searches in the subspace indicated by the outer dashed box (left-hand side). More specialized offspring conditions (shown in brighter gray) are included in the more general offspring conditions (shown in darker gray).

space defined by the non-overlapping parts of the two subspaces. In the example shown in Figure 2.5 the parental classifiers `1##00#` and `00####` are non-overlapping and the maximum subspaces are characterized by either the upper half of the search space (`0#####`) or the lower half of the search space (`1#####`). Note that essentially in the case of non-overlapping classifiers, the structural exchange may or may not be fruitful and strongly depends on the underlying problem structure. If structure is similar over the whole search space, then crossover may be beneficial. However, if the structure differs over the search space, crossover can be expected to be mainly disruptive.

In general, crossover recombines previously successful substructures transferring those substructures to other, close problem spaces. Depending on the complexity and uniformity of problem spaces, crossover may be more or less effective. Also, it can be expected that the recombination of classifiers that cover structurally related problem spaces will be more effective than the recombination of unrelated classifiers. Thus, a good restriction of classifier recombination should result in a more effective genetic search.

As in GAs, the issue of building blocks (BBs) comes into mind. BBs in simple LCSs are complexity units that define a subspace that yields high reward on average. The identification and effective propagation of BBs consequently should result in another type of more effective search in LCSs. Considering such search biases, it needs to be kept in mind, though, that we are searching for a distributed problem representation in which different subsolutions might consist of different BB structures.

Local vs. Global Search Bias

Due to the distributed problem solution representation, LCSs face another challenge in comparison to standard GAs. Although the incoming problem instances must be assumed

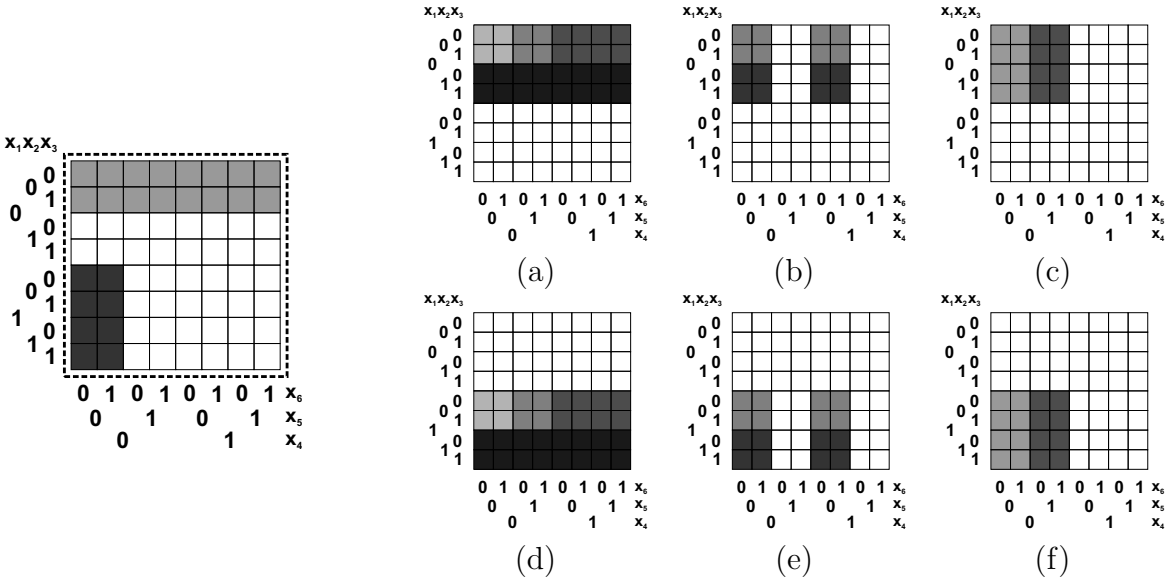


Figure 2.5: Given two non-overlapping parental conditions (left-hand side), crossover explores structure in either of the resulting maximal general, non-overlapping subspaces. Figures a,b,c,d,e,f show condition subspaces that can be generated by uniform crossover. Brighter gray subspaces are included in darker subspaces but also form a condition subspace on their own.

to be structurally and semantically related, the currently evolved problem subsolutions may be unequally advanced and accurate subsolutions may be very unequally complex depending on the problem subspace they apply in. For example, in some regions of the problem space a very low specificity might be sufficient to predict reward correctly whereas in other problem spaces further specificity might be necessary.

Figure 2.6 illustrated a problem space in which some subspaces are highly complex (subspace `00******`) in that the identification of the problem class (black or white) requires several further feature specifications. On the other hand, the rest of the problem space is fairly simple (subspaces `01******`, `10******`, and `11******`) in that classes can be distinguished easily by the specification of only one or two additional features (`01**1***`→`1`, `10**0***`→`1`, and `110**1**`→`1` and `111**0**`→`1`).

Similar differences in complexity can be found in RL problems. For example, in the simple Maze problem in Figure 1.2 and the exemplar population shown in Figure 2.2, we can see that a classifier of specificity 1/8 (order one—specifying the empty position to the north) suffices to predict a reward of 1000 correctly when going north (Classifier 3). On the other hand, in order to predict 900 reward correctly when heading south, at least two positions need to be specified (see Classifier 9). Thus, necessary specificities as well as

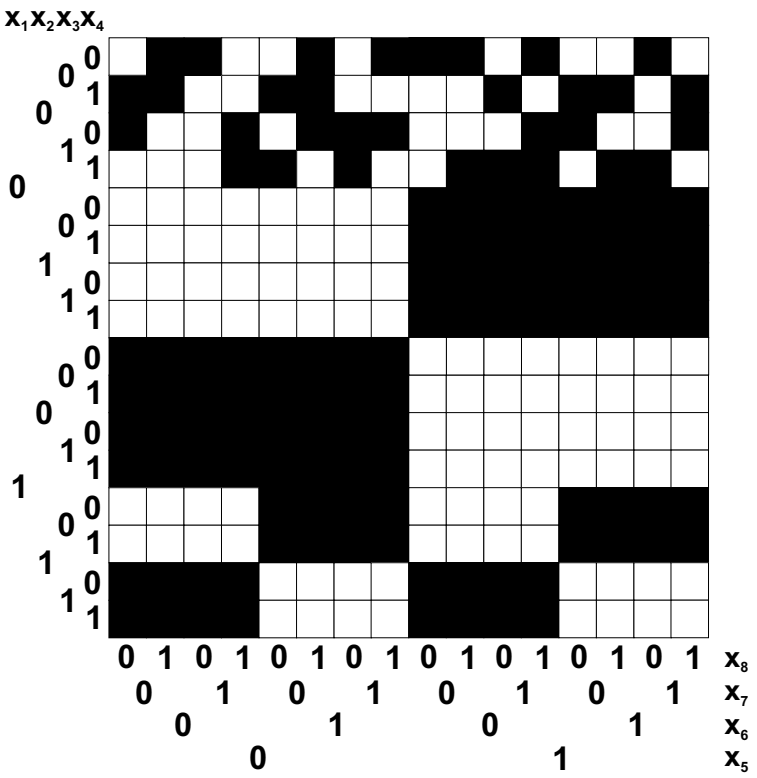


Figure 2.6: Problem spaces may vary in complexity dependent on the problem and the problem subspace. Thus, niching as well as suitable search mechanisms that are able to maintain a complete problem representation that reflects the different complexities are mandatory.

necessary specified positions might differ depending on the problem instance. In effect, global selection and recombination might not be appropriate since different classifiers might represent solutions to completely different subspaces. Thus, a balanced search combining local and global knowledge should be most effective.

In particular, it is desired to only recombine those classifiers that are compatible in the sense that they address related problems as specified by their condition parts. In the simple LCS, selection, recombination, and crossover are usually applied globally in that two potentially unrelated classifiers are selected from the overall population. The consequent recombination then is likely to be ineffective if solution structure structure varies over the problem space. We will see that XCS circumvents this problem reproducing classifiers in action sets. However, if the problem has a much more global structure, further bias towards global selection may result in additional learning advantages. We address this problem in further detail in Chapter 6.

2.3.5 Solution Sustenance

As seen above, a simple GA selects individuals from the whole population, mutates and potentially recombines them. Since a GA is usually designed to optimize a problem searching for one globally optimal solution, the sustenance of different solutions is important only early in the run. It is usually assured through initial diversity and supply as well as the balance between selection and recombination and/or mutation operators.

A different problem arises if the goal is to evolve not only the best solution but a distributed set of solutions. Since LCSs are designed to generate the best solution for every potential problem instance, the population in an LCS needs to evolve optimal solutions for all potential problems in the solution space. Each problem instance represents a new (sub-)problem (defined by the problem instance) that might be related to the other (sub-)problems but represents a new problem in a common problem space.

Due to the necessity of a distributed representation, niching methods (Goldberg & Richardson, 1987; Horn, 1993; Horn, Goldberg, & Deb, 1994; Mahfoud, 1992) are even more important in LCSs than they are in standard GAs. Since LCSs need to evolve and maintain a representation that is able to generate a solution for every possible problem instance, it needs to be assured that the whole problem space is covered by the distributed solution representation. Several niching methods are applicable and different LCSs use different techniques to assure the sustenance of a complete problem solution. Later chapters address the niching issue in further detail.

2.3.6 Additional Multistep Challenges

The above issues are all targeted to problems in which immediate reward indicates the appropriateness of an action. Also, problem instances were thought to be independent of each other. However, in multistep problems, such as the ones modeled by general MDP or POMDP processes, reward propagation as well as self-controlled problem sampling comes into play.

Evaluation and reproduction time issues need to be reconsidered in this case. Since successive problem instances depend on the executed action, which is chosen by the LCS agent itself, problem instance sampling and problem instance frequencies may become highly skewed. Thus, the time issues with respect to classifier selection and propagation in problem subspaces may need to be reevaluated. Additionally, the RL and GA component may be influenced by the current action policy and vice versa, the action policy may depend on the RL and GA constraints. For example, exploration may be enhanced in problem subspaces in

which no appropriate classifier structure evolved so far. This relates to prioritized sweeping and other biased search algorithms in RL (Moore & Atkeson, 1993; Sutton, 1991).

Additionally, the challenge of reward propagation needs to be faced. Since reinforcement may be strongly delayed depending on the problem and current problem subspace, accurate non-disruptive reward backpropagation needs to be ensured. Thus, competent RL techniques need to be applied. Additionally, though, due to the rule generalization component, neural network type update techniques may be advantageous as shown in detail in Chapter 9 for the XCS classifier system.

2.3.7 Facetwise LCS Theory

The above issues lead to the proposition of the following LCS problem decomposition. To assure that a classifier system evolves an accurate problem solution, the following aspects need to be respected:

1. ***Design evolutionary pressures most effectively***:
   Appropriate fitness definition, parameter estimation, and rule generalization.
   (a) **Fitness guidance**: Fitness needs to be defined appropriately to guide towards the optimal solution disabling strong overgenerals.
   (b) **Parameter estimation**: Classifier parameters need to be initialized and estimated most effectively to avoid fitness disruption in young and unexperienced classifiers.
   (c) **Adequate generalization**: Classifier generalization needs to push towards a maximally accurate, maximally general problem solution preventing overgeneralization.
2. ***Ensure solution growth and sustenance***:
   Effective population initialization, classifier supply, classifier growth and niche sustenance.
   (a) **Population initialization**: Effective classifier initialization needs to ensure classifier evaluation and GA application time.
   (b) **Schema supply**: Minimal order schema representatives need to be available.
   (c) **Schema growth**: Schema representatives need to be identified and reproduced before deletion is expected to enable solution growth.

(d) **Solution sustenance**: Niching techniques need to ensure the sustenance of a complete problem solution.

3. ***Enable effective solution search***:
   Effective mutation, recombination, and structural biases.

   (a) **Effective mutation**: Mutation needs to search the neighborhoods of current subsolutions effectively ensuring diversity and supply.

   (b) **Effective recombination**: Recombination needs to combine building block structures efficiently.

   (c) **Local vs. global structure**: Search operators need to detect and exploit global structural similarities but also differences in different local problem solution subspaces.

4. ***Consider additional challenges in multistep problems***:
   Behavioral policies, sampling biases, and reward propagation.

   (a) **Effective behavior**: A suitable behavioral policy needs to be installed to ensure appropriate environmental exploration and knowledge exploitation.

   (b) **Problem sampling reconsiderations**: Subproblem occurrence frequencies might be skewed due to environmental properties and the chosen behavioral policy. Thus, evaluation and reproduction times need to be reevaluated and might be synchronized with currently chosen behavior and encountered environmental properties.

   (c) **Reward propagation**: Accurate reward propagation needs to be ensured to allow accurate classifier evaluation.

The next chapters focus on this LCS problem decomposition investigating how the accuracy-based XCS classifier system faces and solves these problem aspects. Along the lines, we also improve the XCS system in the light of several of the theory facets.

2.4 Summary and Conclusions

In this chapter we introduced the general structure of an LCS. We saw that LCSs are suitable to learn classification problems as well as RL problems. Additionally, we saw that

LCSs are online learning systems that learn from one problem instance at a time, potentially interacting with a real environment or problem.

LCSs learn a distributed problem solution represented by a population of classifiers (that is, a set of rules). Each classifier specifies a condition, which identifies the problem subspace in which it is applicable, an action or classification, and a reward prediction, which characterizes the suitability of that action given the current situation. Although we only considered conditions in the binary problem space, applications in other problem spaces including nominals and real valued inputs are possible. Chapter 8 investigates the performance of the accuracy-based classifier system XCS in such problem spaces.

Classifiers in the population are *evaluated* by RL mechanisms. Classifier structure is *evolved* by a steady-state GA. Thus, while the RL component is responsible for the identification of better classifiers, the GA component is responsible for the propagation of these better classifiers. The consequent strong dependence of the two learning components on each other needs to be considered when designing an LCS.

Dependent on the classifier condition structure, mutation and crossover have slightly different effects in comparison to search in a simple GA. Mutation changes the condition structure searching for other subsolutions in the neighborhood of the parental condition. However, mutation does not only cause a diversification pressure but it also has a direct effect on the *specificity* $\sigma(cl)$ of the condition of a classifier cl. Crossover, on the other hand, does not affect combined classifier specificity but recombines problem substructures. As in GAs, crossover may be disruptive and BB identification and propagation mechanisms may improve genetic search. In contrast to GAs, though, LCSs search for a distributed problem solution so that crossover operators need to distinguish and balance search influenced by local and global problem structure.

The proposed facetwise LCS theory approach for analysis and design is expected to result in the following advantages: (1) Simple computational models of the investigated LCS system can be found. (2) The found models are generally applicable. (3) The models are easily modifiable to the actual problem and representation at hand. (4) The models provide a deeper and more fundamental problem and system understanding. (5) The investigated system can be improved effectively focusing on the currently most restricting model facets. (6) Future, more advanced LCS systems can be designed much more straightforwardly, targeted to effectively solve the problem at hand. The rest of this thesis pursues the facetwise analysis approach, which confirms the expected advantages.

Chapter 3

The XCS Classifier System

The creation of the accuracy-based classifier system XCS (Wilson, 1995) can be considered as a milestone in classifier system research. XCS addresses the general LCS challenges in a very distributed way. The problem of generalization is approached by niched reproduction in conjunction with panmictic (population-wide) deletion. The problem of strong overgenerals is solved by deriving classifier fitness from the estimated accuracy of reward predictions instead of from the reward predictions themselves. In effect, XCS is designed to not only evolve a representation of the best solution for all possible problem instances but rather to evolve a complete and accurate *payoff map* of all possible solutions for all possible problem instances.

This chapter introduces the XCS classifier system. We provide a concise description of problem representation and all fundamental mechanisms. The algorithmic description found in Appendix B provides an exact description of all fundamental mechanisms in XCS facilitating the implementation of the system. After the introduction of XCS, Section 3.2 shows XCS's performance on simple toy problems. Chapters 4 and 5 then investigate how and when XCS is able to learn a solution developing a theory of XCS's learning capabilities.

3.1 System Introduction

As all LCSs, XCS evolves a set of rules, the so-called *population* of *classifiers*. Rules are evolved by the means of a steady-state genetic algorithm (GA). A classifier usually consists of a condition and an action part. The condition part specifies when the classifier is applicable and the action part specifies which action, or classification, to execute. XCS differs in its GA application and its fitness approach. This section gives a concise introduction to XCS starting with knowledge representation progressing to learning iteration and ending

with learning evaluation and the genetic learning component. The introduction focuses on binary problem representations. Nominals and real-valued representations are introduced in Chapter 8.

3.1.1 Knowledge Representation

The population of classifiers represents a problem solution in a probabilistic disjunctive normal form where each classifier specifies one conjunctive term in the disjunction. Thus, each classifier can be regarded as an expert in its problem subspace specifying a confidence value about its expertise.

Each classifier consists of five main components and several additional estimates.

1. *Condition part* C specifies when the classifier is applicable.
2. *Action part* A specifies the proposed action (or classification or solution).
3. *Reward Prediction* R estimates the average reward received when executing action A given condition C is satisfied.
4. *Reward prediction error* ε estimates the mean absolute deviation of R with respect to the actual reward.
5. *Fitness* F estimates the scaled, relative accuracy (scaled, inverse error) with respect to other, overlapping classifiers.

As in LCS1, in the binary case condition part C is coded by $C \in \{0, 1, \#\}^L$ identifying a hypercube in which the classifier is applicable, or *matches*. Action part $A \in \mathcal{A}$ defines one possible action or classification. Reward prediction $R \in \Re$ is iteratively updated resulting in a moving average measure of encountered reward received in the recent problem instances in which condition C matched and action A was executed. Similarly, the reward prediction error estimates the absolute moving average of the error of the reward prediction. Finally, fitness estimates the moving average of the accuracy of the classifier's reward prediction relative to other classifiers that are applicable at the same time.

Each classifier maintains several additional parameters. The *action set size estimate as* estimates the moving average of the action sets it is applied in. It is updated similar to the reward prediction R. The *time stamp ts* specifies the time when the classifier was part of a GA competition. The *experience* counter *exp* counts the number of parameter updates the classifier underwent so far. The numerosity *num* specifies the number of (micro-) classifiers,

this macro-classifier actually represents. In this way, multiple identical classifiers can be represented by one actual classifier in the population speeding up computation (for example, matching).

3.1.2 Learning Interaction

Learning usually starts with an empty population. Alternatively, the population may be initialized generating random classifiers whose condition part have an average specificity of $1 - P_{\#}$ (that is, each attribute in the condition part is a don't care symbol with probability $P_{\#}$ and zero or one otherwise).

Given current problem instance $s \in \mathcal{S}$ at iteration time t, the set of all classifiers in $[P]$ whose conditions match s is called the *match set* $[M]$. If some action is not represented in $[M]$, a covering mechanism is applied.[1] Covering creates classifiers that match s (inserting #-symbols, similar to when the population is initialized at random, with a probability of $P_{\#}$ at each position) and specify unrepresented actions. Given a match set, XCS can estimate the payoff for each possible action forming a *prediction array* $P(\mathcal{A})$,

$$P(A) = \frac{\sum_{cl.A=A \wedge cl \in [M]} cl.R \cdot cl.F}{\sum_{cl.A=A \wedge cl \in [M]} cl.F}. \tag{3.1}$$

Classifier parameters are addressed using the dot notation. Essentially, $P(A)$ reflects the fitness-weighted average of all reward prediction estimates of the classifiers in $[M]$ that advocate classification A. The prediction array is used to determine the appropriate classification. Several action selection policies (that is, behavioral policies) may be applied. Usually, XCS chooses actions randomly during learning and it chooses the best action $A_{max} = \texttt{arg max}_A P(A)$ during testing. All classifier in $[M]$ that specify the chosen action A comprise the action set $[A]$.

After the execution of the chosen action, feedback is received in the form of scalar reward $R \in \Re$, which is used to update classifier parameters. Finally, the next problem instance is received and the next problem iteration begins increasing the iteration time t by one.

[1] Covering is sometimes controlled by the parameter θ_{mna} that requires that at least θ_{mna} actions are covered. For simplicity, we set θ_{mna} per default to the number of possible actions or classifications $|\mathcal{A}|$ in the current problem.

3.1.3 Rule Evaluation

XCS iteratively updates its population of classifiers with respect to the successive problem instances. Parameter updates are usually done in the order: prediction error, prediction, fitness.

In a classification problem, classifier parameters are updated with respect to the immediate feedback R in the current action set $[A]$. In an RL problem, all classifiers in $[A]$ are updated with respect to the immediate reward R plus the estimated discounted future reward as follows:

$$Q = max_{A \in \mathcal{A}} P^{t+1}(A), \tag{3.2}$$

where the $(t+1)$ term refers to the prediction array in the consequent learning iteration $t+1$.

Reward prediction error ε of each classifier in $[A]$ is updated by:

$$\varepsilon \leftarrow \varepsilon + \beta(|\rho - R| - \varepsilon), \tag{3.3}$$

where $\rho = r$ in classification problems and $\rho = r + \gamma Q$ in RL problems. Parameter $\beta \in [0, 1]$ denotes the learning rate influencing accuracy and adaptivity of the moving average reward prediction error. Similar to the learning rate dependence in RL (see Chapter 1), a higher learning rate β results in less history dependence and thus faster adaptivity but also higher variance if different reward values may be received.

Next, reward prediction R of each classifier in $[A]$ is updated by:

$$R \leftarrow R + \beta(\rho - R), \tag{3.4}$$

with the same notation as in the update of ε. Note how XCS essentially applies a Q-learning update. However, Q-values are not approximated by a tabular entry but by a collection of rules expressed in the prediction array $P(\mathcal{A})$.

The fitness value of each classifier in $[A]$ is updated with respect to its current scaled relative accuracy κ', which is derived from the current reward prediction error ε as follows:

$$\kappa = \begin{cases} 1 & \text{if } \varepsilon < \varepsilon_0 \\ \alpha \left(\frac{\varepsilon_0}{\varepsilon}\right)^{\nu} & \text{otherwise} \end{cases}, \tag{3.5}$$

$$\kappa' = \frac{\kappa \cdot num}{\sum\limits_{cl \in [A]} cl.\kappa \cdot cl.num}. \tag{3.6}$$

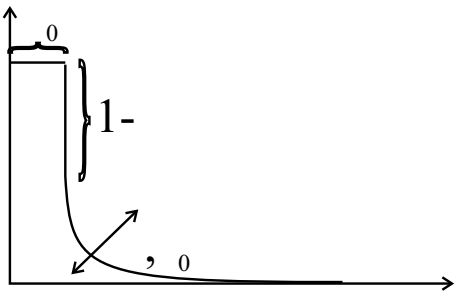


Figure 3.1: The accuracy derivation κ of the current reward prediction error ε has an error tolerance of ε_0. Additionally, exponent ν controls the degree of the drop off and ε_0 scales the drop off. Finally, parameter α emphasizes the difference between accuracy and in-accuracy.

Essentially, κ measures the current absolute accuracy of a classifier using a power function with exponent ν to further prefer low error classifiers. Threshold ε_0 denotes a threshold of maximal error tolerance. That is, classifiers whose error estimate ε drops below threshold ε_0 are considered accurate. The derivation of accuracy κ with respect to ε is illustrated in Figure 3.1. The relative accuracy κ' then reflects the relative accuracy with respect to the other classifiers in the current action set. In effect, each classifier in $[A]$ competes for a limited fitness resource that is distributed dependent on $\kappa \cdot num$.

Finally, fitness estimate F is updated with respect to the current action set relative accuracy κ' as follows:

$$F \leftarrow F + \beta(\kappa' - F). \tag{3.7}$$

In effect, fitness reflects the moving average, set-relative accuracy of a classifier. As before, β controls the sensitivity of the fitness.

Additionally, the action set size estimate as is updated similar to the reward prediction R but with respect to the current action set size $|[A]|$:

$$as \leftarrow as + \beta(|[A]| - as), \tag{3.8}$$

revealing a similar sensitivity to action set size changes $|[A]|$ dependent on learning rate β.

Parameters R, ε, and as are updated using the *moyenne adaptive modifiée* technique (Venturini, 1994). This technique sets parameter values directly to the average of the so far encountered cases as long as the experience of a classifier is less than $1/\beta$.

Each time the parameters of a classifier are updated, experience counter exp is increased by one. Additionally, if genetic reproduction is applied to classifiers of the current action set, all time stamps ts are set to the current iteration time t.

Using the Widrow-Hoff delta rule, the reward prediction of a classifier approximates the mean reward it encounters in a problem. Thus, the reward prediction of a classifier can be approximated by the following estimate:

$$cl.R \approx \frac{\sum_{\{s|cl.C \text{ matches } s\}} p(s)p(cl.A|s)\rho(s, cl.A)}{\sum_{\{s|cl.C \text{ matches } s\}} p(s)p(cl.A|s)}, \tag{3.9}$$

where $p(s)$ denotes the probability that state s is presented in the problem and $p(a|s)$ specifies the conditional probability that action (or classification) a is chosen given state s.

Given that reward prediction is well-estimated, we can derive the prediction error estimate in a similar fashion.

$$cl.\varepsilon \approx \frac{\sum_{\{s|cl.C \text{ matches } s\}} p(s)p(cl.A|s)(|cl.R - \rho(s, cl.A)|)}{\sum_{\{s|cl.C \text{ matches } s\}} p(s)p(cl.A|s)} \tag{3.10}$$

Thus, the reward prediction estimates the mean reward encountered in a problem and the reward prediction error estimates the mean absolute deviation (MAD) from the reward prediction.

In a two-class classification problem which provides 1000 reward if the chosen class was correct and 0 otherwise, we may denote the probability that a particular classifier predicts the correct outcome by p_c. Consequently, the reward prediction $cl.R$ of classifier cl will approximate $1000p_c$. Neglecting the oscillation and consequently setting $cl.R$ equal to $1000p_c$, the following error derivation is possible:

$$\begin{aligned} cl.\varepsilon &= (1000 - cl.R)p_c + cl.R(1 - p_c) = \\ &= 2000(p_c - p_c^2) \end{aligned} \tag{3.11}$$

The formula sums the two cases of executing a correct or wrong action with the respective probabilities. The result is a parabolic function for the error ε that reaches its maximum of 0.5 when $p_c(cl)$ equals 0.5 and is 0 for $p_c(cl) = 0$ and $p_c(cl) = 1$. The function is depicted in Figure 3.2. It should be noted that this parabolic function is very similar to the concept of entropy so that the prediction error estimate can also be regarded as an entropy estimate. The main idea behind this function is that given a 50/50 probability of classifying correctly, the mean absolute error will be on its highest value. The more consistently a classifier classifies problem instances correctly/incorrectly the lower the reward prediction error.

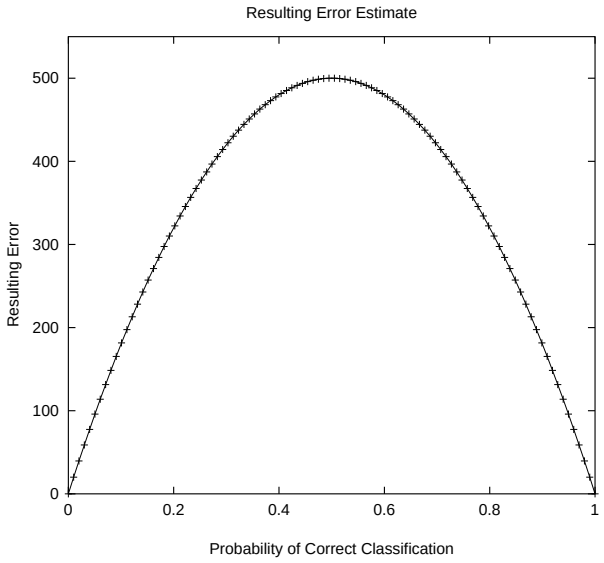


Figure 3.2: Assuming a 1000/0 reward scheme, it can be seen that the prediction error estimate peaks at a probability of a correct classification of 0.5. Given that this is the probability of a completely general classifier, the more probable a classification is correct or wrong, the lower the error.

3.1.4 Rule Evolution

Besides the aforementioned covering mechanism that ensures that all actions in a particular problem instance are represented by at least one classifier, XCS applies a GA for rule evolution. Genetic reproduction is invoked in the current action set $[A]$ if the average time since the last GA application (stored in parameter ts) upon the classifiers in $[A]$ exceeds threshold θ_{GA}.

The GA selects two parental classifiers using proportionate selection where the probability of selecting classifier cl ($p_s(cl)$) is determined by its relative fitness in $[A]$, i.e. $p_s(cl) = F(cl)/\sum_{c\in[A]} F(c)$). Two offspring are generated reproducing the parents and applying crossover and mutation. Parents stay in the population competing with their offspring. Usually, we apply *free mutation* in which each attribute of the offspring condition is mutated to the other two possibilities with equal probability. Another option is to apply *niche mutation* that assures that the mutated classifier still matches the current problem instance.

Offspring parameters are initialized by setting prediction R, ε, F, and as to the parental values. Fitness F is decreased to 10% of the parental fitness being pessimistic about the offspring's fitness. Experience counter exp and numerosity num are set to one.

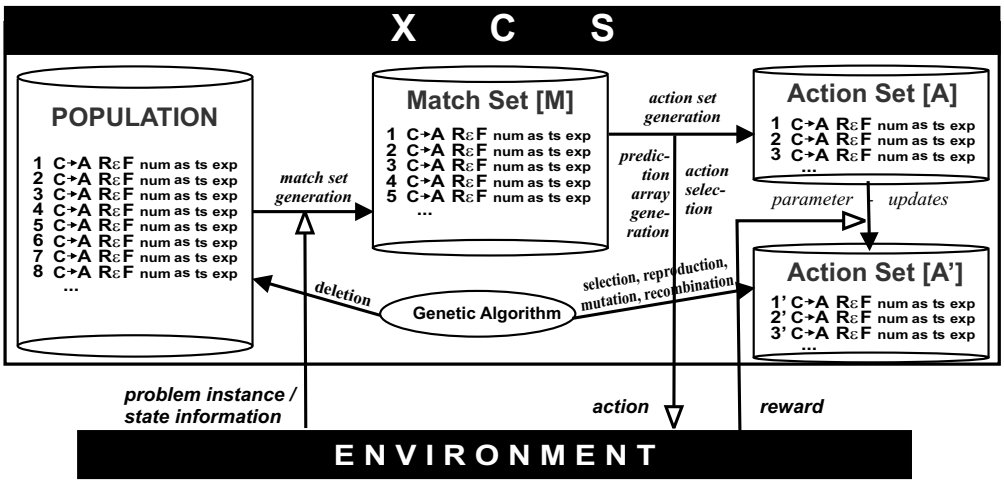


Figure 3.3: A learning iteration in XCS differs from that of a simple LCS in that the GA selects classifiers for reproduction in the current action set based on classifier fitness, which reflects the moving average set-relative classifier accuracy.

In the insertion process, *subsumption deletion* may be applied (Wilson, 1998) to stress generalization. Due to the possible strong effects of *action set subsumption* we apply *GA subsumption* only. *GA subsumption* checks offspring classifiers to see whether their conditions are logically subsumed by the condition of another *accurate* ($\varepsilon < \varepsilon_0$) and *sufficiently experienced* ($exp > \theta_{sub}$) classifier in $[A]$. If an offspring is subsumed, it is not inserted in the population but the subsumer's numerosity is increased.

The population of classifiers $[P]$ is of maximum size N. Excess classifiers are deleted from $[P]$ with probability proportional to the action set size estimate as that the classifiers occur in. If the classifier is sufficiently experienced $exp > \theta_{del}$ and its fitness F is significantly lower than the average fitness $\overline{F}$ of classifiers in $[P]$ ($F < \delta\overline{F}$), its deletion probability is further increased by the factor $\overline{F}/F$.

3.1.5 XCS Learning Intuition

The overall learning process is illustrated in Figure 3.3. As can be seen, in contrast to the simple LCS, XCS reproduces classifiers selecting from the current action set instead of from the whole population. Several additional classifier parameters and essentially the accuracy-based fitness measure F monitor the performance of the classifier.

XCS strives to predict all possible reward values equally accurately. The amount of reward itself does not bias XCS's learning mechanism. Additionally, XCS tends to evolve a *general* problem solution since reproduction favors classifiers that are frequently active (part

of an action set) whereas deletion deletes from the whole population. Among classifiers that are *accurate* (that is, the error is below ε_0) and semantically equally general (the classifier are active equally often), subsumption deletion causes additional generalization pressure in that a syntactically more general classifier absorbs more specialized offspring classifiers.

Due to the niched reproduction in conjunction with action set size based deletion, the evolving representation is designed to stay complete. Since the action set size estimate decreases in classifiers that currently inhabit underrepresented niches, their probability of deletion decreases resulting in effective niching. Reproduction is dependent on the frequency of occurrence. In effect, XCS is designed to evolve accurate payoff predictions for all problem spaces that occur sufficiently frequently.

Together, the learning processes in XCS are designed to achieve one common goal: to evolve a *complete, maximally accurate, and maximally general* representation of the underlying payoff-map, or Q-value function. This representation was previously termed the optimal solution representation $[O]$ (Kovacs, 1996; Kovacs, 1997).

Before we analyze the XCS system in detail in the subsequent chapters, the next section provides further insights into XCS considering learning and problem representation of XCS in a small classification problem and a small RL problem.

3.2 Simple XCS Applications

In order to better understand XCS's mechanisms this section applies XCS to several small exemplar problems. In addition to the understanding of XCS functioning, this section provides a comparison to RL and the Q-learning approximation method in XCS. First, however, we investigate XCS's performance in a simple classification problem.

3.2.1 Simple Classification Problem

In order to disregard the additional complication of reward back-propagation, that is, the Q term derived in Equation 3.2, a big part of the analyses in the subsequent chapters focuses on classification or one-step problems. Since in classification problems successive problem instances usually do neither depend on each other nor on the chosen classification, each learning iteration can be treated independently.

As our simple example, we use the 6-multiplexer problem (see Appendix C). The multiplexer problem is interesting because the solution can be represented by completely non-overlapping niches and the path to the solution is of interest as well. The optimal XCS

Table 3.1: The optimal population $[O]$ in the 6-multiplexer problem.

Nr.	C	A	R	ε	F	Nr.	C	A	R	ε	F
1	000###	0	1000	0	1	2	000###	1	0	0	1
3	001###	0	0	0	1	4	001###	1	1000	0	1
5	01#0##	0	1000	0	1	6	01#0##	1	0	0	1
7	01#1##	0	0	0	1	8	01#1##	1	1000	0	1
9	10##0#	0	1000	0	1	10	10##0#	1	0	0	1
11	10##1#	0	0	0	1	12	10##1#	1	1000	0	1
13	11###0	0	1000	0	1	14	11###0	1	0	0	1
15	11###1	0	0	0	1	16	11###1	1	1000	0	1

population in the 6-multiplexer is shown in Table 3.1.

It is important to notice that XCS represents both the correct classification and the incorrect classification for each problem subsolution. As mentioned before, XCS is designed to evolve a complete and accurate payoff map of the underlying problem. Thus, XCS represents each subsolution in the 6-multiplexer twice: (1) by specifying the correct condition and action combination; (2) by specifying the incorrect class as an action and predicting accurately ($\varepsilon = 0$) that the specified action is incorrect (resulting always in zero reward $R = 0$).

How might XCS evolve such an optimal population? In general, two ways may lead to success starting either from the over-specific or the overgeneral side. Starting over-specific, the don't care probability $P_\#$ is set low so that initial classifiers nearly specify all l attributes in the problem. Most classifiers then are maximally accurate so that inaccurate classifiers quickly disappear since selection favors accurate classifiers. Mutation mainly results in generalized offspring since mainly specified attributes will be chosen for mutation. If mutation results in an inaccurate, overgeneralized classifier, the accuracy and thus fitness of the offspring is expected to drop and the classifier will have hardly any reproductive events and consequently will soon be deleted. Additionally, since more general classifiers on average will be more often part of an action set, more general, accurate classifiers undergo more reproductive events and thus propagate faster. Thus, the process is expected to evolve the accurate, maximally general solution as the final outcome. Despite this appealing description, we will see that a start from the over-specialized side is usually undesirable because of the large requirements on population size. In essence, when starting completely specialized, the population size needs to be chosen larger than 2^l and thus exponentially in problem length which is obviously a highly undesirable requirement.

Thus, XCS usually starts its search for an optimal solution representation from the over-

Table 3.2: The path to an optimal solution from the overgeneral side in the 6-multiplexer problem.

Nr.	C	A	R	ε	Nr.	C	A	R	ε
1	`######`	0	500.0	500.0	2	`######`	1	500.0	500.0
3	`1#####`	0	500.0	500.0	4	`1#####`	1	500.0	500.0
5	`0#####`	0	500.0	500.0	6	`0#####`	1	500.0	500.0
7	`##1###`	0	375.0	468.8	8	`##1###`	1	625.0	468.8
9	`##0###`	0	625.0	468.8	10	`##0###`	1	375.0	468.8
11	`##11##`	0	250.0	375.0	12	`##11##`	1	750.0	375.0
13	`##00##`	0	250.0	375.0	14	`##00##`	1	750.0	375.0
15	`0#1###`	0	250.0	375.0	16	`0#1###`	1	750.0	375.0
17	`0#0###`	0	750.0	375.0	18	`0#0###`	1	250.0	375.0
19	`0#11##`	0	0.0	0.0	20	`0#11##`	1	1000.0	0.0
21	`001###`	0	0.0	0.0	22	`001###`	1	1000.0	0.0
23	`10##1#`	0	0.0	0.0	24	`10##1#`	1	1000.0	0.0
25	`000###`	0	1000.0	0.0	26	`000###`	1	0.0	0.0
27	`01#0##`	0	1000.0	0.0	28	`01#0##`	1	0.0	0.0

general side. In the most extreme case, the don't care probability may be set to $P_{\#} = 1$ and XCS starts its search with the two completely general classifiers `######`→`0` and `######`→`1`. Mutation is responsible for the introduction of initial more specialized classifiers. However, mutation alone is usually not sufficient since more general classifiers are part of more action sets undergoing more reproductive events on average resulting in an overall generalization pressure. Thus, classifiers need to be propagated that are more accurate. Table 3.2 shows the resulting expected average R and ε values for progressively more specialized classifiers. Fitness is not shown since fitness depends on the classifier distribution in the current population.

Note how initially the specialization of the value bits results in a smaller error and thus a larger accuracy. Once a value bit is specified, the specialization of an address bit or an additional value bit has an equal effect on error. Thus, the evolutionary process is initially guided towards specializing value bits. Soon, however, the beneficial specification of an address bit takes over and completely accurate classifiers evolve. Note that lower error somewhat corresponds to lower entropy in the class distribution of the classifier. Both sides are explored: the more incorrect as well as the more correct classification side.

3.2.2 Performance in Maze1

Compared to a classification problem, an RL problem poses the additional complication of back-propagating reward appropriately. Due to the generalized, distributed representation of the Q-function in XCS, additional complications might arise caused by inappropriate reward propagations from inaccurate, young, or overgeneralized classifiers. Chapter 9 investigates performance in RL problems in much more detail.

In this section, we investigate XCS learning and solution representation in the Maze1 problem shown in Figure 1.2. The question is if XCS can be expected to solve this problem and evolve a complete, but generalized representation of the underlying Q-table.

Lanzi (2002) provides a detailed comparison of XCS with Q-learning from the RL perspective. The investigations show that if no generalization is allowed, XCS essentially mimics the Q-learning mechanism. Each classifier corresponds to exactly one entry in the Q-table (shown in Table 1.2 for the Maze1 case) and the Q-learning update function:

$$Q(s,a) \leftarrow Q(s,a) + \beta(r + \gamma \mathtt{max}_{a'} Q(s',a') - Q(s,a)) \tag{3.12}$$

is equivalent to the XCS update function for the reward prediction

$$R \leftarrow R + \beta(r + \gamma \mathtt{max}_{A \in \mathcal{A}} P^{t+1}(A) - R) \tag{3.13}$$

in that the reward prediction values R coincide with the Q-values $Q(s,a)$ since the condition of a classifier specifies exactly one state s and one action a. Thus, the prediction array coincides with Q-value entries since for each prediction array entry exactly one classifier applies and thus the entry is equivalent to the R value of the classifier (see Equation 3.1) which is equivalent to the Q-value as outlined above. Thus, without generalization, XCS is a tabular Q-learner where each tabular entry is represented by a distinct rule.

What is expected to happen in the case when generalization is allowed? What does the perfect representation look like? The optimal population in the Maze1 problem is shown in Table 3.3. In comparison to the Q-table in Table 1.2, we see that XCS is able to represent the maze in a much more compact form requiring only 19 classifiers to represent a Q-table of size 40. Note how XCS is generalizing over the state space with respect to each action as indicated by the action order in Table 3.3. Essentially, since each state combination is representable with a different action code (for example, the condition of classifier one specifies that it matches in situations A or E), XCS generalizes over states that yield identical Q-values with respect to a specific action. Even in our relatively simple environment in which there are

Table 3.3: The optimal population $[O]$ in the Maze1 problem.

Nr.	C	A	R	ε	Nr.	C	A	R	ε
1	11#####1	↑	810.0	0	2	1#0###0#	↑	900.0	0
3	0#######	↑	1000.0	0					
4	11#####1	↗	810.0	0	5	#10###0#	↗	900.0	0
6	#0######	↗	1000.0	0					
7	##0####1	→	900.0	0	8	11####0#	→	810.0	0
9	11#####1	↘	810.0	0	10	##0###0#	↘	900.0	0
11	11#####1	↓	810.0	0	12	##0###0#	↓	900.0	0
13	11#####1	↙	810.0	0	14	##0###0#	↙	900.0	0
15	1#0####1	←	810.0	0	16	#1####0#	←	900.0	0
17	11#####1	↖	810.0	0	18	##0###01	↖	900.0	0
19	#######0	↖	1000.0	0					

only three different Q-values possible (810, 900, and 1000) generalization is effective. Given a larger state space, further generalizations over different states are expectable. In sum, XCS learns a generalized Q-table representation specializing those attributes that are relevant for the distinction of different Q-values.

The question remains how XCS learns the desired complete, accurate, and maximally problem solution shown in Table 3.3. Dependent on the initialization procedure, classifiers might initially randomly distinguish some states from others. Additionally, the back-propagated reward signal is expected to fluctuate significantly. As in Q-learning, XCS is expected to progressively learn starting from those state-action combination that yield actual reinforcement. Since these transitions result in an exact reward of 1000, classifiers that identify these cases quickly become accurate fast and thus will be reproduced most of the time they are activated. Once, the 1000 reward cases are stably represented, back-propagation becomes more reliable and the next reward level (900) can be learned accurately and so forth. Thus, as in Q-learning, reward will be spread backward starting from the cases that yield actual reward. In chapter 9 we show how XCS's performance can be further improved to ensure a more reliable and stable solution in RL problems applying gradient-based update techniques.

In sum, due to the accuracy-based fitness approach, XCS is expected to evolve a representation that covers the whole problem space of Maze1 yielding classifiers that comprise the maximal number of states in their conditions to predict the Q-value accurately with respect to the specified action. However, we still did not answer how XCS evolves a *maximally general* problem representation. For example, Classifier 12 specifies that if moving south

and if currently located in state B, C, or D, then a reward of 900 is expected. Classifier 12 is *maximally general* in that any further generalization of the classifier's condition will result in an inaccurate prediction (that is, state A or E will be accepted by the condition part in which a south action yields a reward of 810). However, Classifier 12 can be further specialized still matching in all three states B, C, and D. In essence, in all three states the positions to the south-east, south, and south-west are blocked by obstacles so that the specification of obstacles in these positions is redundant. Only subsumption is able to favor the syntactically more general Classifier 12. Thus, due to the sparse problem instance coverage (only five perception possible) of the whole problem space (the coding allows $2^l = 2^8 = 256$ problem instances), rather few semantic generalizations over different states are possible and syntactic generalization is much more important. Thus, subsumption is the major factor to evolve maximally general classifiers. If Maze1 was noisy and syntactic generalization was desired, the error threshold ε_0 may need to be increased to enable subsumption, as suggested in Lanzi (1999c).

3.3 Summary and Conclusions

This chapter introduced the accuracy-based learning classifier system XCS. As all (Michigan-style) LCSs, XCS is an online learner applying RL techniques for rule evaluation and action decisions and GA techniques for rule discovery. In contrast to traditional LCSs, XCS derives classifier fitness from the *accuracy of reward prediction* instead of from the reward prediction value directly. The reward updates can be compared to Q-learning updates. Thus, XCS is designed to learn a generalized representation of the underlying Q-function in the problem.

XCS's *niched reproduction* in combination with population-wide deletion results in an *implicit, semantic generalization pressure* that favors classifiers that are more frequently active. Additional, subsumption deletion favors accurate classifiers that are *syntactically* more general. Niche reproduction in combination with the population-wide deletion based on the action set size estimates results in effective problem niching striving to evolve and maintain a complete problem solution.

In effect, XCS is designed to evolve a *complete, maximally accurate, and maximally general problem solution* represented by a population of classifiers. Each classifier defines a problem subspace in which it predicts the corresponding Q-value accurately.

The application to two small problems showed which solution representation XCS is designed to evolve and how it might be evolved. Starting from the overgeneral side, higher accurate classifiers need to be detected and propagated. Starting from the over-specific

side, generalizations due to mutations boil the representation down to the desired accurate, maximally general one. However, when starting over-specialized the required population size needs to grow exponentially in problem length l given that all problem instances are equally likely to occur.

In problems in which only a few samples are available so that the problem space is covered only sparsely, syntactic generalization and thus subsumption becomes more important since the usual generalization pressure due to more frequent reproductive opportunities may not apply. However, subsumption only works for accurate classifiers so that a proper choice of the error threshold ε_0 and thus an appropriate noise tolerance becomes more important.

While the base XCS system introduced in this section learns a generalized representation of the underlying Q-value function, it should be noted that XCS is not limited to approximating Q-functions. In fact, XCS can be modified yielding a general, online learning *function approximation technique* (Wilson, 2001a). Due the rule-based structure, XCS is designed to partition its space dependent on the representation of classifier conditions. Each classifier then approximates, or predicts, the actual function value in its defined subspace. In the base XCS, conditions define hypercubes as subspaces and predict constant reward values—consequently applying piece-wise constant function approximation. Wilson (2001a) experimented with piecewise linear approximations. Lanzi (1999b) experimented with S-expressions for conditions. Dependent on the problem structure at hand, other condition representations may be applied. Similarly, other prediction methods are imaginable.

Chapter 4

How XCS Works: Ensuring Effective Evolutionary Pressures

The last chapter gave a concise introduction to the accuracy-based XCS classifier system. We saw that XCS is designed to evolve online a complete, maximally accurate, and maximally general representation of the underlying Q-value function. The accuracy-based approach assures that no strong overgenerals are possible since the maximally accurate classifiers receive maximal fitness.

With this knowledge in mind, we now turn to the first aspect of our facetwise LCS theory approach investigating the evolutionary pressures in the XCS classifier system. While strong overgenerals are prevented by the accuracy-based fitness approach, it still needs to be assured that fitness guides towards the intended solution. Second, parameter initialization and estimation needs to be most effective. Third, appropriate generalization needs to apply so that the solution becomes maximally general.

To investigate these points in the XCS system, we undertake a general analysis of all *evolutionary pressures* in XCS. Evolutionary pressures can be regarded as evolutionary biases that influence or bias learning in XCS. Often, the pressures influence *specificity* in the classifier population. Specificity was defined in Chapter 2. It essentially characterizes how restricted a classifier condition is. The less space a classifier condition covers, the more specific it is.

The pressure analysis quantifies the generalization in XCS as well as the influence of mutation leading to an equilibrium in population specificity if no fitness pressure applies. This is expressed in the *specificity equation*, which we evaluate in detail. With the addition of fitness pressure and subsumption, the equilibrium lies exactly at the point of the desired maximally accurate and maximally general problem solution.

However, in order to ensure that fitness guides to the equilibrium, the generalization

pressure needs to be overcome. This is assured by an appropriate selection mechanism leading us to the introduction of *tournament selection* for offspring selection. We show that tournament selection assures sufficiently strong fitness guidance and consequently makes XCS much more parameter as well as noise independent.

The next section introduces all evolutionary pressures, analyzes them in separation, and combines them in one general specificity equation. The results are experimentally validated in binary classification problems. Next, tournament selection is introduced and evaluated. The investigations prepare XCS to face the subsequent challenges proposed by our facetwise theory.

4.1 Evolutionary Pressures in XCS

Previous publications have considered the influence of fitness guidance and generalization. The principles underlying evolution in XCS were originally outlined in Wilson's *generalization hypothesis* Wilson (1995), which suggests that classifiers in XCS become maximally general due to niche-based reproduction in combination with population-wide deletion. Kovacs (1996) extended Wilson's explanation to an *optimality hypothesis*, supported experimentally in small multiplexer problems, in which he argued that XCS develops minimal representations of optimal solutions (that is, the optimal solution representation $[O]$). Later, Wilson (1998) suggested that XCS scales polynomially in problem complexity and thus machine learning competitive. Kovacs and Kerber (2001) proposed to relate problem complexity directly to the size of the optimal solution $|[O]|$.

Despite these insights, Wilson's generalization hypothesis remained to be theoretically validated and quantified. In this section, we investigate Wilson's hypothesis developing a fundamental theory of XCS generalization and learning. To avoid the additional complication of back-propagating reward in RL problems, we focus on XCS's performance in classification problems.

In particular, we analyze all evolutionary pressures present in XCS.[1] An evolutionary pressures refers to a learning bias in XCS influencing the population structure, often with respect to specificity. The average *specificity* in a population, denoted by $\sigma[P]$, refers to the

[1]Related publications of parts of this section can be found elsewhere (Butz & Pelikan, 2001; Butz, Goldberg, & Tharakunnel, 2003; Butz, Kovacs, Lanzi, & Wilson, 2004).

average specificity of all classifiers in the population. That is,

$$\sigma[P] = \frac{\sum_{c \in [P]} \sigma(c) \cdot c.num}{\sum_{c \in [P]} c.num}, \tag{4.1}$$

where $\sigma(c)$ refers to the specificity of classifier c. We defined specificity for the binary case in Chapter 2 as the fraction of specified attributes (zero or one) in the condition part of a classifier.

Our analysis distinguishes between the following evolutionary pressures:

1. *Set pressure*, which quantifies Wilson's generalization hypothesis;
2. *Mutation pressure*, which quantifies the influence of mutation on specificity;
3. *Deletion pressure*, which qualifies additional deletion influences;
4. *Fitness pressure*, which qualifies the accuracy-based fitness influence;
5. *Subsumption pressure*, which qualifies the exact influence of subsumption deletion.

Set pressure and mutation pressure are combined to a general *specificity equation* that is evaluated in several experiments progressively increasing the influence of deletion and fitness pressure.

4.1.1 Generalization due to Set Pressure

The basic idea behind the set pressure is that XCS reproduces classifiers in action sets $[A]$ whereas it deletes classifiers from the whole population $[P]$. The set pressure is a combination of the selection pressure produced by the GA applied in $[A]$ and the pressure produced by deletion applied in $[P]$. It was originally qualitatively proposed in Wilson (1995) and further experimentally analyzed in Kovacs (1997).

The generalization hypothesis argues that since more general classifiers appear more often in action sets $[A]$, they undergo more reproductive events. Combined with deletion from $[P]$, the result is an intrinsic tendency towards generality favoring more general classifiers. Classifiers in this respect are *semantically* more general in that they are part of an action set more frequently. Classifiers that are equally often part of an action set but may be distinguished by syntactic generality are not affected by the set pressure.

To formalize the set pressure, we determine the expected specificity $\sigma[A]$ of classifiers in an action set $[A]$ with respect to the current expected specificity $\sigma[P]$ of the classifiers in

population $[P]$. The specificity of the initial random population $\sigma[P]$ is directly correlated with the don't-care probability $P_\#$, i.e., $\sigma[P] = 1 - P_\#$. For our calculations, we assume a binomial specificity distribution in the population. This is essentially the case in a randomly generated population as well as in a population in which selection and deletion is random and genetic operators do not change the distribution. With the assumption of a binomial distribution in the population, we can determine the probability that a randomly chosen classifier in the population $cl \in [P]$ has specificity k/l as follows:

$$P(\sigma(cl) = k/l) = \binom{l}{k} \sigma[P]^k (1 - \sigma[P])^{l-k}, \tag{4.2}$$

where cl is a classifier; l is the length of classifier conditions; k is the number of *specified bits* in the condition, i.e., number of bits different from a don't-care symbol. The equation essentially is able to estimate the proportion of different specificities in a population with average specificity $\sigma[P]$.

The probability that a classifier cl matches a certain input s depends on its specificity $\sigma(cl)$. To match, a classifier cl with specificity k/l must match all k specific bits. This event has probability 0.5^k since each specific attribute matches with probability 0.5. Therefore, the proportion of classifiers in [P] with a specificity k/l that match in a specific situation is as follows:

$$\begin{aligned} &P(cl \; matches \wedge \sigma(cl) = k/l) = \\ &= P(\sigma(cl) = k/l) P(cl \; matches | \sigma(cl) = k/l) = \\ &= P(\sigma(cl) = k/l) \, 0.5^k = \binom{l}{k} \left(\frac{\sigma[P]}{2}\right)^k (1 - \sigma[P])^{l-k} \end{aligned} \tag{4.3}$$

To derive a specificity $\sigma[M]$ of a match set $[M]$, it is first necessary to specify the proportion of classifiers in $[M]$ with specificity k/l given the population specificity $\sigma[P]$. This proportion can be derived as follows:

$$\begin{aligned} P(\sigma[M] = k/l | \sigma[P]) &= \frac{P(cl \; matches \wedge \sigma(cl) = k/l)}{\sum_{i=0}^{l} P(cl \text{ matches} \wedge \sigma(cl) = i/l)} = \\ &= \frac{\binom{l}{k} \left(\frac{\sigma[P]}{2}\right)^k (1 - \sigma[P])^{l-k}}{\sum_{i=0}^{l} \binom{l}{i} \left(\frac{\sigma[P]}{2}\right)^i (1 - \sigma[P])^{l-i}} = \frac{\binom{l}{k} \left(\frac{\sigma[P]}{2}\right)^k (1 - \sigma[P])^{l-k}}{\left(1 - \frac{\sigma[P]}{2}\right)^l} = \\ &= \binom{l}{k} \left(\frac{\sigma[P]}{2 - \sigma[P]}\right)^k \left(1 - \frac{\sigma[P]}{2 - \sigma[P]}\right)^{l-k} \end{aligned} \tag{4.4}$$

To compute $\sigma[M]$ we multiply actual specificity values, k/l, by the proportions $P(\sigma[M] = k/l|\sigma[P])$ and sum up the values to derive the resulting specificity of $[M]$. Since the action set $[A]$ has on average the same specificity as the match set $[M]$ ($\sigma[A] \approx \sigma[M]$), $\sigma[A]$ can be derived as follows:

$$\begin{aligned}
\sigma[A] &\approx \sigma[M] = \\
&= \sum_{k=0}^{l} \frac{k}{l} P(\sigma[M] = k/l|\sigma[P]) = \\
&= \sum_{k=1}^{l} \frac{k}{l} \binom{l}{k} \left(\frac{\sigma[P]}{2-\sigma[P]}\right)^k \left(1 - \frac{\sigma[P]}{2-\sigma[P]}\right)^{l-k} = \\
&= \sum_{k=1}^{l} \binom{l-1}{k-1} \left(\frac{\sigma[P]}{2-\sigma[P]}\right)^k \left(1 - \frac{\sigma[P]}{2-\sigma[P]}\right)^{l-k} = \\
&= \frac{\sigma[P]}{2-\sigma[P]} \sum_{j=0}^{l-1} \binom{l-1}{j} \left(\frac{\sigma[P]}{2-\sigma[P]}\right)^j \left(1 - \frac{\sigma[P]}{2-\sigma[P]}\right)^{l-1-j} = \\
&= \frac{\sigma[P]}{2-\sigma[P]}
\end{aligned} \tag{4.5}$$

The equation can be used to determine the average expected specificity $\sigma[A]$ in an action set $[A]$ assuming a binomially distributed specificity with mean $\sigma[P]$ in the population.

Figure 4.1 depicts Equation (4.5). Except at the extreme points, the specificity of $[A]$ is always smaller than the specificity of $[P]$. Thus since selection takes place in the action sets but deletion occurs in the population as a whole, there is a tendency for the generality of the population to increase—in line with Wilson's generalization hypothesis. In the absence of fitness pressure, the equation provides an estimate of the difference in specificity of selected and deleted classifiers. Equation (4.5) is enhanced below accounting for mutation as well. It is experimentally validated in Section 4.2.

4.1.2 Mutation's Influence

Although usually only a low mutation probability is applied, mutation still influences specificity. In the absence of other evolutionary influences, mutation pushes the population towards a certain proportion of zeros, ones, and don't-cares. As outlined in Chapter 2, free mutation pushes towards a distribution of 1:2 general:specific. *Niche mutation*, which mutates a specified attribute always to a don't care and a don't care always to the current value of the respective attribute, pushes towards a distribution of 1:1 general:specific.

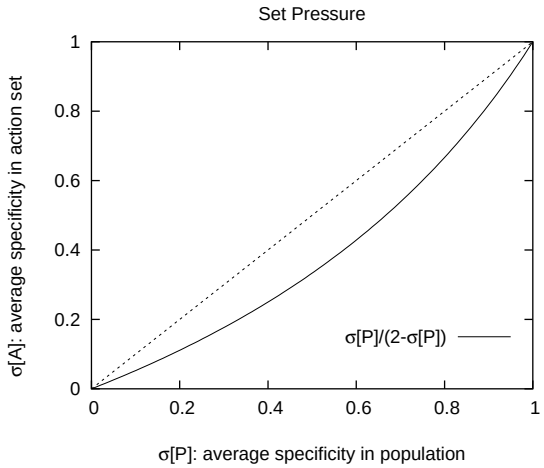


Figure 4.1: Except at the extreme points of zero or one specificity, the expected average specificity of action sets $\sigma[A]$ is always smaller than that of the current population.

The average expected change in specificity between the parental classifier c_p and the mutated offspring classifier c_o for the niche mutation case can be written as follows:

$$\begin{aligned}\Delta_{mn}(\sigma(c_p)) &= \sigma(c_o) - \sigma(c_p) = \\ &= \sigma(c_p)(1-\mu) + (1-\sigma(c_p))\mu - \sigma(c_p) = \\ &= \mu(1-2\sigma(c_p)) \end{aligned} \tag{4.6}$$

and for free mutation as

$$\begin{aligned}\Delta_{mf}(\sigma(c_p)) &= \sigma(c_o) - \sigma(c_p) = \\ &= \sigma(c_p)(1-\mu/2) + (1-\sigma(c_p))\mu - \sigma(c_p) = \\ &= 0.5\mu(2-3\sigma(c_p)). \end{aligned} \tag{4.7}$$

As expected, the increase in specificity is higher when free mutation is applied given low parental specificity ($\sigma(c_p) < 1/2$) and specificity decrease is lower when parental specificity is high ($\sigma(c_p) > 2/3$). Applying random selection, mutation, and random deletion, mutation pushes the population towards a specificity of 0.5 applying niche mutation and 0.66 applying free mutation. The current intensity of the pressure depends on the mutation type, the current parental specificity, and on the frequency of the GA application (influenced by the parameter θ_{GA}).

4.1.3 Deletion Pressure

The probability of a classifier being deleted depends on its action set size estimate as and (depending on classifier experience) its fitness F. Due to the resulting bias towards deleting classifiers that occupy larger action sets, deletion pushes the population towards an equal distribution of classifiers in each environmental niche. With respect to specificity, classifier selection for deletion from $[P]$ is essentially random and there is no particular deletion pressure for or against general classifiers. In the absence of other biases the average expected specificity of deleted classifiers is equal to the average specificity in the population $\sigma[P]$.

A more significant effect can be observed with respect to overlapping niches. Given there are two non-overlapping, accurate niches and another accurate niche that overlaps with either one of the former, the action set size estimate of the overlapping niche will be larger than that of the non-overlapping ones. For example, given the non-overlapping niches `000***` and `01*0**` and the overlapping niche `0*00**` (this is actually the case in the multiplexer problem), and given further that all niches are represented by accurate, maximally general classifiers with a numerosity of say ten, then, given there are no further overlaps, the action set size estimate of the overlapping classifier will stay on 20 whereas the estimate of the non-overlapping ones will approximate 15 making the deletion of the overlapping classifier more likely. Thus, apart from emphasizing equal niche support, the action-set size estimate based deletion pushes the population towards a non-overlapping solution representation.

4.1.4 Subsumption Pressure

Subsumption deletion applies only to classifiers that are accurate ($\varepsilon < \varepsilon_0$) and sufficiently experiences ($exp > \theta_{sub}$). The accuracy requirement suggests that the problem is less noisy than ε since otherwise classifiers are not expected to satisfy the criterion ever.

If accurate classifiers evolve, subsumption deletion pushes towards maximal *syntactic* generality in contrast to the set pressure which pushes towards *semantic* generality. *GA subsumption deletion* prevents the insertion of offspring into $[P]$, if there is a classifier in $[A]$ that is more general than the generated offspring. Thus, once an accurate, maximally general solution was found for a particular niche, no accurate, more specialized classifier will be inserted anymore disabling any specialization in the current niche.

Action set subsumption is stronger than GA subsumption since it allows an accurate more general classifier in an action set to eliminate all classifiers in the set that are more specific. Effectively, once an accurate, maximally general solution was found every more specialized solution is eliminated (increasing the numerosity of the accurate maximally general classifier).

Thus, action set subsumption can be highly disruptive, if an overgeneral classifier becomes temporary accurate. It is not applied in the remainder of this thesis.

To summarize, subsumption pressure is an additional pressure towards accurate, maximally syntactically general classifiers from the over-specific side. It applies only when accurate classifiers are found. Thus, subsumption pressure is helpful mainly later in the learning process once accurate classifiers are found. It usually results in a strong decrease of population size focusing on maximally general classifiers.

4.1.5 Fitness Pressure

Until now we have not considered the effect of fitness pressure which can influence several other pressures. Fitness pressure is highly dependent on the particular problem being studied and is therefore difficult to formalize. In general, fitness results in a pressure which pushes $[P]$ from the overgeneral side towards accurate classifiers. Late in the run, when the optimal solution is mainly found, it prevents overgeneralization.

Additionally, as in the case of subsumption, fitness pushes the population towards a non-overlapping problem representation. Since fitness is derived from the relative accuracy, the accuracy-share is lower in classifiers that overlap with many other accurate classifiers. In effect, unnecessary, overlapping classifiers have a lower fitness on average, are thus less likely to reproduce, and are thus likely to be deleted from the population.

In terms of specificity, fitness pressure towards higher accuracy results usually in a specialization pressure since higher specificity usually implies higher accuracy. Certain problems, however, may mislead fitness guidance in that more general classifiers may actually have higher accuracy. This is particularly the case in problems with unbalanced class distributions and multiple classes as analyzed in Butz, Goldberg, and Tharakunnel (2003) as well as in Bernadó-Mansilla and Garrell-Guiu (2003). Since the problem is not as severe as originally suspected, we do not investigate it any further in this thesis. As suggested by our facetwise theory, though, fitness guidance needs to be ensured.

In sum, fitness pressure usually works somewhat in the opposite direction (towards higher specificity) of the set pressure. Thus, given fitness pressure in a problem, the specificity in the population is expected to decrease less or even to increase dependent on the amount of fitness pressure. Fitness pressure is certainly highly dependent on the investigated problem and thus hard to quantify. The following section combines the pressure influences into one specificity equation.

4.1.6 Pressure Interaction

We now combine the above evolutionary pressures and analyze their interaction. Initially, we consider the interaction of set pressure, mutation pressure, and deletion pressure which yields an important relationship we call the *specificity equation*. Next, we consider the effect of subsumption pressure and potential fitness influences. Finally, we provide a visualization of the interaction of all the pressures. The analyses are experimentally evaluated in Section 4.2.

Specificity Equation

Since mutation is only dependent on the specificity of the selected parental classifier and deletion can be assumed to be random, selection and mutation can be combined into one *specificity equation*. Essentially, set pressure generalizes whereas mutation generalizes or specializes dependent on the specificity of the currently selected classifier.

Since fitness pressure is highly problem dependent, we disregard fitness influences essentially assuming equal fitness of all classifiers in our analysis. As shown later in Section 4.2, this assumption *holds* when all classifiers are accurate and nearly holds when all are similarly inaccurate. Despite the fitness equality assumption, deletion is also dependent on the action set size estimate *as* of a classifier. However, in accordance with Kovacs' insight on the relatively small influence of this dependence (Kovacs, 1999), we assume a random deletion from the population in our formulation. Thus, as stated above, a deletion results on average in the deletion of a classifier with a specificity equal to the specificity of the population $\sigma[P]$. The generation of an offspring, on the other hand, results in the insertion of a classifier with an average specificity of $\sigma[A] + \Delta_{mx}$ $(x \in f, n)$ dependent on the type of mutation used. Putting the observations together we can calculate the average specificity of the resulting population after one time step:

$$\sigma[P(t+1)] \quad = \quad \sigma[P(t)] + f_{GA}\frac{2(\sigma[A] + \Delta_{mx}(\sigma[A]) - \sigma[P(t)])}{N} \tag{4.8}$$

The parameter f_{GA} denotes the frequency of a GA application per time step assuming a constant application frequency for now. The formula adds to the current specificity in the population $\sigma[P(t)]$ the expected change in specificity calculated as the difference between the specificity of the two reproduced and mutated classifiers, that is, $\sigma[A] + \Delta_{mx}(\sigma[A])$ and $\sigma[P(t)]$. Note that although the frequency f_{GA} is written as a constant in the equation, f_{GA} actually depends on $\sigma[P(t)]$, as well as on the specificity distribution in the population. Thus, f_{GA} cannot be written as a constant in general. However, by setting θ_{GA} to zero, it is

possible to force f_{GA} to be one since the average time since the last application of the GA in an action set will always be at least one.

XCS's tendency towards accurate, maximally general classifiers from the overgeneral side is not dependent on the use of subsumption. However, subsumption is helpful in focusing the population more on the maximally general representation. In fact, although the set pressure pushes the population towards more general classifiers once all are accurate the pressure is somewhat limited. Equation 4.8 shows that without subsumption the complete convergence of the population towards maximally accurate, maximally general classifiers is not assured. However, XCS is an online learning system that should be flexible with respect to problem dynamics so that complete convergence is usually not desired.

Another reason for a potential lack of complete convergence can be that the set pressure is not present at all. This can happen, if the state space of a problem is a proper subspace of all possible representable states $\{0,1\}^l$ (as is essentially the case in most datamining applications as well as in RL problems). Subsumption can be helpful in generalizing the population further.

As mentioned above, fitness pushes towards higher specificity from the overgeneral side. Equation 4.8 assumes that the selected parental classifier has an expected average specificity of $\sigma[A]$ effectively assuming random selection in the action set. However, selection is biased towards the selection of more accurate classifiers. In effect, $\sigma[A]$ needs to be replaced by the expected offspring specificity that depends on the expected fitness distribution in the action set. Since this distribution is not only dependent on the problem but also on the selection method used and the current specificity distribution in each action set, we won't analyze the fitness influence any further. However, it should be kept in mind that fitness influences are expected to cause a specialization pressure that diminishes the generalization effect of the set pressure.

All Pressures

The interaction of all the pressures is illustrated in Figure 4.2. In particular, the fitness pressure pushes $[P]$ towards more accurate classifiers; the set pressure pushes $[P]$ towards more general classifiers; the subsumption pressure pushes $[P]$ towards classifiers that are accurate and syntactically maximally general; the mutation pressure pushes towards a fixed proportion of symbols in classifier conditions. Deletion pressure is implicitly included in the notion of set pressure. More detailed effects of deletion are not depicted. Overall, these pressures lead the population towards a population of accurate maximally general classifiers.

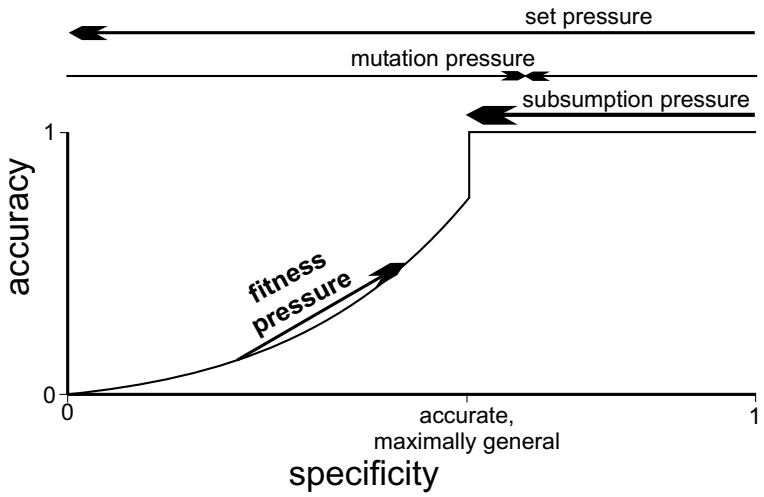


Figure 4.2: The interaction of all evolutionary pressures in XCS on imaginary specificity and accuracy axes.

While set pressure and mutation pressure (free mutation is represented) are independent of classifier accuracy, subsumption pressure and of course fitness pressure are influenced by accuracy.

4.1.7 Steady State Specificity Distribution

From Equation 4.8 it is possible to derive the steady-state specificity the population is expected to converge to (regardless of the initial specificity) assuming no fitness influence. Setting the difference $(\sigma[A]) + \Delta_{mut} - (\sigma[P])$ to zero, we derive

$$\begin{aligned} \sigma[A] + \Delta_{\mu_f} &= \sigma[P] \\ \frac{\sigma[P]}{2-\sigma[P]} + \frac{\mu}{2}\left(2 - 3\frac{\sigma[P]}{2-\sigma[P]}\right) &= \sigma[P], \end{aligned} \tag{4.9}$$

solving for $\sigma[P]$:

$$\begin{aligned} \sigma[P]^2 - (2.5\mu + 1)\sigma[P] + 2\mu &= 0 \\ \sigma[P] &= \frac{1 + 2.5\mu - \sqrt{6.25\mu^2 - 3\mu + 1}}{2}, \end{aligned} \tag{4.10}$$

Table 4.1: Converged specificities in theory and in empirical results in a random function.

μ	0.02	0.04	0.06	0.08	0.10	0.12	0.14	0.16	0.18	0.20
$\sigma[P]$, theory	0.040	0.078	0.116	0.153	0.188	0.223	0.256	0.288	0.318	0.347
$\sigma[P]$, empirical	0.053	0.100	0.160	0.240	0.287	0.310	0.329	0.350	0.367	0.394

Table 4.2: Mutation settings for desired specificities

$\sigma[P]$	0.1	0.2	0.3	0.4	0.5	0.6	0.7
μ	0.051	0.107	0.168	0.240	0.333	0.480	0.840

solving for μ:

$$\mu\left(2-\frac{5}{2}\sigma[P]\right)=\sigma[P]-\sigma[P]^2$$
$$\mu=\frac{\sigma[P]-\sigma[P]^2}{2-\frac{5}{2}\sigma[P]}. \tag{4.11}$$

Applying the above two equations enables us to determine the expected specificity in the population given a fixed mutation probability. On the other hand, given a desired specificity in the population, we can determine the mutation probability necessary to achieve that specificity.

Table 4.1 shows the resulting specificities in theory and empirically determined on a completely randomized Boolean function (a Boolean function that returns uniformly randomly either zero or one for each problem instance). We note that the resulting specificity values can be roughly approximated by twice the value of mutation. The empirical results actually reveal slightly higher values than determined. The generalization study shown below investigates this effect in further detail. The effect is mainly due to the offspring initialization mechanism in conjunction with the application frequency of classifiers.

Table 4.2 shows necessary mutation rates for desired specificities. In practice, slightly lower mutation rates should actually lead to the desired specificities due to the fitness influences. Note that apart from its effect on specificity, mutation may have an additional negative effect when set too high basically randomizing the offspring structure.

4.2 Validation of the Specificity Equation

We now present a set of experiments to validate the influences of the evolutionary pressures identified in the previous section. In particular, we validate the specificity equation, formulated in Equation (4.8), which summarizes the effect of the three main evolutionary pressures in XCS: set pressure, mutation pressure, and deletion pressure.

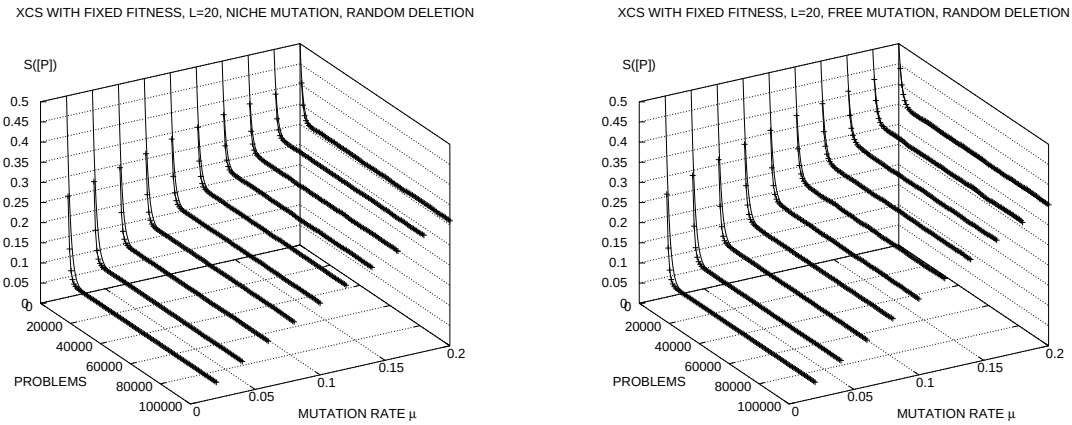
We apply XCS to Boolean strings of length $l = 20$ with different settings. The following figures show runs with mutation rates varying from 0.02 to 0.20. In each plot, solid lines denote the result from Equation (4.8); while dotted lines represent the result of actual XCS runs. Curves are averages over 50 runs. If not stated differently, the population is initially filled up with random classifiers with don't-care probability $P_{\#} = 0.5$. *Niche mutation* is applied. The other XCS parameters are set as follows: $N = 2000$; $\beta = 0.2$; $\alpha = 0.1$; $\varepsilon_0 = 10$; $\nu = 5$; $\theta_{GA} = 0$; $\chi = 0.8$, $\theta_{del} = 20$; $\delta = 0.1$; and $\theta_{sub} = \infty$. Note that the discount factor γ is irrelevant here since these are classification or single-step problems. Since this section is concerned with the set pressure, subsumption is turned off to prevent the additional generalization effect due to the subsumption pressure.

4.2.1 Fixed Fitness

We begin the validation of Equation (4.8) examining runs in which neither fitness pressure nor deletion pressure are present. Fitness pressure as well as deletion pressure are eliminated by selecting classifiers for reproduction at random in the given action set. Classifiers are deleted at random from the population solely considering numerosity num. With these settings, we investigate the influence of (i) free mutation, (ii) niche mutation, (iii) the GA threshold θ_{GA}, and (iv) the initialization of the population.

Niche Mutation. Figure 4.3a depicts the specificity $\sigma[P]$ of the population $[P]$ when the fitness is fixed, deletion is random, and niche mutation is used. As the plot in Figure 4.3a shows, the runs match very closely to the model expressed in Equation (4.8). The initial specificity of 0.5 drops off quickly in the beginning due to the strong set pressure. However, soon the effect of the mutation pressure becomes visible and the specificity in the population converges as predicted. Furthermore we note that, the higher the mutation rate μ, the stronger the influence of mutation, which is manifested in the higher convergence value in the curves with higher μ.

Free Mutation. Figure 4.3b depicts the specificity of the population [P] when free mutation is used. Besides the visibility of the mutation pressure due to the variation of μ, Figure 4.3b reveals that free mutation causes a slightly stronger influence on specificity as formulated in Equation (4.7). When directly comparing Figures 4.3a and 4.3b we note that the higher the parameter μ, the higher the influence of mutation pressure and thus the higher the differences in specificity due to the different mutation types.

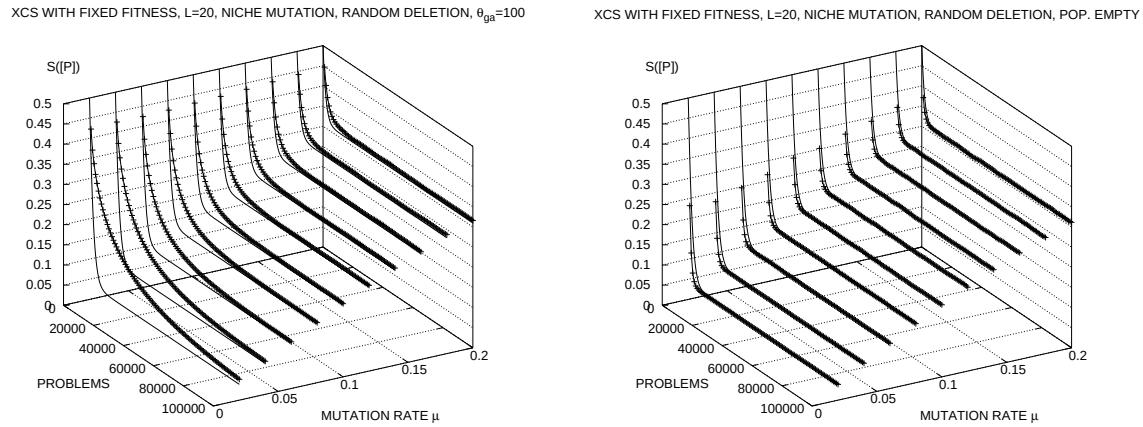


(a) (b)

Figure 4.3: Solid lines represent the specificity as predicted in Equation (4.8). Marked lines represent the actual experimental specificity in [P]. Without the fitness influence, the actual specificity behaves nearly exactly as predicted by the model applying either (a) niche mutation or (b) free mutation.

θ_{GA} **Threshold.** As noted above, the GA frequency f_{GA} in Equation (4.8) should not be written as a constant value, since it actually depends on the specificity $\sigma[P(t)]$ of the population. However, the GA frequency f_{GA} equals one when the GA threshold θ_{GA} is set to one. When setting θ_{GA} to a higher value (100), Figure 4.4a reveals the lower GA frequency effect. Once the specificity in the population has dropped, the action set sizes increase since more classifiers match a specific state. Consequently, more classifiers take part in a GA application, more time stamps *ts* are updated, the average time since the last GA application in the population and in the action sets decrease, and finally the GA frequency decreases. The decrease is observable in the slower specificity decrease. However, as predicted by Equation (4.8), despite its dependence on the actual specificity, f_{GA} does not influence the convergence value.

Initialization of [P]. Until now, we initialized the population with random classifiers. This hypothesis assures a perfect binomial specificity distribution in the beginning of the run. However, the hypothesis of an initial random population appears not to be strictly necessary. Figure 4.4b reports runs in which this hypothesis is relaxed. The population is initially empty and first classifiers are generated by covering. The only noticeable effect in Figure 4.4b in comparison to Figure 4.3a is that in the very beginning of a run the specificity drops off slightly faster than in the case of an initial random population. This is expectable,



(a) (b)

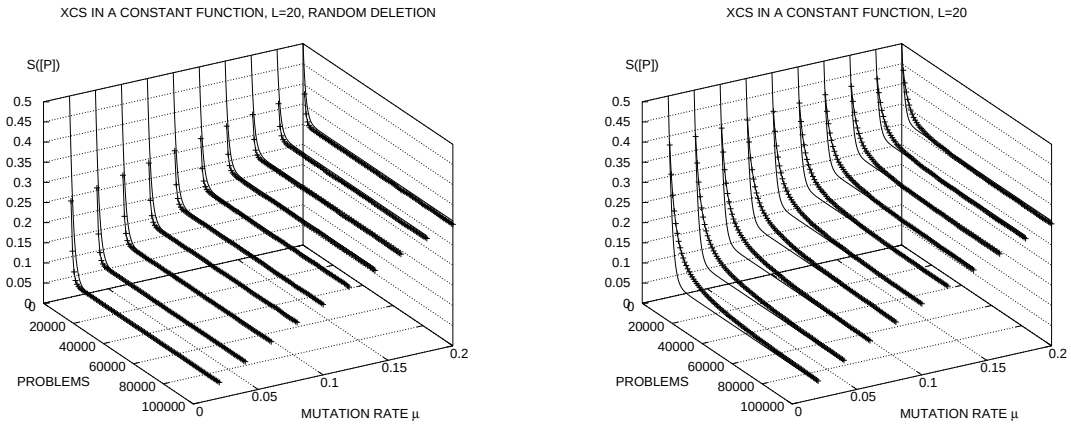
Figure 4.4: (a) When applying a GA threshold of $\theta_{GA} = 100$, the GA frequency and consequently the specificity pressure decreases. Convergence values are not influenced. (b) With an initially empty population, the specificity drops off slightly faster in the beginning.

since the population does not contain 2000 classifiers initially. Thus, the specificity pressure is initially stronger as also observable in Equation (4.8) when N is initially smaller than 2000.

4.2.2 Constant Function

While fitness influence was intentionally omitted in the previous experiments, we now study the actual influence of fitness on $\sigma[P]$. In this section, we apply XCS to a constant Boolean function which always returns a reward of 1000. With these settings, all classifiers turn out to be accurate since their prediction error is always zero. Note however that a zero prediction error does not necessarily mean constant fitness values. In fact, since fitness is determined by the classifier's *relative accuracy*, fitness should still have an influence on evolutionary pressure. Figure 4.5a reports runs with random deletion. It can be seen that the assumption of a binomial distribution indeed also holds later in the run—or is at least not too harsh—since the specificity behaves exactly as predicted.

The behavior of $\sigma[P]$ changes, though, when we delete proportional to the action set size estimate parameter *as* (as done in XCS). Figure 4.5b reports runs in which the population is initially empty and the usual deletion is used. Note that in Figure 4.5b the slopes of the curves decrease in comparison to the ones in Figure 4.5a. In the end, though, specificity of $[P]$ converges to the value predicted by the theory. The difference can only be the result of



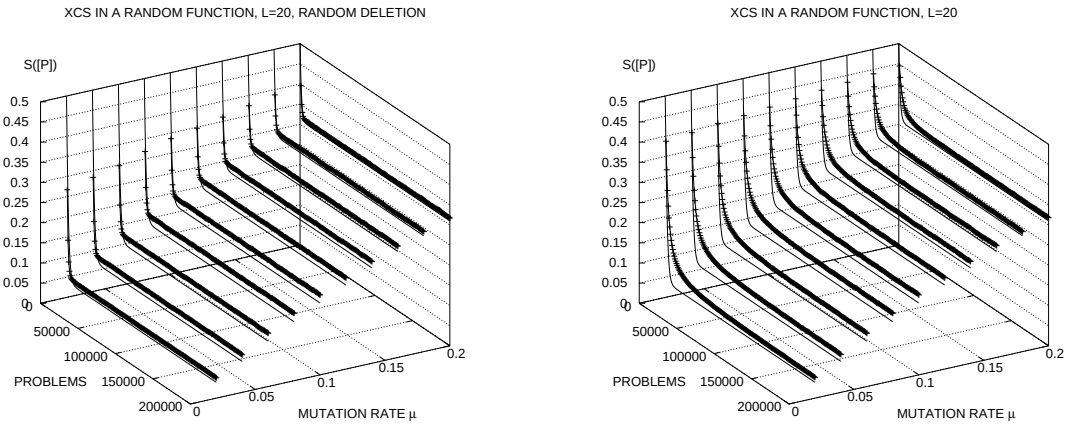
(a) (b)

Figure 4.5: (a) When applied to a constant function with random deletion, the changing specificity still matches the proposed theory. (b) Due to slow adaptation of the action set size estimate parameter, specificity convergence takes longer when action set size estimate based deletion is applied.

the bias in the deletion method of deleting classifiers with larger action set size estimates as. As the specificity of $[P]$ decreases, the action set size increases as noted before. Thus, since more general classifiers are more often present in action sets, their action set size estimate as is more sensitive to the change in the action set size and consequently, it is larger in more general classifiers while specificity drops. Eventually, all as values will have adjusted to the change and the predicted convergence value is met. This explanation is further confirmed by the fact that the difference between the actual runs and the curves given by Equation 4.8 become smaller and converge faster for higher mutation rates μ since the specificity slope is not as steep as in the curves with lower μ values.

4.2.3 Random Function

The results in the previous section show that the influence of the fitness in XCS with a constant function is rather small. Accordingly, we now apply XCS with the two different deletion strategies to a much more challenging problem, that is, to a random Boolean function which randomly returns rewards of 1000 or 0. Figure 4.6a reports the runs in which XCS with random deletion is applied to the random function. The experiments show that in the case of a random function the fitness influences the specificity slope as well as the convergence value. In fact, the convergence value is larger than that predicted by the model in Equation 4.8. Two factors cause this effect. (i) The high variance in less experienced classifiers and (ii) the



(a) (b)

Figure 4.6: (a) Applied to a random function, the specificity stays on a higher level due to the higher error variance in more specialized classifiers and parameter initialization effects. (b) Fitness-biased deletion decreases convergence of specificity due to the discussed action set size estimate influence.

parameter initialization technique.

Since the possible rewards are 0 and 1000 and assuming accurate parameter estimates in a classifier, classifier predictions fluctuate around 500, and consequently also the prediction errors fluctuate around 500. As in the more sensitive action set size estimates in Section 4.2.2, here the sensitivity is manifested in the prediction error ε. More specific classifiers have a less sensitive ε and consequently a higher variance in the ε values. Since the accuracy calculation expressed in Equation (3.5) scales the prediction error to the power ν, the higher variance causes *on average* higher accuracy and thus higher fitness.

Different parameter initialization techniques in combination with the moyenne adaptive modifiée technique enhance this influence. The more a classifier is inexperienced, the more the classifier parameters are dependent on the most recent cases. This, in combination with the scaled fitness approach, can make the effect even stronger. Since we set the experience exp of a new classifier to 1, XCS keeps the decreased parental parameter estimates so that fitness over-estimation is prevented. In fact, experimental runs with $exp = 0$ show that the specificity can increase to a level of even 0.2 independent of the mutation setting.

When applying the usual deletion strategy, based on as and the fitness estimate f, deletion causes an increase in the specificity of $[P]$ early on as shown in Figure 4.6b. This longer convergence time is attributable to the bias on as as already observed in Figure 4.5b. The additional fitness bias causes hardly any observable influence.

Overall, it can be seen that in a random function fitness causes a slight intrinsic pressure towards higher specificity. This pressure is due to the parameter initialization method and the higher variance in more specific classifiers. The on-average higher fitness in more specific classifiers causes fitness pressure and deletion pressure to favor those more-specific classifiers; thus the resulting undirected slight pressure towards higher specificity. Note that the specificity change and the convergence to a particular specificity level observed in Figure 4.6b should essentially take place in all problems that are similar to a random function. This is particularly the case if classifiers are overgeneral and the investigated problem provides no fitness guidance from the overgeneral side.

4.3 Improving Fitness Pressure

The previous section has shown that XCS has an intrinsic generalization pressure that needs to be overcome by a sufficiently strong fitness pressure in order to evolve accurate classifiers. Selection in XCS and other LCSs has always been done by means of proportionate selection. Although proportionate selection is known to be strongly dependent on fitness scaling and the fitness distribution in the population (Baker, 1985; Goldberg & Deb, 1991; Goldberg & Sastry, 2001; Goldberg, 2002), the LCS community has somewhat adhered to proportionate selection.

This section shows that XCS can actually suffer from the pitfalls of proportionate selection observed in the GA literature. Moreover, we show that tournament selection with tournament sizes proportional to the action set size can solve the problem resulting in a strong, stable, and reliable fitness pressure towards more accurate classifiers.[2]

4.3.1 Proportionate vs. Tournament Selection

Proportionate selection was applied and analyzed in Holland's original GA work (Holland, 1975). However, proportionate selection strongly depends both on fitness scaling (Baker, 1985; Goldberg & Deb, 1991) as well as on the current fitness distribution in the population. The smaller the fitness differences in the population the smaller the fitness pressure. Goldberg and Sastry (2001) show that evolutionary progress stalls when a population comes close to convergence since the fitness differences are not sufficiently strong anymore

Fitness of XCS classifiers is derived from the scaled, set-relative accuracy. Although

[2]Related publications of parts of this section can be found elsewhere (Butz, Sastry, & Goldberg, 2003; Butz, Goldberg, & Tharakunnel, 2003; Butz, Sastry, & Goldberg, 2004).

fitness scaling usually works well and proportionate selection is applied in the current action sets and not in the whole population, the more similarly accurate classifiers are, the less fitness pressure due to proportionate selection is expectable. In effect, similar accuracy of all classifiers in an action set should decrease or even annihilate fitness pressure.

Tournament selection, on the other hand, does not care about the current relative fitness differences. What matters is fitness rank. Thus, tournament selection does not suffer from fitness scaling nor from very small differences in accuracy. As long as there are significant differences in accuracy, tournament selection detects them and propagates the higher accurate classifier.

The next section shows that proportionate selection does not only suffer from cases in which classifiers are expected to have similar fitness values but fitness pressure can actually be insufficiently strong. We propose set-size relative tournament selection as the remedy and confirm its superior performance in the exemplar multiplexer problem.

4.3.2 Limitations of Proportionate Selection

To reveal the limitations of proportionate selection, we apply XCS to the multiplexer problem with various parameter settings or with additional noise in the problem. We show that learning in XCS with proportionate selection is disrupted if the learning parameter β is set too low or if noise in set too high. The multiplexer problem is introduced in Appendix C.[3]

Figure 4.7 reveals the strong dependence on parameter β. Decreasing the learning rate hinders XCS from evolving an accurate problem solution. The problem is that initially overgeneral classifiers occupy a big part of the population. Better offspring often loose against the overgeneral parents since the fitness of the offspring only increases slowly (due to the low β value). Small differences in the fitness F only have small effects when using proportionate selection. Altering the slope of the accuracy curve by changing parameters α and ε_0 does not have any positive learning effect.

Figure 4.8 reveals XCS's dependence on initial specificity. Increasing $P_\#$ (effectively decreasing initial specificity) impairs the learning speed of XCS since fitness does not cause sufficient specialization pressure. Decreasing the mutation rate μ also has a detrimental effect strongly delaying learning progress. The generalizing set pressure appears to be often

[3]Unless stated otherwise, all results in this section are averaged over 50 experimental runs. Performance is assessed by test trials in which no learning takes place and the better prediction array value is chosen as the classification. During learning, classifications are chosen at random. Parameters are set as follows: $N = 2000$, $\beta = 0.2$, $\alpha = 1$, $\varepsilon_0 = .001$, $\nu = 5$, $\theta_{GA} = 25$, $\chi = 1.0$, $\mu = 0.04$, $\theta_{del} = 20$, $\delta = 0.1$, $\theta_{sub} = 20$, and $P_\# = 0.6$.

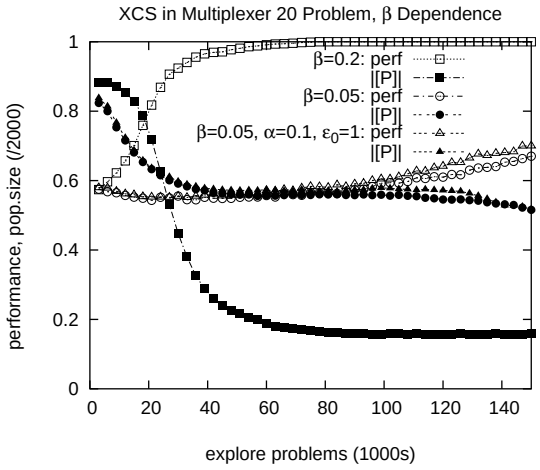


Figure 4.7: A lower learning rate β decreases XCS learning performance. Accuracy function parameters have no immediate positive influence.

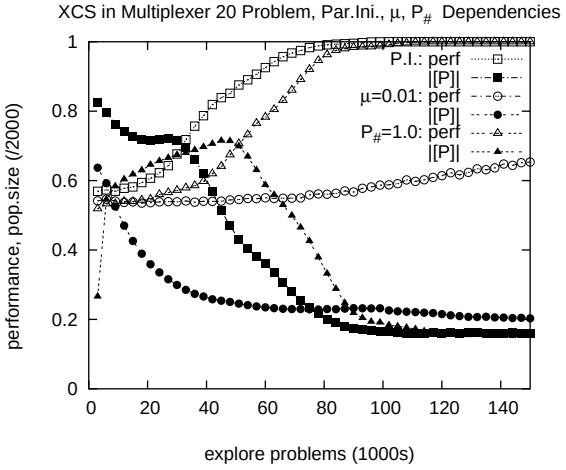


Figure 4.8: Proportionate selection is strongly dependent on offspring initialization, mutation rate, and initial specificity.

stronger than the specializing fitness pressure so that the chances of reaching higher accurate classifiers are significantly decreased.

Additionally, we detected significant parameter initialization effects. If prediction p is set to the current reward value r and reward prediction error is set to the average error in the action set, learning speed is slightly increased. Since in our problems only zero or thousand reward is possible and reward prediction is set directly to either one of the values, if the classifier is accurate, its error will decrease faster and fitness will increase faster.

In addition to the dependency on parameters β, $P_{\#}$, μ, and initialization, we can show that XCS with proportionate selection is often not able to solve noisy problems. We added two kinds of noise to the multiplexer problem: (1) Gaussian noise with a standard deviation σ is added to the payoff provided by the environment; (2) The payoff is swapped with a certain probability, termed *alternating noise* in the remainder of this work. Figures 4.9 and 4.10 show that XCS's performance is strongly degraded when adding only a small amount of either noise. Similar observations were made in Kovacs (2003), who added Gaussian noise to the reward in some of the output classes.

In general, the more noise is added, the smaller the fitness difference between accurate and inaccurate classifiers. Thus, selection pressure decreases due to proportionate selection and the population starts to drift at random. Lanzi (Lanzi, 1999c) proposed an extension to XCS that detects noise in environments and adjusts the error estimates accordingly. This approach, however, does not solve the parameter dependencies nor problems in which noise

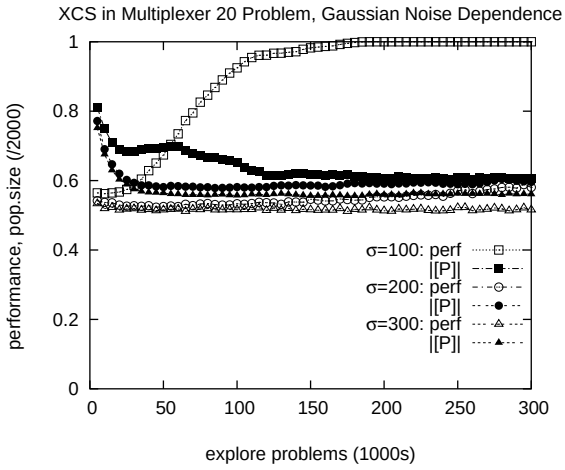


Figure 4.9: Adding Gaussian noise to the payoff function of the multiplexer problem significantly deteriorates performance of XCS.

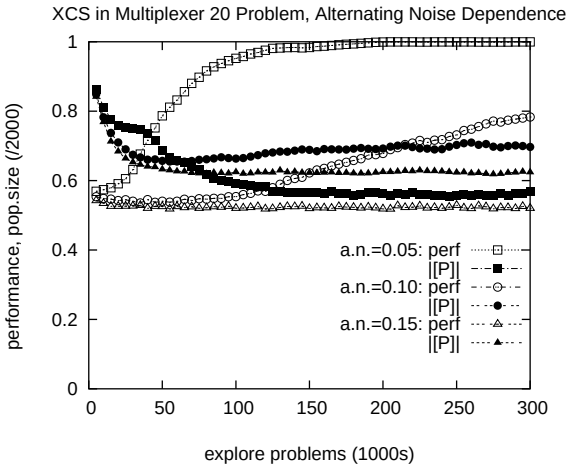


Figure 4.10: Also alternating noise, in which a percentage of examples is assigned to the incorrect class, significantly degrades performance of XCS.

is not equally distributed over the problem space.

4.3.3 Tournament Selection

In contrast to proportionate selection, tournament selection is independent of fitness scaling (Goldberg & Deb, 1991). In tournament selection parental classifiers are not selected proportional to their fitness, but tournaments are held in which the classifier with the highest fitness wins (stochastic tournaments are not considered herein). Participants for the tournament are usually chosen at random from the population in which selection is applied. The size of the tournament controls the selection pressure. Fixed tournament sizes are generally used in GAs.

Compared to standard GAs, the GA in XCS is a steady-state, niched GA. Only two classifiers are selected in each GA application and selection is restricted to the classifiers in the current action set. Thus, some classifiers might not get any reproductive opportunity at all before being deleted from the population. Additionally, action set sizes can vary significantly. Initially, action sets are often over-populated with overgeneral classifiers. Thus, a relatively strong selection pressure appears to be necessary which adapts to the current action set size.

Our tournament selection process holds tournaments of sizes dependent on the current

action set size $|[A]|$ choosing a subset of size $\tau|[A]|$ ($\tau \in (0,1]$) of the classifiers in $[A]$.[4] Instead of proportionate selection, two independent tournaments are held in which the classifier with the highest fitness is selected. We also experimented with fixed tournament sizes, shown below, which did not result in a stable selection pressure. The tournament selection procedure is described in algorithmic form in Appendix B.

The action set size proportionate tournament size assures that the current best classifier (assuming only one copy) is selected at least once with probability $1-(1-\tau)^2$. For example, if the tournament size is set to $\tau = 0.4$ of the population, we assure that the maximally accurate classifier is part of at least one of the two tournaments with a probability of 0.64. On average, $2\tau^2 + 2(\tau(1-\tau)) = 2\tau$ optimal classifiers are selected. The more copies of the best classifier exist, the higher the probability. This derivation is impossible when fixed tournament sizes are used since the action set size continuously varies and consequently the probability of selecting the best classifier continuously varies as well.

XCSTS in the Previous 20 Multiplexer Settings

Figures 4.11 and 4.12 show that XCS with tournament selection, referred to as *XCSTS* in the remainder of this work, can solve the 20 multiplexer problem even with a low parameter value β, a low parameter value μ, or a high parameter value $P_{\#}$. XCSTS is also more independent from initial parameter settings. The much stronger and stable fitness pressure overcomes the generalizing set pressure even without the help of mutation pressure. A reliable and stable performance increase is observable.

Figures 4.13 and 4.14 show that XCSTS is much more robust in noisy problems as well. XCSTS solves the same Gaussian noise 20 multiplexer problem with nearly no performance degradation (Figure 4.13). Despite the noisy parameter estimation values, tournament selection detects the more accurate classifiers generating a sufficiently strong fitness pressure. The decrease in the differences of accuracy due to the additional noise hardly affects XCSTS. Also in the alternating noise case, XCSTS reaches a higher performance level (Figure 4.14). Note that as expected, the population sizes do not converge to the sizes achieved without noise since subsumption does not apply. Nonetheless, in both noise cases the population sizes decrease indicating the detection and convergence to maximally accurate classifiers.

[4] If not stated differently, τ is set to 0.4 in the subsequent experimental runs.

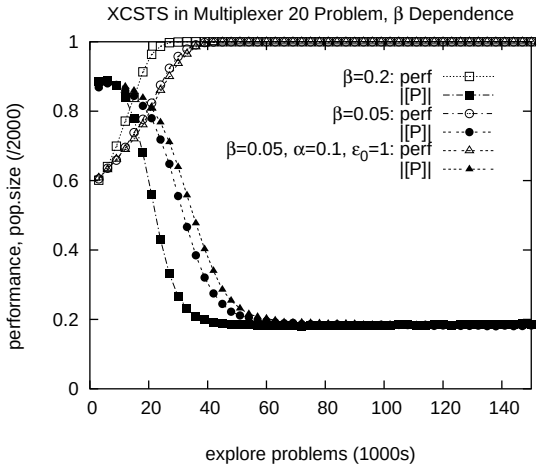


Figure 4.11: XCSTS is hardly influenced by a decrease in learning rate β. As in the proportionate selection case, accuracy parameters do not influence learning.

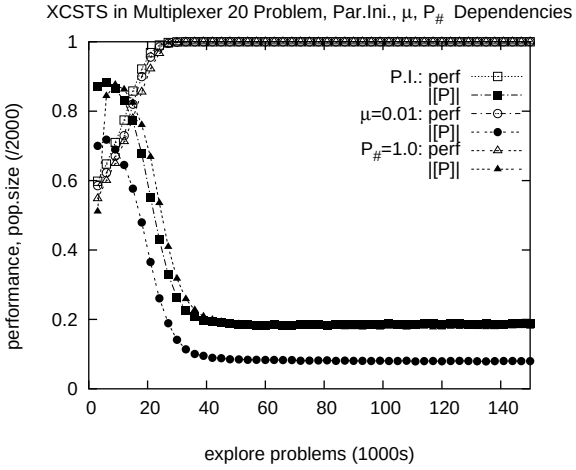


Figure 4.12: Other parameter variations such as a lower initial specificity ($P_{\#} = 1.0$), a low mutation rate ($\mu = 0.01$), or a different offspring initialization ($P.I.$) does hardly influence XCSTS's learning behavior.

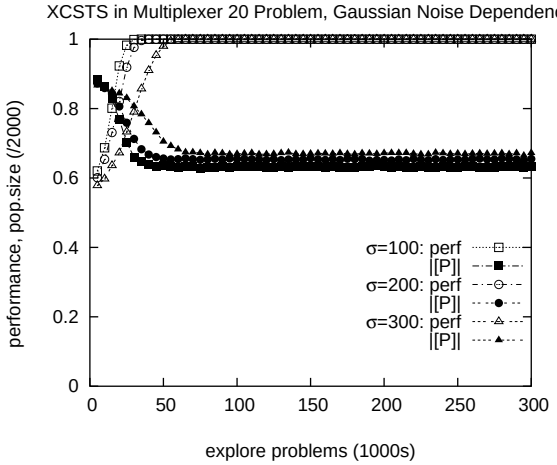


Figure 4.13: XCSTS performs much better than XCS in noisy problems. Performance is hardly influenced when adding Gaussian noise of up to $\sigma = 300$.

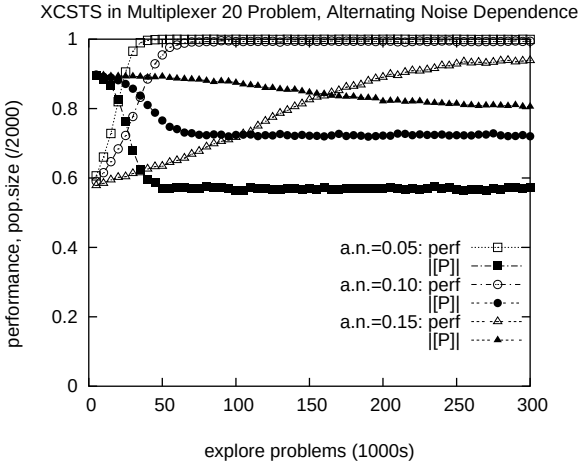


Figure 4.14: Also in the alternating noise case, XCSTS's learning behavior is faster and more reliable.

Tournament Selection with Fixed Size

XCS's action sets vary in size and in distribution. Dependent on the initial specificity in the population (controlled by parameter $P_{\#}$), the average action set size is either large or

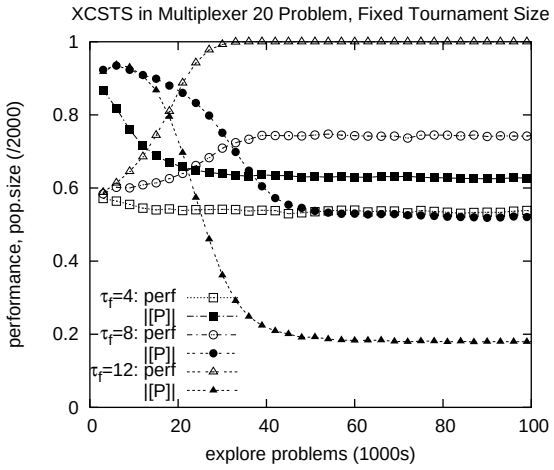


Figure 4.15: Due to fluctuations in action set sizes and distributions as well as the higher proportion of more general classifiers in action sets, fixed tournament sizes are inappropriate for selection in XCSTS.

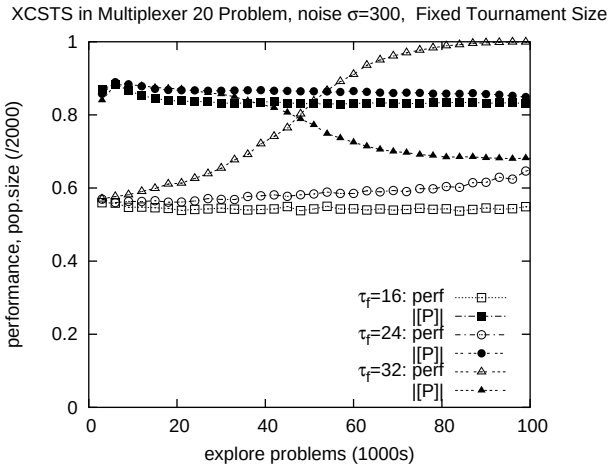


Figure 4.16: In the noisy problem case, tournament selection with a fixed tournament size can hardly solve the problem.

small initially. As was shown above, the average specificity in an action set is on average smaller than the specificity in the whole population. Replication in action sets and deletion from the whole population results in an implicit generalization pressure that can only be overcome by a sufficiently large specialization pressure. Additionally, the distribution of the specificities depends on initial specificity, problem properties, the resulting fitness pressure, and learning dynamics. Thus, an approach with fixed tournament size is dependent on the particular problem and probably not flexible enough. Kovacs used fixed tournament sizes to increase fitness pressure in his comparison to a strength-based XCS version (Kovacs, 2003).

In Figure 4.15 we show that XCSTS with fixed tournament size τ_f only solves the multiplexer problem with the large tournament size of $\tau_f = 12$. Since the population is usually over-populated with overgeneral classifiers early in a run, action set sizes are large so that a small tournament size mainly causes competition only among overgeneral classifiers. Thus, not enough fitness pressure is generated. When adding noise, an even larger tournament size is necessary (Figure 4.16). A tournament size of $\tau_f = 32$, however, does not allow any useful recombinatory events anymore since the action set size itself is usually not much bigger than that. Thus, fixed tournament sizes are inappropriate for XCS's selection mechanism.

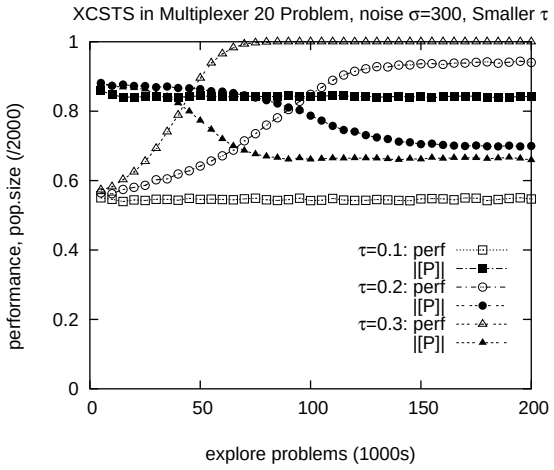


Figure 4.17: The strength of the selection pressure can be manipulated by the tournament size proportion. Very small proportions result in an insufficiently strong pressure.

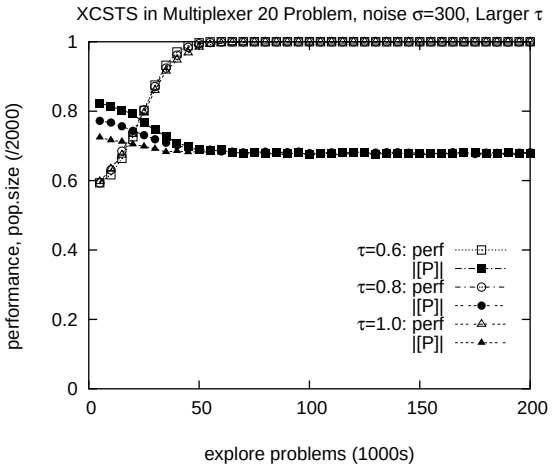


Figure 4.18: Very large tournament set size proportions may prevent effective recombination. In the multiplexer problem, this restriction results in hardly any performance influence.

Different Tournament Sizes

A change in the relative tournament size effectively changes the strength of the selection pressure applied. While a tournament size $\tau = 0$ corresponds to random selection, $\tau = 1$ corresponds to a deterministic selection of the classifier with the currently highest fitness in the action set and thus the strongest selection pressure. As can be seen in Figure 4.17 and Figure 4.18, XCSTS is able to generate a complete and accurate problem representation for a large range of τ. However, if selection pressure is too weak, learning may not take place at all or may be delayed. On the other hand, if selection pressure is very strong, crossover never has any effect since identical classifiers are crossed. The lack of effective recombinations does hardly influence XCS performance in the multiplexer problem as shown in Figure 4.18. However, in other problems ineffective crossover may strongly impair XCS's learning capabilities. Thus, τ may not be set to 1. In practice, a value of 0.4 proved to be robust.

4.3.4 Specificity Guidance Exhibited

As theorized above, the two selection methods differ in their dependence on the fitness estimate and fitness distribution. The fitness pressure, resulting from proportionate selection, depends on fitness scaling and in particular on the relative amount of fitness difference.

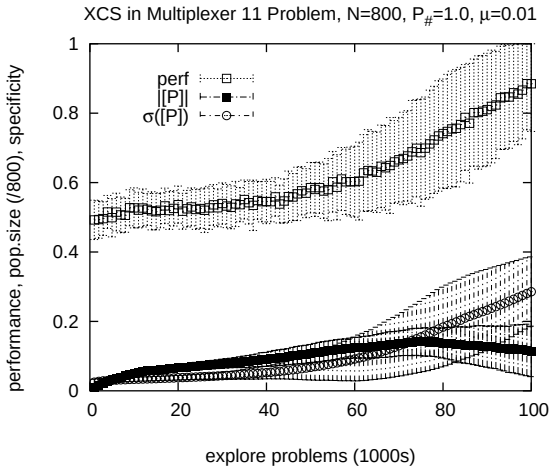


Figure 4.19: Starting with an overgeneral population and low mutation rate, XCS is not able to pick up the accuracy signal reliably.

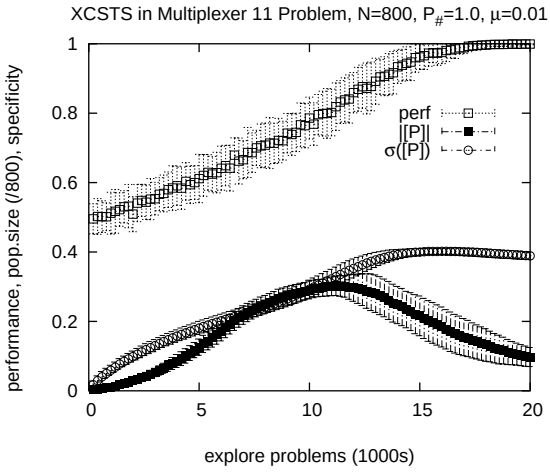


Figure 4.20: XCSTS immediately pushes the population towards higher specificity and thus higher accuracy.

Tournament selection on the other hand only depends on the fitness difference itself and not on the amount of the difference. That is, as long as the more accurate classifiers have a higher fitness estimate (regardless of how much higher the estimate is), tournament selection causes fitness pressure towards higher accuracy.

To exhibit this pressure, we monitor the average specificity in the population as well as the average standard deviation of the specificity. Figures 4.19 and 4.20 show the change in specificity when starting with a completely general population ($P_{\#} = 1.0$) in the 11 multiplexer problem. Figure 4.20 shows that XCSTS immediately identifies higher accurate classifies causing the specificity to rise. XCS, on the other hand, stalls at the overgeneral level apparently relying on a lucky guess for successful learning (Figure 4.19).

Another indicator can be found in the performance increase showing also the standard deviation over the experiments. Since XCSTS detects the fitness guidance in the problem immediately, the standard deviation between the runs remains small and the performance is hardly affected by a higher initial generality. Performance of XCS with roulette wheel selection on the other hand is much less reliable and strongly depends on initial generality. If there is no accurate classifier generated initially, roulette wheel selection has problems to generate a sufficiently strong fitness guidance.

4.4 Summary and Conclusions

This chapter has shown how XCS evolves a complete, accurate, and maximally general problem solution. Several *evolutionary pressures* guide the initial classifier population to the solution.

1. *Fitness pressures* is the main pressure towards higher accuracy.
2. *Set pressure* causes classifier generalization towards higher *semantic* generality.
3. *Mutation pressure* causes diversification searching in the syntactic neighborhood of currently best subsolutions. Mutation has also an effect on specificity.
4. *Deletion pressure* emphasizes the maintenance of a complete solution.
5. *Subsumption pressure* propagates accurate classifiers that are *syntactically* more general.

Combining set pressure and mutation pressure, we derived a *specificity equation*, which is able to predict the expected specificity change in an XCS population as long as no additional fitness pressure applies. In conjunction with fitness pressure and subsumption pressure, the pressures push towards an equilibrium that coincides with the desired complete, maximally accurate, and maximally general problem solution.

However, to reach this equilibrium, fitness pressure needs to be strong enough to reliably overcome the generalizing set pressure. We showed that proportionate selection may not be sufficiently strong due to parameter influences as well as problem influences such as a noisy reward function. Thus, we introduced *tournament selection* with tournament sizes proportional to the current action set size. Since the tournament size is set proportionate to the current action set size, the minimal probability of selecting the most accurate classifier is fixed so that also the minimal fitness pressure is fixed. We showed that XCS with tournament selection is able to solve the investigated problems reliably confirming the theorized reliable fitness pressure towards higher accuracy.

With respect to the facetwise theory approach for LCSs, we can now assure the first major aspects of the approach: (1) Due to the addition of tournament selection, fitness guides reliably to the intended solution. The accuracy-based fitness approach prevents strong overgenerals; (2) Parameters are estimated appropriately using adapted Q-learning and the moyenne adaptive modifiée technique; (3) Appropriate generalization applies as quantitatively analyzed in the *specificity equation*. Other influences with respect to generalization

cause (crossover) or slight additional specificity influence (deletion, parameter initialization, accuracy determination) but no disruption. Subsumption pushes towards maximally syntactically general, accurate classifiers as long as complete accuracy can be reached in the problem (reward prediction error ε drops below ε_0).

With the first main aspect of our facetwise LCS theory understood and satisfied, we are now ready to face the second aspect. The next chapter consequently investigates the computational effort necessary to ensure solution growth and sustenance.

Chapter 5

When XCS Works: Towards Computational Complexity

The last chapter investigated fitness guidance and generalization in the XCS classifier system. We showed that several evolutionary pressures apply that are designed to push the evolutionary process towards the desired complete, maximally accurate, and maximally general problem solutions. We saw that in order to enable reliable fitness pressure, offspring selection should be done using *tournament selection* with tournament sizes proportional to the current action set size to ensure a constant minimal pressure towards selecting better offspring in action sets.

The analysis in the last chapter enables us now to investigated the next four points of our facetwise LCS theory approach to ensure solution growth and sustenance:

- *Population initialization* needs to ensure time for classifier evaluation and successful GA application.
- *Schema supply* needs to be ensured to have better classifiers available.
- *Schema growth* needs to be ensured to grow those better classifiers evolving a complete problem solution.
- *Solution sustenance* needs to be ensured to sustain a complete problem solution.

We address these issues in the subsequent sections.

First, we derive the *covering bound* to ensure proper initialization. Second, we derive the *schema bound* to ensure the availability of better schema representatives for reproduction. We use the schema bound to derive initial settings for population specificity and population size. Also the time it takes to generate a better classifier at random is considered. Third, we

derive the *reproductive opportunity bound* to ensure that better classifiers can be detected and reproduced successfully. We show that the schema bound and the reproductive opportunity bound somewhat interact since, intuitively, too much supply implies too large specificity consequently disabling reproduction.

While the reproductive opportunity bound assures that better classifier grow, we are also interested in how long it takes them to grow. This is expressed in the learning time bound derived next.

Once enough time is given to make better classifiers grow until the final solution is found, we finally need to assure *sustenance* of the grown solution. XCS applies niching in that it reproduces in problem subspaces and deletes from the whole population. Larger niches are preferred for deletion. The analysis results in a final population size bound ensuring the sustenance of a complete problem solution as long as there are no severe problem solution overlaps.

Putting the results together, we are able to derive a positive computational learning theory result for an evolutionary-based learning system with respect to *k-DNF* functions. We show that XCS is able to PAC-learn *k-DNF* functions with few restrictions. However, the reader should keep in mind that XCS is a system that is much more broadly applicable and actually an online generalizing RL system. XCS's capability of PAC-learning *k-DNF* functions confirms its general learning scalability and potential widespread applicability.

5.1 Proper Population Initialization: The Covering Bound

Several issues need to be considered when intending to make time for classifier evaluation and thus the identification of better classifiers. The first bound is derived from the rather straight-forward requirement that the evolutionary algorithm in XCS needs to apply. That is, reproduction in action sets and deletion in the population needs to take place.[1]

The requirement is not met if the initial population is over-specialized since this can cause XCS to get stuck in an infinite covering - deletion loop. Normally, covering occurs only briefly in the beginning of a run. However, if covering continues indefinitely because inputs are continuously not covered due to an over-specialized population, the GA cannot take place and the evolutionary pressures do not apply. The issue basically can only happen

[1]Related publications of parts of this and the following section can be found elsewhere (Butz, Kovacs, Lanzi, & Wilson, 2001; Butz, Goldberg, & Tharakunnel, 2003; Butz, Kovacs, Lanzi, & Wilson, 2004).

if parameter $P_\#$ is set too low so that the initial specificity in the population is too high.

As described in the XCS introduction, covering creates classifiers given a problem instance that is not matched by at least one classifier for each possible classification. Since the population is of fixed size N, if the population is already filled up with classifiers, other classifiers are deleted to make space for the new covering classifiers. In the beginning of a run, with a population of classifiers that have a very low experience, the fitness F as well as the action set size estimate *as* of these classifiers is basically meaningless. Consequently, the deletion method chooses classifiers for deletion at random. Dependent on the specificity of classifiers generated by covering (determined by the parameter $P_\#$) the average specificity in the population may be too high to cover the whole problem space. Thus, it may happen that the population fills up with over-specialized classifiers and a *covering-random deletion* cycle continues forever.

Assuming a uniform problem instance distribution over the whole problem space $\mathcal{S} = \{0,1\}^l$, we can determine the probability that a given problem instance is covered by at least one classifier in a randomly generated population:

$$P(\texttt{cover}) = 1 - \left(1 - \left(\frac{2-\sigma[P]}{2}\right)^l\right)^N, \tag{5.1}$$

where $\sigma[P]$ may be equated with $1-P_\#$ since the covering bound only applies in the beginning of a run. Similarly, we can derive the actual necessary maximal specificity given a certain population size using the inequality $1 - \exp^x < x$ setting $(1 - \texttt{cover})$ to $1/\exp$:

$$\sigma[P] < 2(1 - (\frac{1}{N})^{1/l}) < 2 - 2(1 - (1 - \texttt{cover})^{1/N})^{1/l}, \tag{5.2}$$

showing that increasing N results in an increase in maximal specificity polynomial in $1/l$ deriving an effective rule of thump of how low the don't care probability $P_\#$ may be set to assure an appropriate specificity $\sigma[P]$. Figure 5.1 shows the resulting boundary conditions on population size and specificity for different problem lengths requiring a confidence level of 0.01.

To automatically avoid the covering bound, XCS could be enhanced to detect infinite covering and consequently increase the $P_\#$ value. However, we did not experiment with such an enhancement so far since usually the covering bound can be easily circumvented by setting the parameter $P_\#$ large enough.

Given that the problem space is not sampled uniformly at random, the covering bound

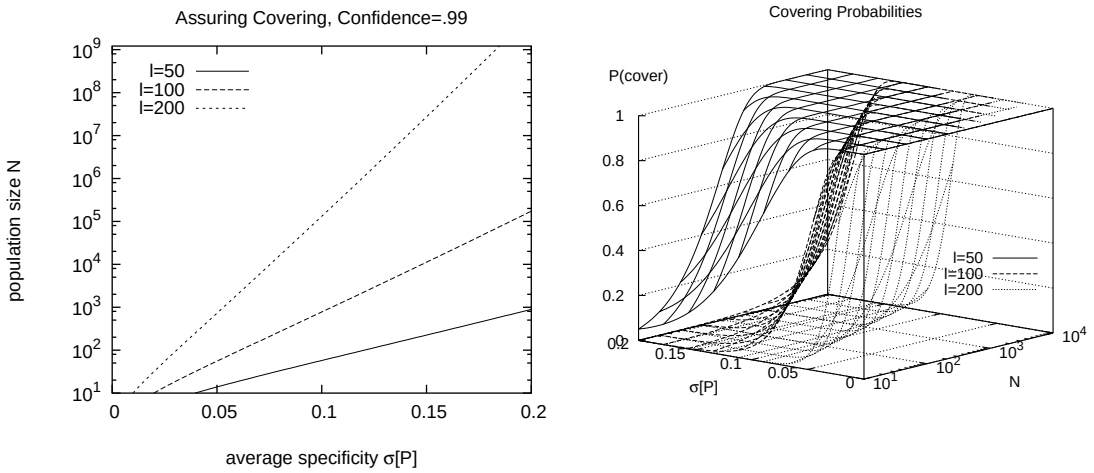


Figure 5.1: To ensure that better classifiers can be identified, population size needs to be set high enough and specificity low enough to satisfy the covering bound.

can be used as an upper bound. The smaller the set of sampled problem instances, the larger the probability that an instance is covered since the covering mechanism generates classifiers that cover actual instances and the genetic algorithm mainly focuses on generating offspring classifiers that apply in the current problem niche. Since in most RL problems as well as in datamining problems, the number of distinct problem instances is usually much smaller than the whole problem space so that the covering bound becomes less important.

5.2 Ensuring Supply: The Schema Bound

The supply question relates to the schema supply in GAs. GAs process BBs. However, BBs may be misleading as shown in the introduction of the trap problem in Chapter 1. While the fitness structure of one BB may point towards the mediocre solution (that is, a local optimum), the optimal solution to a BB as a whole needs to be processed. Thus, disregarding mutation effects, optimal BB structure needs to be available from the beginning.

The same observation applies in XCS albeit in slightly different form. The question arises, what a BB is in the XCS classifier system. We know that fitness is based on accuracy. Thus, a BB should be a substructure in the problem that increases classification accuracy. As there are BBs for GAs, there are minimal substructures in classification problems that result in higher accuracy (and thus fitness).

To establish a general notion of BBs in XCS, we use the notion of a *schema* as suggested elsewhere (Holland, 1971; Holland, 1975). A schema for an input of length l is defined as

a string that specifies some of the positions and ignores others. The number of specified positions is termed the *order* k of the schema. A schema is said to be *represented* by a classifier if the classifier correctly specifies *at least* all positions that are specified in the schema. Thus, a representative of a schema of order k has a specificity of at least $\sigma(.) = k/l$. For example, a classifier with condition $C =$`##10#0` is a representative of schema `**10*0`, but also of schemata `**10**`, `**1**0`, `***0*0`, `**1***`, `***0**`, `*****0`, and `******`.

Let's consider now a specific problem in which a minimal order of at least k_m bits need to be specified to reach higher accuracy. We call such a problem a problem that has a *minimal order of difficulty* k_m. That is, if less than k_m bits are specified in the problem, the class distribution is equal to the overall class distribution. In other words, the *entropy* of the class distribution decreases only if at least some k_m bits are specified. Since XCS's fitness is derived from accuracy, representatives of the schema of order k_m need to be present in the population.

5.2.1 Population Size Bound

To assure the supply of the representatives of a schema of order k_m, the population needs to be specific enough and large enough to avoid the covering bound. It is possible to determine the probability that a randomly chosen classifier from the current population is a schema representative by:

$$P(\texttt{representative}) = \frac{1}{n}\left(\frac{\sigma[P]}{2}\right)^{k_m}, \tag{5.3}$$

where n denotes the number of possible actions and $\sigma[P]$ denotes the specificity in the population, as before. From Equation 5.3 we can derive the probability of the existence of a representative of a specific schema in the current population

$$P(\texttt{representative exists}) = 1 - \left(1 - \frac{1}{n}\left(\frac{\sigma[P]}{2}\right)^{k_m}\right)^{N}, \tag{5.4}$$

basically deriving the probability that at least on schema representative of order k_m exists in $[P]$.

As we noted in the last chapter, in a problem in which no current fitness pressure applies, specificity can be approximated by twice the mutation probability μ, that is, $\sigma[P] \approx 2\mu$. Additionally, the population may be initialized to a desired specificity value by choosing parameter $P_\#$ appropriately. It should be kept in mind, though, that albeit $P_\#$ might bias specificity further early in the run, without any fitness influence, specificity converges to the

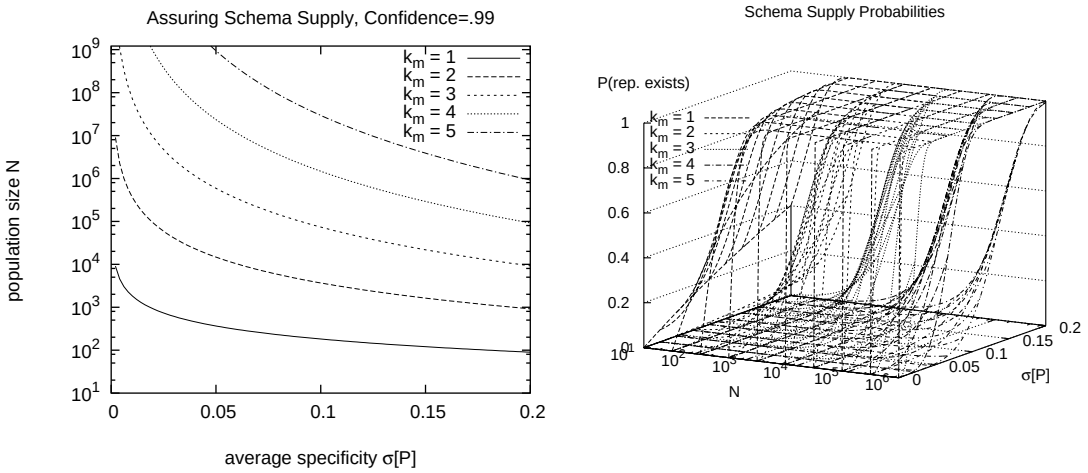


Figure 5.2: The schema bound requires population size to be set sufficiently high with respect to a given specificity. The larger the population size, the lower the necessary specificity.

values derived in the previous chapter and approximated by 2μ. Thus, mutation determines specificity on the long term. Well-chosen $P_\#$ values may boost initial XCS performance.

Requiring a high probability for the existence of a representative, we can derive the following population size bound using the inequality $x < -ln(1-x)$:

$$N > -n(\frac{2}{\sigma[P]})^{k_m} ln(1 - P(\texttt{rep.exists})) > \frac{\log(1 - P(\texttt{rep.exists}))}{\log(1 - \frac{1}{n}(\frac{\sigma[P]}{2})^{k_m})}, \tag{5.5}$$

which shows that N needs to grow logarithmically in the probability of error and exponentially in the minimal order of problem difficulty k_m given a certain specificity. Enlarging the specificity, we can satisfy the schema bound. However, schema growth may be violated if specificity is chosen too high as shown in the subsequent sections.

Figure 5.2 shows the schema bound plotting the required population size with respect to a given specificity and several minimal orders k_m (left-hand side). Population size is plotted in log scale due to its exponential dependence on the order k_m. We also show a 3-D plot of the behavior of the probability that a representative exists, formulated in Equation 5.4.

5.2.2 Specificity Bound

Similar to the bound on population size, given a certain specificity we can derive a minimal specificity bound from Equation 5.4 assuming a fixed population size.:

$$\sigma[P] > 2n^{1/k_m}(1 - (1 - P(\texttt{rep.exists}))^{1/N})^{1/k_m} \tag{5.6}$$

Setting $(1 - P(\texttt{rep.exists}))$ to $1/\exp$ we can use the inequality $1 - \exp^{-x} < x$ to derive that:

$$\sigma[P] > 2\left(\frac{n}{N}\right)^{1/k_m}. \tag{5.7}$$

Note that an identical derivation is possible determining the expected number of schema representatives in $[P]$ given specificity $\sigma[P]$:

$$E(representative) = \frac{N}{n}\left(\frac{\sigma[P]}{2}\right)^{k_m} \tag{5.8}$$

Requiring that at least one representative can be expected in the current population:

$$E(representative) > 1, \tag{5.9}$$

yields again Equation 5.7.

We may rewrite Equation 5.7 using the O-notation. Given a population size of N and the necessary representation of an unknown schema of order k_m, the necessary specificity $\sigma[P]$ can be bounded by

$$\sigma[P] : \; O\left(\left(\frac{n}{N}\right)^{1/k_m}\right), \tag{5.10}$$

showing that the required specificity decreases polynomially with increasing population size N and increases exponentially with increasing problem complexity k_m. Since we have shown that population size N also needs to increase exponentially in k_m but necessary specificity decreases polynomially in N, the two effects cancel each other out so that it is possible to leave specificity and thus mutation untouched focusing only on a proper population size to assure effective schema supply.

5.2.3 Extension in Time

Given we start with a completely general or highly general initial classifier population (that is, $P_\#$ is close to 1.0), the schema bound also extends in time. In this case, it is the responsibility of mutation to push the population towards the intended specificity generating initial supply.

Given a mutation probability μ, the probability can be approximated that a classifier is generated that has all k_m relevant positions specified given a current specificity $\sigma[P]$:

$$P(\texttt{generation of representative}) = (1-\mu)^{\sigma[P]k_m} \cdot \mu^{(1-\sigma[P])k_m} \tag{5.11}$$

With this probability, we can determine the expected number of steps until at least one classifier may have the desired attributes specified. Since this is a geometric distribution:

$$\begin{aligned} &E(t(\texttt{generation of representative})) = \\ &1/P(\texttt{generation of representative}) = \\ &\left(\frac{\mu}{1-\mu}\right)^{\sigma[P]k_m} \mu^{-k_m} \end{aligned} \tag{5.12}$$

Given a current specificity of zero, the expected number of steps until the generation of a representative consequently equals to μ^{-k_m}. Thus, given we start with a completely general population, the expected time until the generation of a first representative is less than μ^{-k_m} (since $\sigma[P]$ increases over time). Requiring that the expected time until a classifier is generated is smaller than some threshold Θ, we can generate a lower bound on the mutation μ:

$$\begin{aligned} \mu^{-k_m} &< \Theta \\ \mu &> \Theta^{\frac{1}{-k_m}} \end{aligned} \tag{5.13}$$

The extension in time is directly correlated with the specificity bound in Equation 5.7. Setting Θ to N/n we get the same bound (since σ can be approximated by 2μ).

As mentioned before, although supply may be assured easily by setting the specificity σ and thus $P_\#$ and more importantly the mutation rate μ sufficiently high, we yet have to assure that the supplied representatives can grow. This is the concern of the following section in which the schema bound is also experimentally evaluated in conjunction with the derived reproductive opportunity bound.

5.3 Making Time for Growth: The Reproductive Opportunity Bound

To ensure the growth of better classifiers, we need to ensure that *better* classifiers get reproductive opportunities. So far, the covering bound only assures that reproduction and evaluation are taking place. This is a requirement for ensuring growth but not sufficient for it. This section derives and evaluates the *reproductive opportunity bound* that provides a

population size and specificity bound that assures the growth of better classifiers.[2]

The idea is to assure that more accurate classifiers need to be ensured to undergo reproductive opportunities before being deleted. To do this, we minimize the probability of a classifier being deleted before reproduced. The constraint effectively results in another population and specificity bound since only a larger population size and a sufficiently small specificity can assure reproduction before deletion.

5.3.1 General Population Size Bound

Let's first determine the expected number of steps until deletion. Assuming neither any fitness estimate influence nor any action set size estimate influence, the probability of deletion is essentially random as already assumed in Chapter 4. Thus, the probability of deleting a particular classifier in a learning iteration equals:

$$P(\texttt{deletion}) = \frac{2}{N}, \tag{5.14}$$

since two classifiers are deleted per iteration. A reproductive opportunity takes place if the classifier is part of an action set. As we have seen in the previous chapter, the introduced tournament selection bounds the probability of reproduction of the best classifier from below by a constant. Thus, the probability of being part of an action set directly determines the probability of reproduction:

$$P(\texttt{reproduction}) = \frac{1}{n} 2^{-l\sigma(cl)} \tag{5.15}$$

Note that this probability assumes binary input strings and further that all 2^l possible binary input strings occur with equal probability. Combining Equation 5.14 with Equation 5.15, we can determine that neither reproduction nor deletion occurs at a specific point in time:

$$\begin{aligned} P(\texttt{no rep., no del.}) &= (1 - P(\texttt{del.}))(1 - P(\texttt{rep.})) = \\ &= (1 - \frac{2}{N})(1 - \frac{2^{-l\sigma[P]}}{n}) = 1 - \frac{2}{N} - \frac{2^{-l\sigma[P]}}{n}(1 - \frac{2}{N}) \end{aligned} \tag{5.16}$$

[2]Related publications of parts of this section can be found elsewhere (Butz & Goldberg, 2003; Butz, Goldberg, & Tharakunnel, 2003).

Together with equations 5.14 and 5.15, we can now derive the probability that a certain classifier is part of an action set before it is deleted:

$$P(\texttt{rep.before del.}) = P(\texttt{rep.})(1 - P(\texttt{del.})) \sum_{i=0}^{\infty} P(\texttt{no rep., no del.})^i =$$
$$P(\texttt{rep.})(1 - P(\texttt{del.})) \frac{1}{1 - P(\texttt{no rep., no del.})} =$$
$$\frac{\frac{1}{n}2^{-l\sigma[P]}(1 - \frac{2}{N})}{\frac{2}{N} + \frac{1}{n}2^{-l\sigma[P]}(1 - \frac{2}{N})} = \frac{\frac{1}{n}2^{-l\sigma[P]}}{\frac{2}{N-2} + \frac{1}{n}2^{-l\sigma[P]}} = \frac{N-2}{N-2+n2^{l\sigma[P]+1}} \tag{5.17}$$

Requiring a certain minimal reproduction before deletion probability and solving for the population size N, we get the following bound:

$$N > \frac{2n2^{l\sigma[P]}}{1 - P(\texttt{rep.before del.})} + 2 \tag{5.18}$$

This bounds the population size by $O(n2^{l\sigma})$. Since specificity σ can be set proportional to $\sigma = 1/l$, the bound actually diminishes usually. However, in problems in which the problem complexity $k_m > 1$, we have to ensure reproductive opportunities to classifiers that represent order k_m schemata.

The expected specificity of such a representative of an order k_m schema can be estimated given a current specificity $\sigma[P]$. Given that the classifier specifies all k positions, its expected average specificity can be approximated by:

$$E(\sigma(\texttt{repres.of schema of order } k_m)) = \frac{k_m + (l - k_m)\sigma[P]}{l} \tag{5.19}$$

Substituting $\sigma(cl)$ from Equation 5.18 with this expected specificity of a representative of a schema of order k, the population size N can be bounded by

$$N > 2 + \frac{n2^{l \cdot \frac{k+(l-k)\sigma[P]}{l}+1}}{1 - P(\texttt{rep.before del.})}$$
$$N > 2 + \frac{n2^{k+(l-k)\sigma[P]+1}}{1 - P(\texttt{rep.before del.})} \tag{5.20}$$

This bound ensures that the classifiers necessary in a problem of order of difficulty k_m get reproductive opportunities. Once the bound is satisfied, existing representatives of an order k_m schema are ensured to reproduce before being deleted and XCS is enabled to evolve a more accurate population.

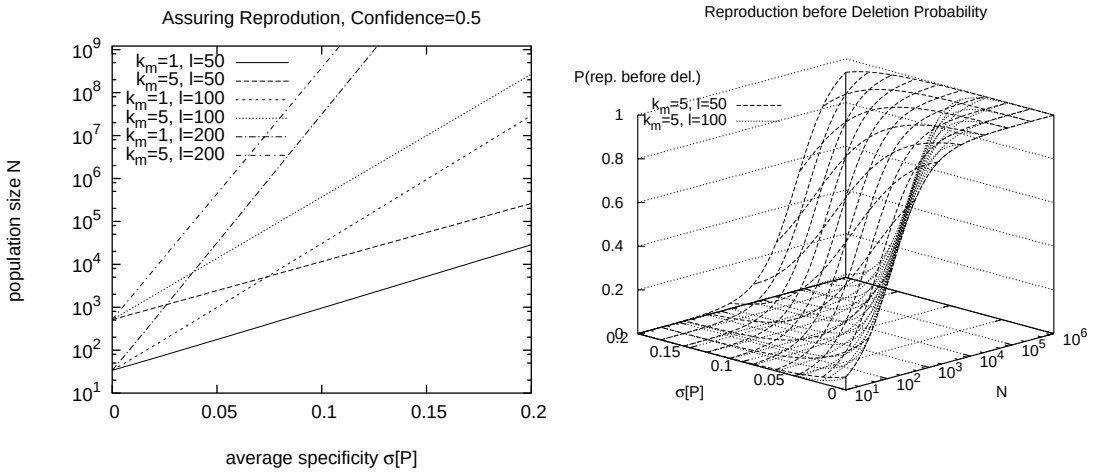


Figure 5.3: To ensure successful identification and reproduction of better classifiers, population size needs to be set high enough with respect to a given specificity. Specificity interacts with population size as shown in the three dimensional plot on the right-hand side.

Note that this population size bound is actually exponential in schema order k_m and in string length times specificity $l\sigma[P]$. This would mean, that XCS scales exponentially in the problem length which is certainly highly undesirable. However, since specificity in $[P]$ decreases with larger population sizes as shown in Equation 5.10, we can show that out of the specificity constraint a general *reproductive opportunity bound (ROP-bound)* can be derived that shows that population size grows as $O(l^{k_m})$.

Figure 5.3 shows several settings for the reproductive opportunity bound. On the left-hand side, the dependency of population size N on specificity $\sigma[P]$ is shown requiring a confidence value of .99. In comparison to the covering bound, shown in Figure 5.1, it can be seen that the reproductive opportunity bound is always stronger than the covering bound making the latter nearly obsolete. However, the covering bound is still useful to set the don't care probability $P_\#$ appropriately. The mutation rate and thus the specificity the population converges to, however, is stronger constraint by the reproductive opportunity bound. Figure 5.3 also shows a 3-D plot of the dependence of the probability of reproduction before deletion on population size and specificity (right-hand side).

5.3.2 General Reproductive Opportunity Bound

While the above bound ensures the reproduction of *existing* classifiers that represent a particular schema, it does not assure the actual presence or generation of such a classifier. Thus, we need to *combine schema bound* and *reproductive opportunity bound*. Figure 5.4 shows a

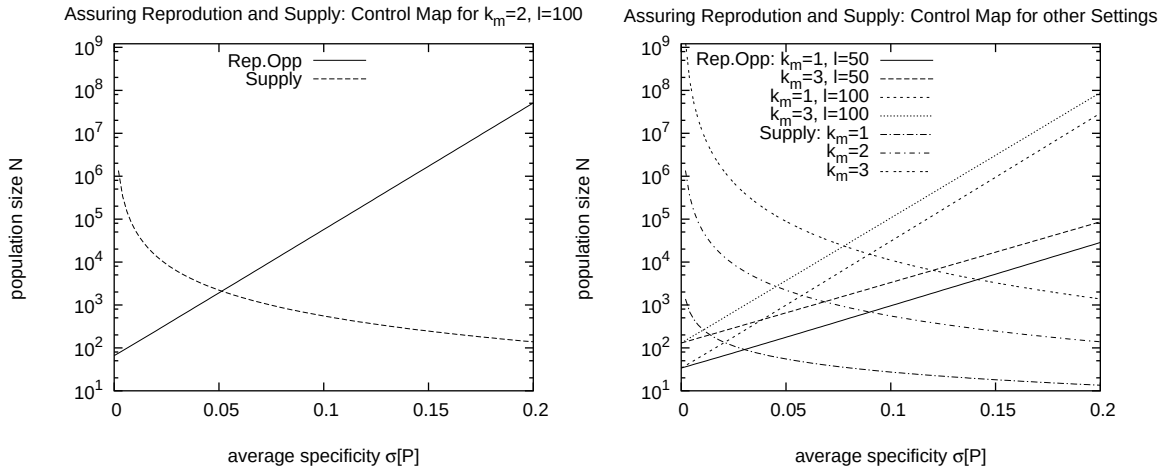


Figure 5.4: The shown control map clarifies the competition between reproduction and supply. The shown boundaries assure a 50% success probability. High specificity ensures supply but may hinder reproduction. Vice versa, low specificity ensures reproduction but lowers the probability of supply.

control map of certain reproductive opportunity bound values and schema bound values requiring a probability of success of 50% (plotting equations 5.5 and 5.20). The corresponding intersections denote the best value of specificity and population size to ensure supply and growth with high probability. Initial specificity may be set slightly larger than the value of the intersection to boost initial performance as long as the covering bound is not violated.

We can also quantify the interaction. Substituting the O-notated specificity bound in Equation 5.10 of the schema bound in the O-notated dependence of the representative bound on string length l ($N : O(2^{l\sigma[P]})$) and ignoring additional constants, we can derive the following enhanced population size bound:

$$\begin{aligned} N &> 2^{l(\frac{n}{N})^{1/k_m}} \\ \log_2 N &> l\left(\frac{n}{N}\right)^{1/k_m} \\ N^{1/k_m}\log_2 N &> l n^{1/k_m} \\ N(\log_2 N)^{k_m} &> n l^{k_m}. \end{aligned} \tag{5.21}$$

This general *reproductive opportunity bound* (ROP-bound) essentially shows that population size N needs to grow approximately exponentially in the minimal order of problem difficulty

k_m and polynomially in the string length.

$$N : O(l^{k_m}) \tag{5.22}$$

Note that in the usual case, k_m is rather small and can often be set to one. Essentially, when k_m is greater than one, other classification systems in machine learning have also shown a similar scale up behavior. For example, the inductive generation of a decision tree in $C4.5$ (Quinlan, 1993) would not be able to decide on which attribute to expand first (since any expansion leads to the same probability distribution and thus no information gain) and consequently would generate an inappropriately large tree.

5.3.3 Sufficiently Accurate Values

Although we assume in the main parts of this thesis that the estimation values of XCS are sufficiently accurate, we need to note that this assumption does not hold necessarily. All classifier parameters are only an approximation of the average value. The higher parameter β is set, the higher the expected variance of the parameter estimates. Moreover, while the classifiers are younger than $1/\beta$, the estimation values are approximated by the average of the so far encountered values. Thus, if a classifier is younger than $1/\beta$, its parameter variances will be even higher than the one for experienced classifiers.

Requiring that each offspring has the chance to be evaluated at least $1/\beta$ times to get as close to the real value as possible, the reproductive opportunity bound needs to be increased by the number of evaluations we require.

From Equation 5.14 and Equation 5.15, we can derive the expected number of steps until deletion and similarly, we can derive the expected number of evaluations during a time period t.

$$E(\texttt{\# steps until deletion}) = \frac{1}{P(\texttt{deletion})}/2 = \frac{N}{2} \tag{5.23}$$

$$E(\texttt{\# of evaluations in } t \texttt{ steps}) = P(\texttt{in } [A]) \cdot t = \frac{1}{n} 0.5^{l\sigma(cl)} t \tag{5.24}$$

The requirement for success can now be determined by requiring that the number of evaluations before deletion must be larger than some threshold Θ where Θ could for example be

set to $1/\beta$.

$$E(\texttt{\# of evaluations in (\# steps until deletion)}) > \Theta$$
$$\frac{1}{n} 0.5^{l\sigma(cl)} \frac{N}{2} > \Theta$$
$$N > \Theta n 2^{l\sigma(cl)+1} \tag{5.25}$$

Setting Θ to one, Equation 5.25 is basically equal to Equation 5.20 since one evaluation is equal to at least one reproductive opportunity disregarding the confidence value in this case. It can be seen that the sufficient evaluation bound only increases the reproductive opportunity bound by a constant. Thus, scale-up behavior is not affected.

As a last point in this section, we want to point out that fitness is actually not computed by averaging, but the Widrow-Hoff delta rule is used from the beginning. Moreover, fitness is usually set to 10% of the parental fitness value (to prevent disruption). Thus, fitness is derived from two approximations and it starts off with a disadvantage so that the early evolution of fitness strongly depends on fitness scaling and on accurate approximations of the prediction and prediction error estimates. To ensure a fast detection and reproduction of superior classifiers it is consequently necessary to choose initial classifier values as accurate as possible. Alternatively, the expected variance in fitness values could be considered to prevent potential disruption. (Goldberg, 1990), for example suggests to use the variance sensitive bidding. Accordingly using the estimated expectable variance, the fitness of young classifiers could be modified for selection to prevent disruption but also enable the earlier detection of better classifiers.

5.3.4 Bound Verification

We show that the derived bounds hold using a Boolean function problem of order of difficulty k_m where k_m is larger than one. The hidden parity function, introduced in Appendix C and originally investigated in XCS in Kovacs and Kerber (2001), is very suitable to manipulate k_m. The basic problem is represented by a Boolean function in which k relevant bits determine the outcome. If the k bits have an even number of ones, the outcome will be one and zero otherwise. The hidden parity function can also be viewed as an XOR function over k relevant bits in a bit string of length l. The order of difficulty k_m is equivalent to the number of relevant bits k.

The major problem in the hidden parity is that there is no fitness-guidance whatsoever before all k relevant hidden parity bits are specified. That is, if a classifier does not specify

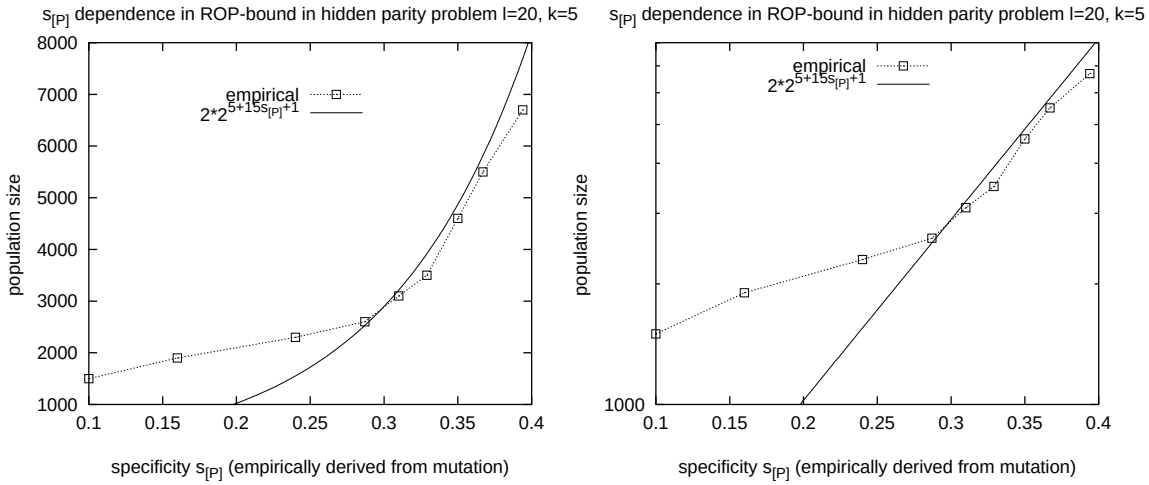


Figure 5.5: Minimal population size depends on applied specificity in the population (manipulated by varying mutation rates). When mutation and thus specificity is sufficiently low, the bound becomes obsolete.

the values of all k relevant positions, it has a 50/50 chance of getting the output correct. Thus, the accuracy for all classifiers that do not have all k bits specified is approximately equal.

Figure 5.5 shows that the reproductive opportunity bound is nicely approximated by Equation 5.20. The experimental values are derived by determining the population size needed to reliably reach 100% performance. The average specificity in the population is derived from the applied mutation using Table 4.1. XCSTS is assumed to reach 100% performance reliably when all 50 runs reach 100% performance after $200,000$ steps. Although the bound corresponds to the empirical points when specificity is high, in the case of lower specificity the actual population size needed departs from the approximation. The reason for this is the time extension of the schema bound derived in Section 5.2.3 and expressed in Equation 5.12.

For $k = 5$ and a small mutation rate $\mu = 0.04$, the reproductive opportunity bound requires a population size of less than 1000 classifiers. However, the time extension becomes significant since representatives are not present in the beginning of a run and take much longer to generate. This is confirmed in Figure 5.6 in which performance becomes rather independent of the chosen population size when mutation, and thus specificity, is set low. The reproductive opportunity bound becomes obsolete and the small mutation delays the detection of representatives significantly.

To validate the $O(l^{k_m})$ bound of Equation 5.22, we ran further experiments in the hidden

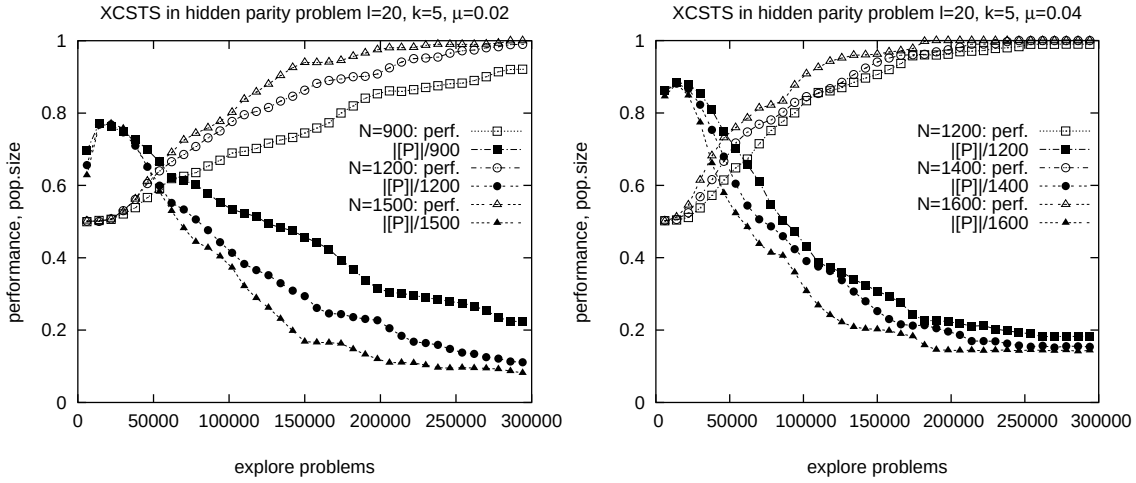


Figure 5.6: In the case of small mutation rates, different population sizes hardly affect performance and the time extension of the schema bound takes over.

parity problem with $k = 4$, varying l, using an optimal mutation rate μ (and thus specificity). Results are shown in Figure 5.7. The bound was derived from experimental results averaged over 50 runs determining the population size for which 98% performance is reached after 300000 steps. With appropriate constants added, the experimentally derived bound is well-approximated by the theory. Showing the same bound on a log-scale confirms that the population size needs to grow polynomially in the string length l.

5.4 Estimating Learning Time

Given schema, covering, and reproductive opportunity bound are satisfied, we essentially ensure the first three aspects of the second main point of our facetwise theory approach: Minimal order conditions are supplied due to the schema bound, evaluation time is available due to the covering bound, and finally time for reproduction is available due to the reproductive opportunity bound. Thus, it is assured that better classifier structures can grow in the population.

Before we address the next aspect in the facetwise theory, we are interested in how long it may take to evolve a complete problem solution by the means of this growing process.[3]

Assuming that the other problem bounds are satisfied by the choice of mutation and population size, we can estimate how long it takes to discover successively better classifiers

[3]Related publications of parts of this and the following section can be found in Butz, Goldberg, and Lanzi (2004a).

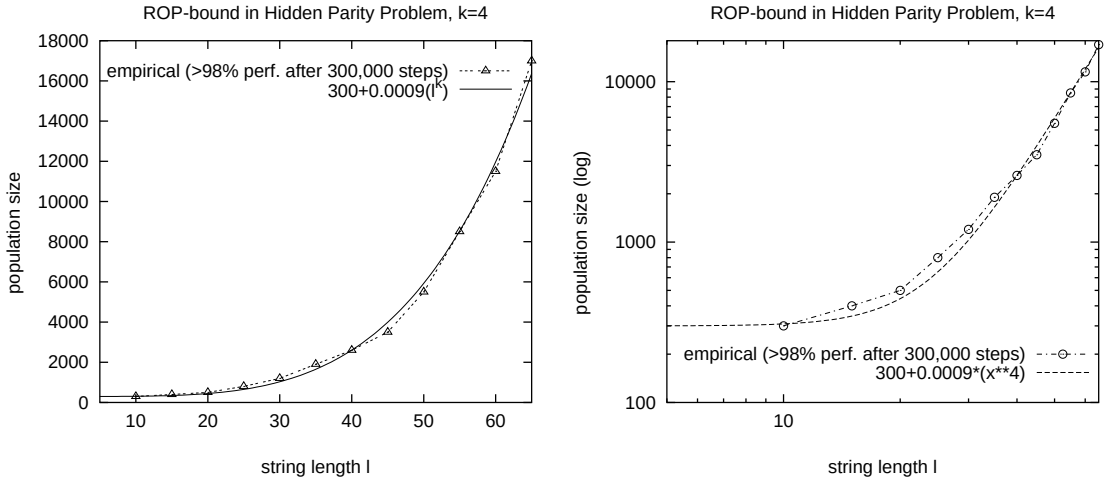


Figure 5.7: The general *reproductive opportunity bound* can be observed altering problem length l and using optimal specificity $\sigma[P]$, given a problem of order of difficulty k_m. The experimental runs in the hidden parity problem of order $k = 4$ ($k_m = k$) confirm that the population size necessary in XCS to ensure a proper evolutionary process indeed scales as $O(l^{k_m})$.

until the maximally general, accurate classifiers are found. To do this, we assume a domino convergence model (Thierens, Goldberg, & Pereira, 1998) estimating the time until each relevant attribute can be expected to be specialized to the correct value. Considering mutation only, we estimate the time until reproduction and the time until generation of the next best classifier in each problem niche. Using this approach we can show that learning time scales polynomially in problem length and problem complexity.

5.4.1 Time Bound Derivation

To derive our learning time bound, we estimate the time until reproduction of the current best classifier as well as the time until creation of the next best classifier via mutation given a reproductive event of the current best classifier. The model assumes to start with an initially completely general population (that is, $P_{\#} = 1.0$). Initial specializations are randomly introduced via mutation. Problem-specific initialization techniques or higher initial specificity in the population may speed-up learning time (as long as the covering bound is not violated).

Further assumptions are that the current best classifier is not lost and selected as the offspring when it is part of an action set (assured by the ROP-bound in conjunction with tournament selection). The time model assumes domino convergence (Thierens, Goldberg,

& Pereira, 1998) in which each attribute is successively specified. This means that only once the first attribute is correctly specified in a classifier, the second attribute influences fitness and so forth.

Using the above assumptions, we can estimate the probability that mutation correctly specifies the next attribute

$$P(\texttt{perfect mutation}) = \mu(1-\mu)^{l-1} \tag{5.26}$$

where l specifies the number of attributes in a problem instance. This probability can be relaxed in that we only require that the k already correctly set features are not unset (changed to don't care), the next feature is set, and we do not care about the others:

$$P(\texttt{good mutation}) = \mu(1-\mu)^{k} \tag{5.27}$$

Whereas Equation 5.26 specifies the lower bound on the probability that the next best classifier is generated, Equation 5.27 specifies an optimistic bound.

As seen before, the probability of reproduction can be estimated by the probability of occurrence in an action set. The probability of taking part of an action set again, is determined by the current specificity of a classifier. Given a classifier which specifies k attributes, the probability of reproduction is

$$P(\texttt{reproduction}) = \frac{1}{n}\frac{1}{2}^{k}, \tag{5.28}$$

where n denotes the number of actions in a problem. The best classifier has a minimal specificity of k/l. With respect to the current specificity in the population $\sigma[P]$, the specificity of the best classifier may be expected to be $k + \sigma[P](l-k)$ assuming a uniform specificity distribution in the other $l-k$ attributes. Taking this expected specificity into account, the probability of reproduction is

$$P(\texttt{rep.in [P]}) = \frac{1}{n}\frac{1}{2}^{k+\sigma[P](l-k)} \tag{5.29}$$

Since the probability of a successful mutation assumes a reproductive event, the probability

of generating a better offspring than the current best is determined by

$$P(\texttt{generation of next best cl.}) = P(\texttt{rep.in [P]})\ P(\texttt{good mutation}) = \frac{1}{n}\frac{1}{2}^{k+\sigma[P](l-k)}\mu(1-\mu)^{l-1}. \tag{5.30}$$

Since we assume uniform sampling from all possible problem instances, the probability of generating a next best classifier conforms to a geometric distribution (memoryless property, each trial has an independent and equally probable distribution), the expected time until the generation of the next best classifier is

$$E(\texttt{time until gen.of next best cl.}) = 1/P(\texttt{time until gen.of next best cl.}) = \frac{1}{\frac{1}{n}\frac{1}{2}^{k+\sigma[P](l-k)}\mu(1-\mu)^{l-1}} = \frac{n2^{k+\sigma[P](l-k)}}{\mu(1-\mu)^{l-1}} \leq \frac{n2^{k+\sigma[P]l}}{\mu(1-\mu)^{l-1}} \tag{5.31}$$

Given now a problem in which k_d features need to be specified and given further the domino convergence property in the problem, the expected time until the generation of the next best classifier can be summed to derive the time until the generation of the global best classifier:

$$E(\texttt{time until generation of maximally accurate cl}) = \sum_{k=0}^{k_d-1}\frac{n2^{k+\sigma[P]l}}{\mu(1-\mu)^{l-1}} = \frac{n2^{\sigma[P]l}}{\mu(1-\mu)^{l-1}}\sum_{k=0}^{k_d-1}2^k < \frac{n2^{k_d+\sigma[P]l}}{\mu(1-\mu)^{l-1}}. \tag{5.32}$$

This time bound shows that XCS needs an exponential number of evaluations in the order of problem difficulty k_d. As argued above, the specificity and consequently also mutation needs to be decreased indirect proportional to the string length l. In particular, since specificity $\sigma[P]$ grows as $O((\frac{n}{N})^{\frac{1}{k_m}})$ (Equation 5.10) and population size grows as $O(l^{k_m})$ (Equation 5.22), specificity essentially grows as

$$O(\frac{n}{l}) \tag{5.33}$$

Using the O-notation and substituting in Equation 5.32, we derive the following adjusted time bound making use of the inequality $(1+\frac{n}{l})^l < e^n$:

$$O\left(\frac{l2^{k_d+n}}{(1-\frac{n}{l})^{l-1}}\right) = O\left(\frac{l2^{k_d+n}}{e^{-n}}\right) = O\left(l2^{k_d+n}\right) \tag{5.34}$$

Thus, learning time in XCS is bound mainly by the order of problem difficulty k_d and the number of problem classes n. It is linear in the problem length l. This derivation essentially also validates Wilson's hypothesis that XCS learning time grows polynomially in problem complexity as well as problem length (Wilson, 1998). The next section experimentally validates the derived learning bound.

5.4.2 Experimental Validation

In order to validate the derived bound, we evaluate XCS performance on an artificial problem in which domino convergence is forced to take place. Similar results are expected in typical Boolean function problems in which similar fitness guidance is available, such as in the layered multiplexer problem (Wilson, 1995; Butz, Goldberg, & Tharakunnel, 2003). In other problems, additional learning influences may need to be considered such as the influence of crossover or the different fitness guidance in the problem (Butz, Goldberg, & Tharakunnel, 2003). These issues are addressed in more detail in the next chapter.

To force domino convergence, instead of using the usual Widrow-Hoff delta rule to update classifier estimates, we set the reward prediction error directly to a fixed value according to the current specificity of the classifier. Given a problem of problem difficulty k_d, the prediction error of a classifier is set to $500(k_d - k)/k_d$ where k denotes the number of successive relevant attributes specified in the classifier. For example, given a problem of length $l = 6$ and $k_d = 3$ (and assuming a left-to-right order with the first three features being relevant), classifier 1#1111 would be assigned an error of 333 whereas classifier 011#1# would be assigned an error of 0.

If not stated differently, the XCS classifier system is applied with a GA threshold $\theta_{GA} = 0$ (the GA is always applied), tournament selection with an action set proportionate tournament size of $\tau = 0.4$, selection based on minimizing error instead of maximizing fitness, niche mutation, no action mutation, no crossover, an initial completely general population ($P_\# = 1.0$), and GA subsumption ($\theta_{sub} = 0, \varepsilon_0 = 1$). The results are averaged over 20 experiments. The error-based selection approach eliminates the additional evolutionary influence due to fitness sharing.

To validate the time bound, we monitor the specificity of the relevant attributes. According to the domino convergence theory, the system should successively detect the necessary specialization of each relevant attribute eventually converging to a specificity of nearly 100%. The time bound estimates the expected time until all relevant attributes are detected. To evaluate the bound, we record the number of steps until the specificity of a particular

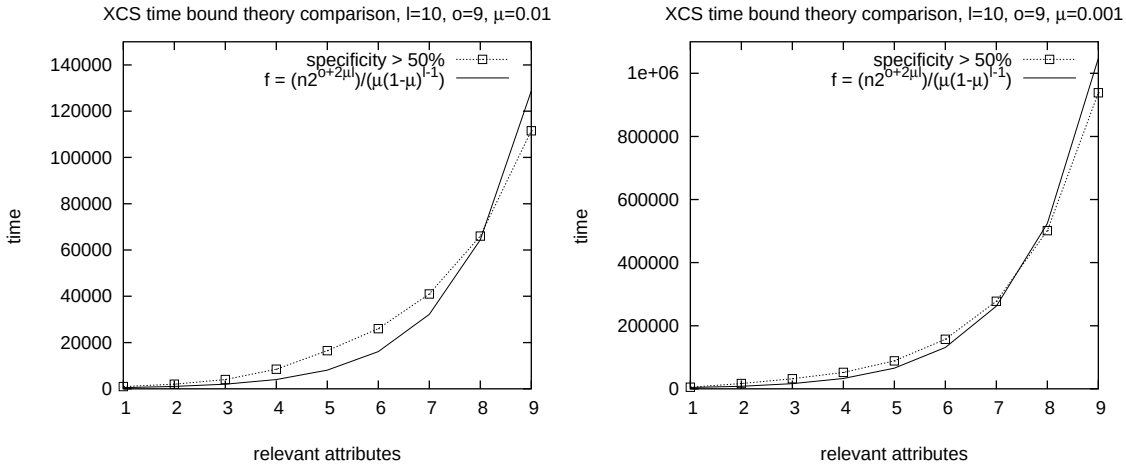


Figure 5.8: The theory comparison shows that the time until the specificity of the successive attributes has reached 50% is approximated by the theoretical bound. Maximum population size N is set to 32000.

attribute reaches 50%. This criterion indicates that the necessary specificity is correctly detected but it does not require full convergence.

Figure 5.8 shows the time until 50% specificity is reached in the successive attributes in the setting with $l = 10$ and $k_d = 9$. The experimental runs are matching with the theoretical bounds approximating the specificity in the population $\sigma[P]$ with 2μ as done above. As predicted by the theory, decreasing the mutation rate (Figure 5.8, right-hand side) increases the time until the required specificity is reached. Although nearly all interactions between the different niches are prevented by disallowing the mutation of the action part and by applying niche mutation only, the specificity in the later attributes still is learned slightly faster than predicted by the theory. Two-stage interactions might occur in which mutation first overgeneralizes a highly accurate classifier and then specializes it into another niche.

The second concern is the influence of the number of irrelevant attributes. Figure 5.9 shows that also in this case the theory closely matches the empirical results. Since a higher mutation rate results in a higher specificity, the influence of the number of irrelevant attributes is more significant in the setting with a mutation rate of $\mu = 0.01$. Using a smaller population size can delay or stall the evolutionary process due to the reproductive opportunity bound.

Figure 5.10 shows the behavior of the specificities of the six relevant positions and several of the other positions (that all behave similarly). It can be seen that complete convergence is delayed if population size is set not high enough. The reproductive opportunity bound slowly comes into play. With a string length of more than 150, evolution partially stalls completely

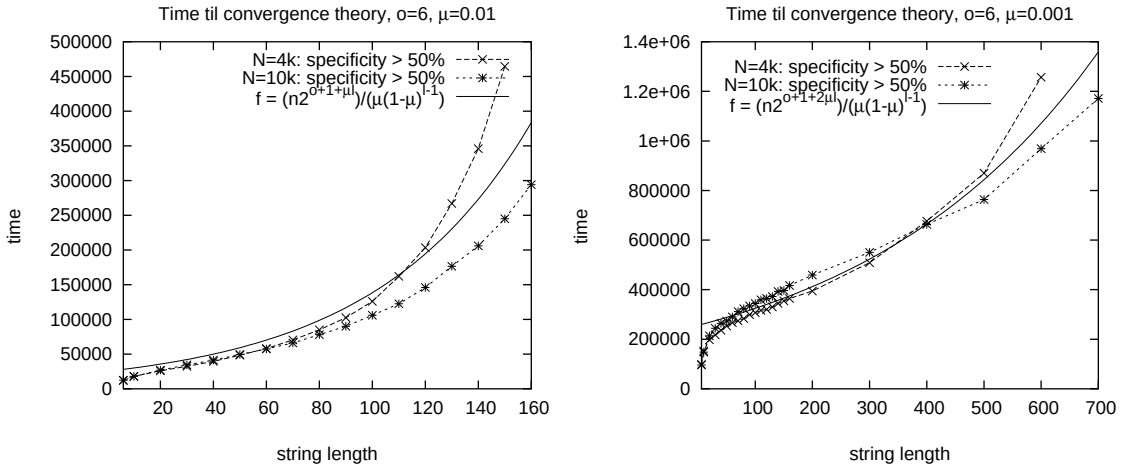


Figure 5.9: The influence of problem length is properly bound by the theory. The reproductive opportunity bound increasingly outweighs the time bound when the population size is set too low.

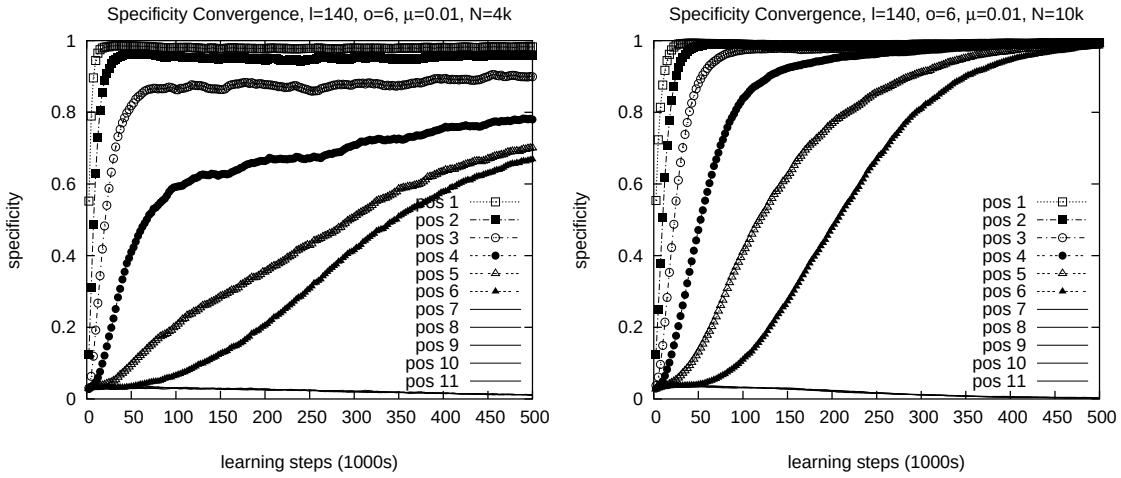


Figure 5.10: Increasing the problem length further, the reproductive opportunity bound delays or even stalls convergence when population size is set too low and mutation rate too high.

since the over-specialized irrelevant attributes prevent sufficient reproductive opportunities. With a higher population size, the reproductive opportunity bound vanishes and all six specificities converge to one without delay. Similarly, Figure 5.11 shows that when decreasing string length or mutation rate, the time bound correctly estimates learning time.

In addition to the above bound, we investigate the effects of several parameters and additional mechanisms in XCS. Figure 5.12 reveals dependencies on several XCS mechanisms in the setting with a string length $l = 10$ and the number of relevant attributes $k_d = 4$

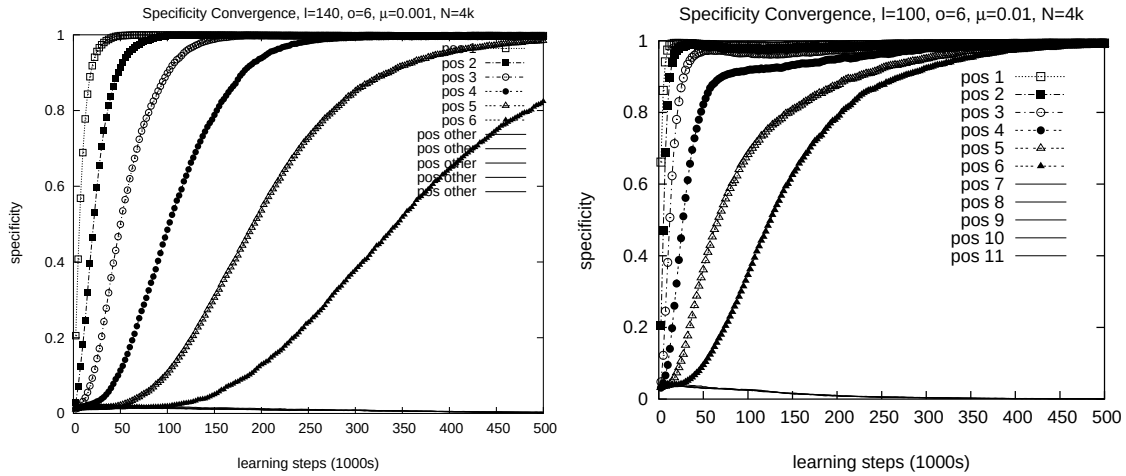


Figure 5.11: Choosing a lower mutation rate or facing a smaller problem length l, the learning time bound becomes most relevant again.

(left-hand side) and $k_d = 8$ (right-hand side). In the setting with four relevant attributes, we can see that the disallowance of action mutation as well as the restriction to niche mutation decreases performance. Allowing action mutation or free mutation, subsolutions in one problem niche can propagate much easier to another niche (by mutating action 1 to 0 or specified attribute 1 to 0 or vice versa). Increasing the GA threshold θ_{GA} delays the evolutionary process. Uniform crossover has an additional beneficial effect enhancing the possibility of knowledge exchange between different niches. Fitness-based selection also slightly speeds-up learning. Due to the fitness decrease in offspring (fitness is set to 10% of the parental fitness), a slight generalization pressure (Bull, 2003) is generated that decreases specificity (in particular the specificity of the irrelevant attributes) and thus facilitates learning. In the setting with eight relevant attributes (right-hand side), the performance decrease due to restricted mutation overshadows the decrease due to a higher GA threshold.

5.5 Assuring Solution Sustenance: The Niche Support Bound

The above bounds assure that problem subsolutions represented by individual classifiers evolve. The time bound additionally estimates how long the evolution takes. Since the time bound and all other bounds consider individual classifiers integrated in the whole population, the population as a whole is required to evolve a complete problem solution supplying, evaluating, and growing currently best subsolutions in parallel. What remains to be assured

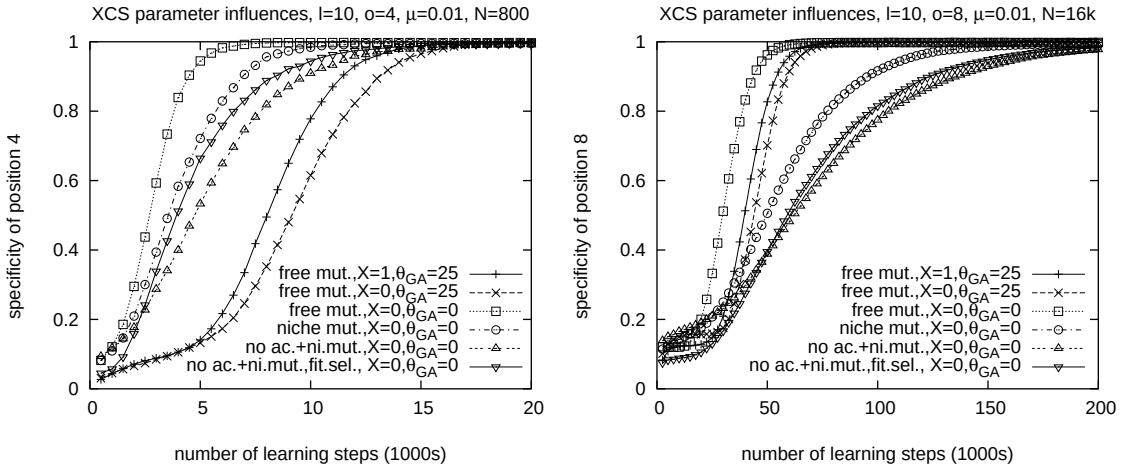


Figure 5.12: Several mechanisms influence the learning speed: Action mutation and free mutation facilitate the additional knowledge exchange between different problem niches and subsolutions. A higher GA threshold delays evolution especially given a more generalized population. Fitness-based selection slightly speeds-up learning. More relevant attributes and thus a more specialized population show similar performance influences (right-hand side).

is that the final problem solution, represented by a set of maximally accurate and maximally general classifiers, can be sustained in the population. This is expressed in the sixth point of the facetwise theory approach to LCS success: Niching techniques need to assure the sustenance of a complete problem solution.

Thus, we now derive a population size bound that assures the niche support of all subsolutions in the problem with high probability. To derive the niche support bound, we develop a simple Markov chain model of classifier support in XCS. Essentially, we model the change in niche size of particular problem subsolutions (that is, niches) using a Markov chain.

To derive the bound, we focus on the support of one niche only disregarding potential interactions with other niches. Again we assume that problem instances are encountered according to a uniform distribution over the whole problem space. Additionally, we assume random deletion from a niche. Given the Markov chain over the niche size, we then determine the steady state distribution that estimates the expected niche distribution.

Using the steady state distribution, we derive the probability that a niche is lost. This probability can be bounded minimizing the loss probability resulting in a final population size bound. The bound assures the maintenance of a low-error solution with high probability. The experimental evaluations show that the assumptions hold in non-overlapping problems. In problems that require overlapping solution representations, the population size may need to be increased further.

5.5.1 Markov Chain Model

As already introduced for the schema bound in Section 5.2, we define a problem niche by a schema of order k. A representative of a problem niche is defined as a classifier that specifies at least all k bits correctly. The Markov chain model constructs a Markov chain over the number of classifier representatives in a particular problem niche.

Suppose we have a particular problem niche represented by k classifiers; let p be the probability that an input belonging to the niche is encountered, and let N be the population size. Since classifiers are deleted with probability proportional to their action set size estimate, a classifier will be deleted from the niche with probability k/N. Assuming that the GA is always applied (this can be assured by setting $\theta_{GA} = 0$) and disregarding any disruptive effects due to mutation or crossover, the probability that a new classifier is added to the niche is exactly equal to the probability p that an input belonging to the niche is encountered.

However, overgeneral classifiers might inhabit the niche as well so that also an overgeneral classifier might be chosen for reproduction decreasing the reproduction probability p of a niche representative. As shown in Chapter 4, though, due to the action set size relative tournament selection, the probability of selecting a niche representative for reproduction is larger than some constant dependent on the relative tournament size τ. Given that τ is chosen sufficiently large and given further that the population mainly converged to the niche representatives, the probability approaches one.

In the Markov chain model, we assume that at each time step both the GA reproduction and the deletion are applied. Accordingly, we derive three transition probabilities for a specific niche. Given the niche is currently represented by j classifiers, at each time steps, (i) with probability r_j the size of the niche is increased (because a classifier has been reproduced from the niche, while another classifier has been deleted from another niche); (ii) with probability d_j the size of the niche is decreased (because genetic reproduction took place in another niche, while a classifier was deleted from this niche); (iii) with probability s_j the niche size remains constant (either because no classifier has been added nor deleted from the niche or because one classifier has been added to the niche while another one has been deleted from the same niche).

The Markov chain associated to the model is depicted in Figure 5.13. States in the model indicate the niche size determined by the number of representatives in a niche. Arcs labeled with r_j represent the event that the application of the GA and deletion results in an increase of the niche size. Arcs labeled with s_j represent the event that the application of the genetic

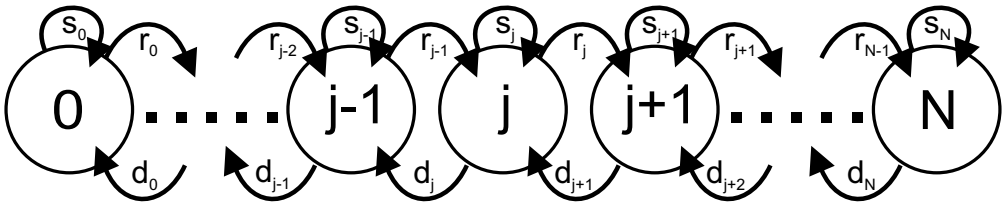


Figure 5.13: Markov chain model for the support of one niche in XCS: j is the number of classifiers in the niche; N is the population size; r_j is the probability that a classifier is added to a niche containing j representatives; s_j is the probability that the niche containing j representatives is not modified through reproduction; d_j is the probability that a classifier is deleted from a niche containing j representatives.

algorithm and deletion results in no overall effect on the niche size. Arcs labeled with d_j represent the event that the application of the genetic algorithm and deletion results in a decrease of the niche size.

More formally, since the current problem instance is part of a particular niche with probability p, a niche representative will be generated via GA reproduction with approximately probability p. Assuming random deletion, a representative of a niche is deleted with probability j/N since there are by definition j representatives in the current population of size N. Accordingly, we compute the probabilities r_j, s_j, and d_j as follows:

$$r_j = p\left(1 - \frac{j}{N}\right) \tag{5.35}$$

$$s_j = (1-p)\left(1 - \frac{j}{N}\right) + p\frac{j}{N} \tag{5.36}$$

$$d_j = (1-p)\frac{j}{N} \tag{5.37}$$

For $j = 0$ we have $r_0 = p$, $s_0 = 1 - p$, and $d_0 = 0$. When $j = 0$, the niche is not represented in the population, therefore: (i) when an input belonging to the niche is presented to the system (with probability p), one classifier is generated through *covering*, therefore $r_0 = p$; (ii) since the niche has no classifiers, deletion cannot take place, therefore $d_0 = 0$; finally, (iii) the probability that the niche remains unrepresented is $1 - r_0 - s_0$, that is $s_0 = 1 - p$. Similarly, when $j = N$ all the classifiers in the population belong to the niche, accordingly: (i) no classifier can be added to the niche, therefore $r_N = 0$; (ii) with probability p an input belonging to the niche is encountered so that a classifier from the niche is reproduced while another one from the niche is deleted, leaving the niche size constant, therefore $s_N = p$; (iii) when an input that does not belong to the niche is presented to the system (with probability

$1-p$), a classifier is deleted from the niche to allow the insertion of the new classifier to the other niche, therefore $d_N = 1-p$. Thus, for $j = N$ we have $r_N = 0$, $s_N = p$, and $d_N = 1-p$.

Note that our approach somewhat brushes over the problem of overgeneral classifiers in that overgeneral classifiers are not considered as representatives of any niche. In addition, covering may not be sufficient in the event of an empty niche since overgeneral classifiers might still be present so that $r_0 = p$ is an approximation. However, as pointed out in Horn (1993), as long as a sufficiently large population size is chosen, chopping off or approximating the quasi absorbing state r_0 approximates the distribution accurately enough. This is also confirmed by our experimental investigations. However, overgeneral classifiers and more importantly overlapping classifiers can influence the distribution as evaluated below. Given the above assumptions, we are now able to derive a probability distribution over niche support.

5.5.2 Steady State Derivation

To estimate the distribution over the number of representatives of a problem niche, we derive the distribution when the Markov chain is in steady state. Essentially, we derive probabilities u_j that the niche has j representatives. To derive the steady state distribution, we first write the fixed point equation for our Markov chain:

$$u_j = r_{j-1}u_{j-1} + s_j u_j + d_{j+1}u_{j+1},$$

which equates the probability that the niche has j representatives with the probability that the niche will have j representatives in the next time step. In the next time step, three events will contribute to the probability of having j representatives: (i) reaching state j from state $j-1$, with probability $r_{j-1}u_{j-1}$, by means of a reproductive event; (ii) remaining in state j with probability $s_j u_j$; (iii) reaching state j from state $j+1$, with probability $d_{j+1}u_{j+1}$, by means of a deletion event. The same equation can be rewritten by acknowledging the fact that in steady state the incoming proportion needs to be equal to the outgoing proportion in each state in the Markov chain:

$$(r_j + d_j)u_j \quad = \quad r_{j-1}u_{j-1} + d_{j+1}u_{j+1}. \tag{5.38}$$

Replacing d_{j+1}, d_j, r_j, and r_{j+1} with the actual values from the previous section (equations 5.35, 5.36, and 5.37) we get

$$\left[p\left(1-\frac{j}{N}\right)+\frac{j}{N}(1-p)\right]u_j=(1-p)\left(\frac{j+1}{N}\right)u_{j+1}+p\left(1-\frac{j-1}{N}\right)u_{j-1}. \quad (5.39)$$

Equation 5.39 is a second order difference equation whose parameters are dependent on j, i.e., on the current state. We use Equation 5.39 and the condition:

$$\sum_{j=0}^{j=N} u_j = 1 \quad (5.40)$$

to derive the steady state distribution. Multiplying Equation 5.39 by $N/((1-p)u_{j-1})$, we derive the following:

$$\left[\frac{p}{1-p}(N-j)+j\right]\frac{u_j}{u_{j-1}}=(j+1)\frac{u_{j+1}}{u_{j-1}}+\frac{p}{1-p}(N-j+1). \quad (5.41)$$

To derive the steady state distribution of probabilities u_j we use Equation 5.41 to derive an equation for the ratio $\frac{u_j}{u_0}$. Next, we use the equation for $\frac{u_j}{u_0}$ and the condition in Equation 5.40 to derive the steady state distribution.

As the very first step, we write the following fixed point equation for the transition between state 0 and state 1

$$u_0 = s_0u_0 + d_1u_1 \quad (5.42)$$

substituting the values of r_0 and d_1 we obtain:

$$\begin{aligned} u_0 &= (1-p)u_0 + (1-p)\frac{1}{N}u_1 \\ pu_0 &= (1-p)\frac{1}{N}u_1 \end{aligned} \quad (5.43)$$

from which we derive:

$$\frac{u_1}{u_0}=\frac{p}{1-p}N \quad (5.44)$$

To derive the equation for u_2/u_0 we start from Equation 5.41 and set $j = 1$:

$$\left[\frac{p}{1-p}(N-1)+1\right]\frac{u_1}{u_0} = 2\frac{u_2}{u_0} + \frac{p}{1-p}N \tag{5.45}$$

so that,

$$\frac{u_2}{u_0} = \frac{1}{2}\left[\left(\frac{p}{1-p}(N-1)+1\right)\frac{u_1}{u_0}\frac{p}{1-p}N\right] \tag{5.46}$$

We replace u_1/u_0 with Equation 5.44:

$$\begin{aligned}
\frac{u_2}{u_0} &= \frac{1}{2}\left[\left(\frac{p}{1-p}(N-1)+1\right)\frac{u_1}{u_0}\frac{p}{1-p}N\right] \\
&= \frac{1}{2}\left[\left(\frac{p}{1-p}(N-1)+1\right)\frac{p}{1-p}N\frac{p}{1-p}N\right] \\
&= \frac{1}{2}\left[\frac{p}{1-p}(N-1)\frac{p}{1-p}N\right] \\
&= \frac{N(N-1)}{2}\left(\frac{p}{1-p}\right)^2 \\
&= \binom{N}{2}\left(\frac{p}{1-p}\right)^2
\end{aligned} \tag{5.47}$$

This leads us to the hypothesis that

$$\frac{u_j}{u_0} = \binom{N}{j}\left(\frac{p}{1-p}\right)^j, \tag{5.48}$$

which we prove by induction. Using Equation 5.48, we can first derive that

$$u_{j+1} = \frac{N-j}{j+1}\frac{p}{1-p}u_j \tag{5.49}$$

$$u_j = \frac{N-j+1}{j}\frac{p}{1-p}u_{j-1} \tag{5.50}$$

$$u_{j-1} = \frac{1-p}{p}\frac{j}{N-j+1}u_j. \tag{5.51}$$

With Equation 5.41 substituting Equation 5.51 as the inductive step, we now derive

$$\begin{aligned} u_{j+1} &= \frac{\left(\left(\frac{p}{1-p}(N-j)+j\right)\frac{u_j}{u_{j-1}} - \frac{p}{1-p}(N-j+1)\right)u_{j-1}}{j+1} \\ &= \frac{\left(\frac{p}{1-p}\right)^2 \frac{(N-j)(N-j+1)}{j}u_{j-1}}{j+1} \\ &= \frac{N-j}{j+1}\frac{p}{1-p}u_j, \end{aligned} \tag{5.52}$$

which proves the hypothesis.

We can now derive the steady state distribution from Equation 5.40 dividing both sides by u_0

$$\sum_{j=0}^{N} \frac{u_j}{u_0} = \frac{1}{u_0}, \tag{5.53}$$

substituting Equation 5.48 we derive

$$\begin{aligned} \sum_{j=0}^{N} \frac{u_j}{u_0} &= \sum_{j=0}^{N} \binom{N}{j} \left(\frac{p}{1-p}\right)^j \\ &= \left[\sum_{j=0}^{N} \binom{N}{j} p^j (1-p)^{N-j}\right] \frac{1}{(1-p)^N}, \end{aligned} \tag{5.54}$$

where the term $\sum_{j=0}^{N} \binom{N}{j} p^j (1-p)^{N-j}$, equals to $[p+(1-p)]^N$", that is 1, so that:

$$\sum_{j=0}^{N} \frac{u_j}{u_0} = \frac{1}{(1-p)^N}, \tag{5.55}$$

accordingly,

$$u_0 = (1-p)^N \tag{5.56}$$

Finally, combining Equation 5.48 and Equation 5.56, we derive the steady state distri-

bution over u_j as follows:

$$\begin{aligned} u_j &= \binom{N}{j}\left(\frac{p}{1-p}\right)^j u_0 \\ &= \binom{N}{j}\left(\frac{p}{1-p}\right)^j (1-p)^N \\ &= \binom{N}{j} p^j (1-p)^{N-j}. \end{aligned} \tag{5.57}$$

Note that the same derivation is possible noting that the proposed Markov chain results in an *Engset* distribution (Kleinrock, 1975).

Essentially, we see that the constant probability of reproduction p in combination with a linear increasing probability of deletion j/N, results in a binomial distribution over niche support sizes in steady state. In the next sections we validate Equation 5.57 experimentally, and evaluate the assumptions made such as the influence of mutation, overgeneral classifiers, the r_0 approximation, and overlapping niches.

5.5.3 Evaluation of Niche Support Distribution

Our evaluation focuses on three Boolean function problems introduced in Appendix C: (1) the layered count ones problem; (2) the multiplexer problem; (3) the carry problem. While the layered count ones problem requires a non-overlapping solution representation, the multiplexer problem allows overlapping subsolutions. The carry problem requires an overlapping solution representation. Apart from the effect of overlapping solutions, we also evaluate the influence of proportionate selection and tournament selection on the support distribution as well as several parameter influences.

Layered Count Ones Problem

In the layered count ones problem, each niche is represented by specializing a subset of k relevant attributes. There are 2^k such non-overlapping subsets. Since we generate problem instances uniformly randomly, the probability p of reproduction in a particular subset equals $p = 1/2^k$. We evaluate the theory on the layered count ones problem with $k = 5$, $l = 11$. We run XCS for $100,000$ steps to assure that the problem is learned completely. To evaluate the effects of mutation and crossover, we ran additional runs with $50,000$ subsequent condensation steps, in which GA reproduction and deletion are executed as usual but neither mutation nor crossover are applied (Wilson, 1995). Other parameters are set to the standard

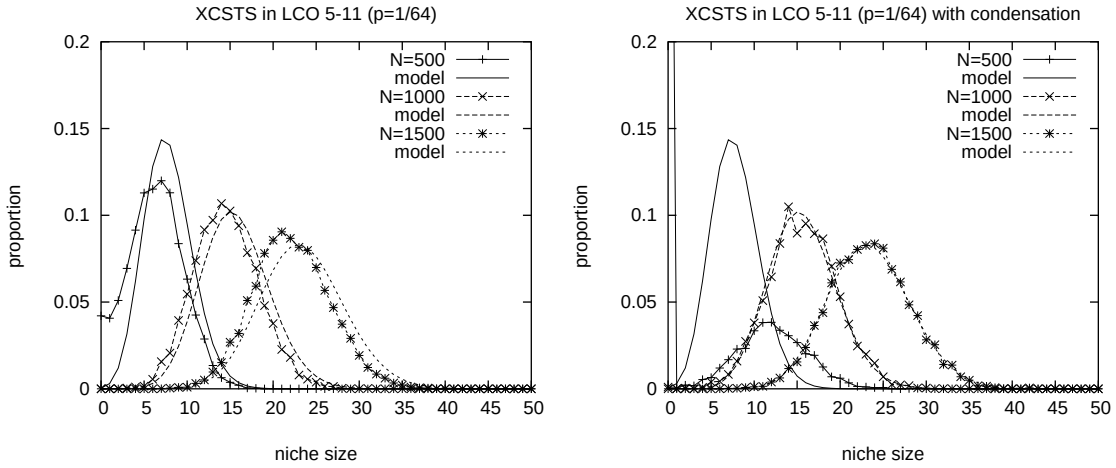


Figure 5.14: The niche distribution is closely matched by the theory. Eliminating the influences of mutation and crossover results in a nearly exact match.

values introduced above. The niche size distribution is derived from all 2^k niche support sizes of each of the 1000 experimental runs.

Applying tournament selection and continuous GA application ($\theta_{GA} = 0$), Figure 5.14 (left-hand side) shows the resulting distributions without condensation varying the population sizes and Figure 5.14 (right-hand side) shows the resulting distributions when additional $50,000$ condensation steps were executed. Without condensation, the actual distributions are slightly over-estimated by the theory, that is, the theory predicts larger niche support. This is explainable by the continuous application of mutation and crossover that causes the reproduction of overgeneral classifiers that are not representing any niche. When condensation is applied and niche-loss is prevented by a sufficiently large population size, the theory agrees nearly exactly with the experiments. If the population size is chosen too low ($N = 500$) niches are continuously lost and cannot be recovered once condensation is applied. Thus, the niches that were not lost have an on average higher support but a large fraction of niches is lost. Applying proportionate selection, the resulting distributions are nearly identical to the one with tournament selection (Figure 5.15).

We made several assumptions in the theory including the continuous application of the GA (assured by setting $\theta_{GA} = 0$). To observe the effect of θ_{GA}, we set the parameter to 200 and re-ran the experiments. Figure 5.16 shows the consequent distributions. It can be seen that the distributions have a smaller deviation effectively focusing on the mean. Due to the smaller deviation, the probability of loosing a niche is significantly decreased as indicated by the maintenance of an appropriate support distribution even with low population size

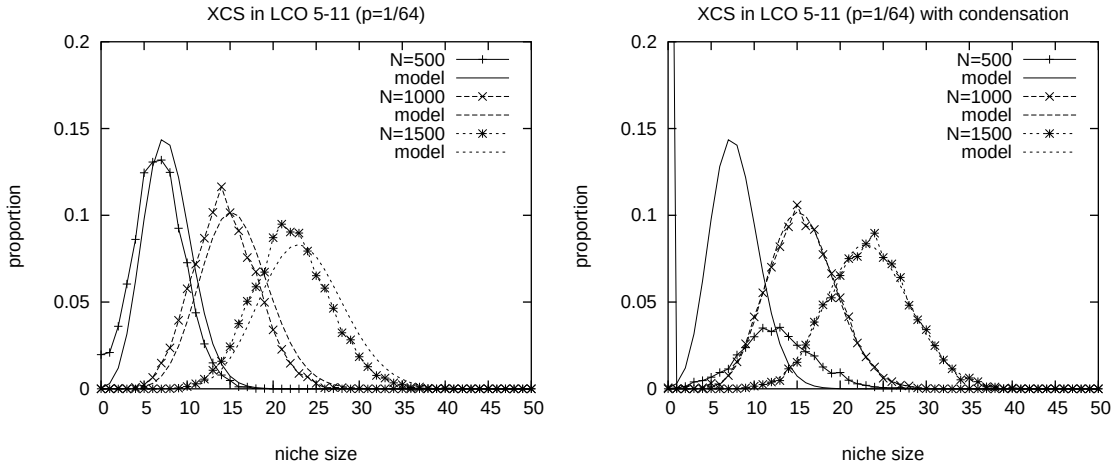


Figure 5.15: Applying proportionate selection, the niche size distribution is hardly influenced in comparison to tournament selection as long as the problem requires a non-overlapping classifier population.

(Figure 5.16, right-hand side, $N = 500$). Given a problem niche is sampled by several problem instances in succession, the threshold effectively prevents an over-reproduction in the niche by preventing GA reproduction. Thus, the GA threshold prevents the frequent reproduction in one problem niche and consequently causes a further balance in the niche size distribution. In essence, our Markov-chain approximation does not hold anymore when $\theta_{GA} > 0$ because the probability of reproduction in a niche now depends also on the recent history of reproductive events and not only on the probability of niche occurrence. The result is the prevention of niche over-reproduction and consequently also the further prevention of niche loss. Thus, the niche size distribution has a smaller variance but the mean stays approximately the same. If niche loss if prevented due to the lower variance, a very significant distribution change is encountered.

Our model assumes deletion proportional to the current niche size. This is approximated by the actual deletion which is proportional to the current action set size estimates as. The estimate as coincides with the actual current niche size if all niches are represented by exactly one classifier (in which case the niche size equals the numerosity of the classifier). However, niche-size proportionate deletion does not take into account the case when a niche is represented by several different classifiers (including over-specialized ones) nor the influence of overgeneral classifiers. In this case, the action set size estimate based deletion is not exactly niche-size proportionate anymore. Thus, we conducted runs in which random selection was applied. The resulting distributions shown in Figure 5.17 indicate that the estimate as-based deletion results in additional stabilization of the support distribution. If random deletion is

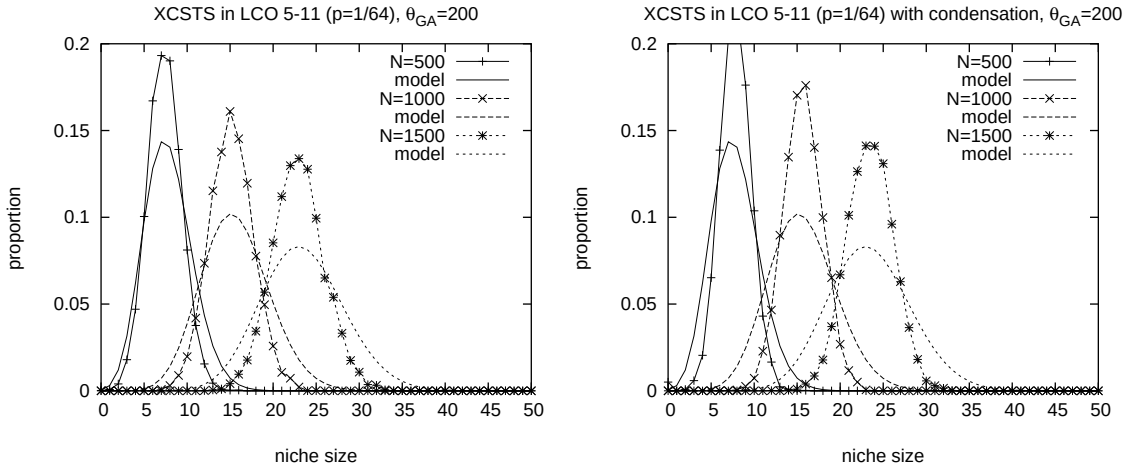


Figure 5.16: Increasing the GA threshold results in a more balanced distribution causing an overall decrease in deviation from the expected distribution mean.

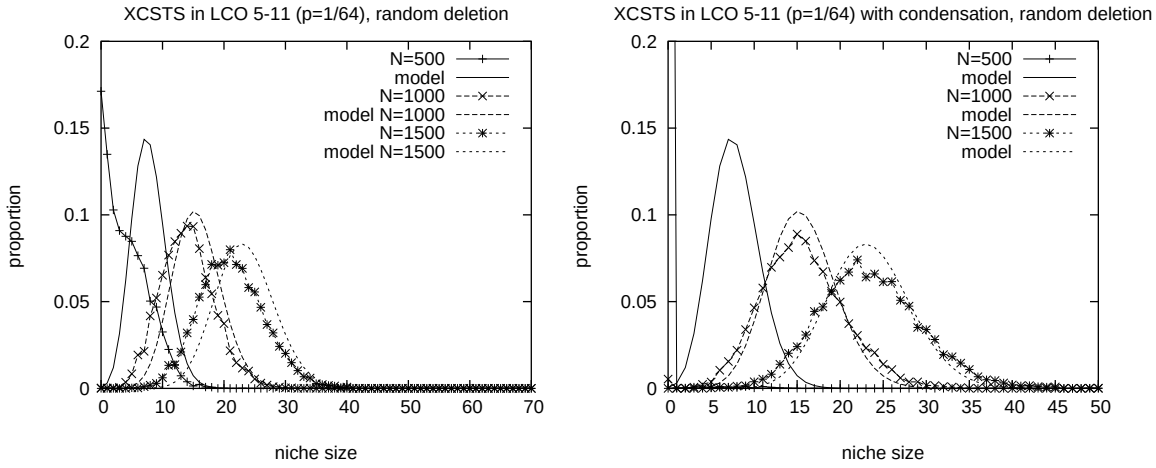


Figure 5.17: Random deletion results in further niche loss due to potentially disruptive overgeneral as well as over-specialized classifiers.

applied, further niche loss occurs in comparison to Figure 5.14. The result indicates that the action set size based deletion causes further stabilization since overgeneral and over-specialized classifiers tend to occur in larger niches (resulting in a larger action set size estimate). Consequently, the action set size estimate-based deletion results in a further focus on the accurate, maximally general classifiers and thus a more stable niche support distribution.

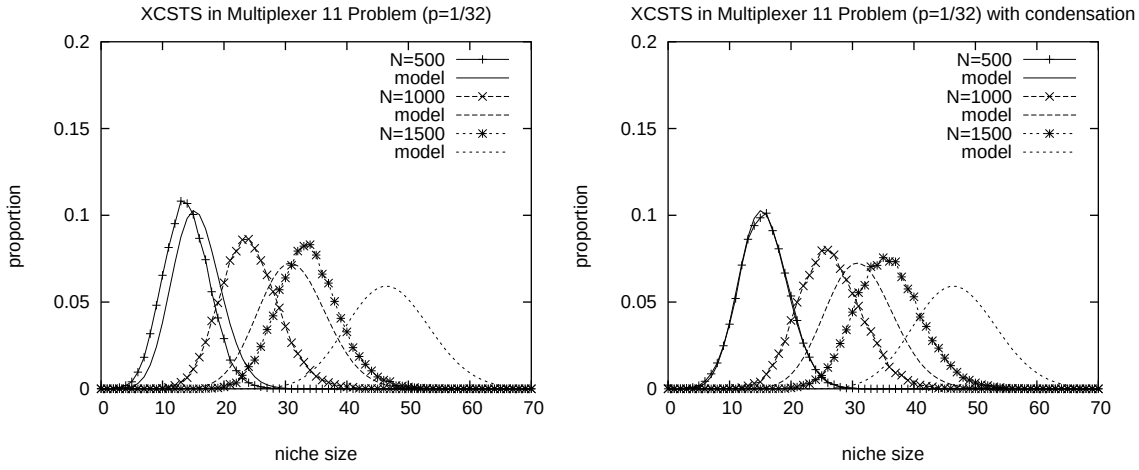


Figure 5.18: Due to the potentially overlapping solution in the multiplexer problem, niche sizes remain slightly smaller than predicted.

Multiplexer Problem

The multiplexer problem is of interest with respect to niche support because there exists a non-overlapping solution but also overlapping, maximally general classifiers may evolve. For example, in the 6-multiplexer, a complete, accurate, and maximally general subsolution is represented by classifiers cl_1 =000###→0 and cl_2 =00#0##→0. However, there is another classifier that is as general and as accurate as these two but overlaps with both, that is, cl_3 =0#00##→0. Only due to the competition with the two former, the latter classifier can be extinct. The competition takes place due to the fitness sharing mechanism as well as the shared reproductive opportunities when an overlap occurs.

Figure 5.18 shows the resulting distributions in the 11-multiplexer problem. While the distribution again nearly perfectly coincides with the theory when condensation is applied and the population size is small, larger population sizes result in a skewed distribution. The effect appears to be the mentioned competition in conjunction with tournament selection. While competition is high when the population size is small ($N = 500$), the overlapping classifiers are eventually discarded from the population converging to the optimal, accurate, maximally general, and non-overlapping problem solution. However, if a larger population size is chosen, the overlapping classifiers are not necessarily extinct so that the niche-size distribution (in which the overlapping classifiers are ignored) decreases. This was also confirmed by hand by an actual investigation of the final population in several runs.

Interestingly, when applying proportionate selection the distributions coincide with the theory in all cases. Figure 5.19 shows that as long as condensation is applied the resulting

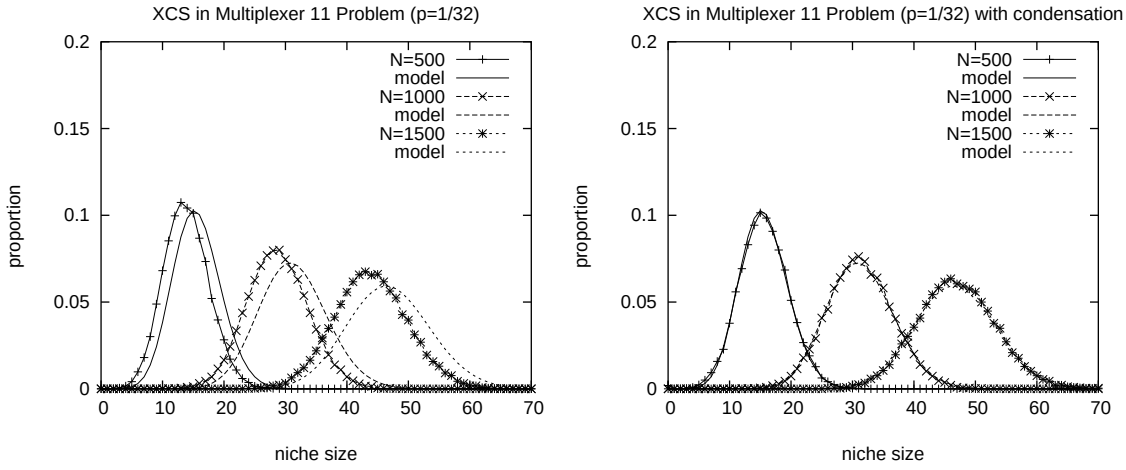


Figure 5.19: In contrast to tournament selection, proportionate selection distinguishes small fitness differences less severely consequently keeping up the competition with overlapping classifiers.

niche size distribution is nearly identical to the theory. The following effect can explain this observation. In tournament selection, slight fitness differences matter whereas in proportionate selection nearly identical fitness result in nearly identical selection probabilities. Thus, while proportionate selection gives the overlapping and non-overlapping classifiers (e.g cl_1 vs. cl_3) approximately the same probability of reproduction, tournament selection prefers the one with the slightly higher fitness. However, since cl_3 always overlaps (either with cl_1 or with cl_2), it should have a lower fitness on average since the niche size will be higher on average. However, there is an additional effect due to classifier deletion. Since fitness is not adjusted when the numerosity of a classifier is decreased (effectively deleting one of the representatives of that macro-classifier), the consequent fitness of the remaining classifiers is effectively increased sharing the fitness of the deleted offspring. Consequently, the overlapping classifier will have higher fitness if it underwent deletion numerous times thus preventing its loss. The result is the observed on-average smaller niche distribution than expected by the theory since overlapping classifiers remain in the population but are not considered as niche representatives in the graphs.

Carry Problem

The carry problem is a typical problem in which overlapping classifiers are evolved to generate a complete solution. Additionally, the niche sizes of the overlapping classifiers differ. That is, the probability of a niche occurrence p differ. Our investigations consider the still very

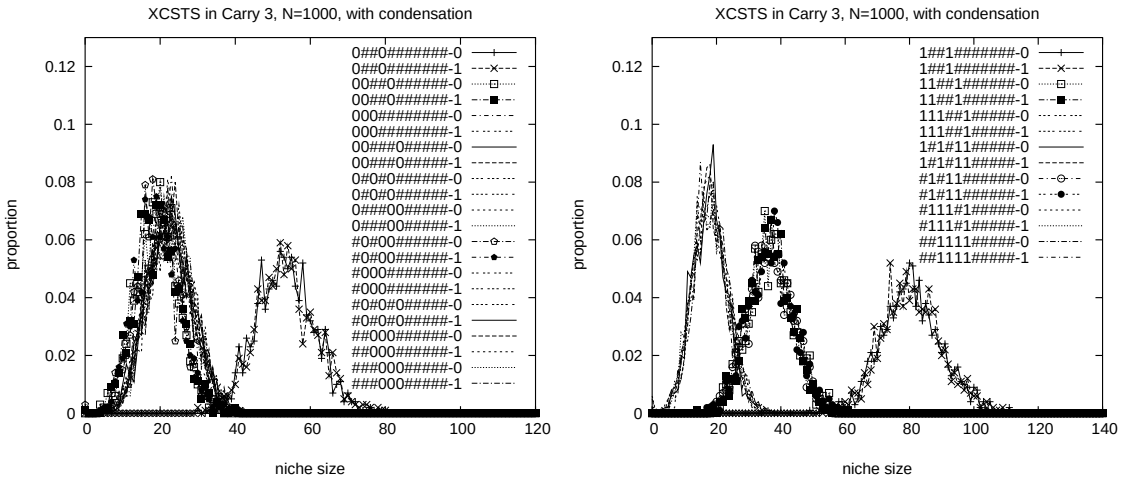


Figure 5.20: The overlapping, differently specific niches in the carry-3 problem cause interference. This makes it hard to predict the final niche size distribution accurately.

small carry-3 problem. We add five irrelevant bits so that the problem is as large as the eleven multiplexer problem. In this problem two binary numbers of length three are added. If the result triggers a carry, then the class is one and zero otherwise. For example, to assure that a carry occurs, it is sufficient to assure that the highest two bits are set to one, that is, `1##1##`→`1` is a completely accurate and maximally general classifier. However also two ones at the second positions and at least one one at the first suffice to assure the event of a carry, that is, classifier `11##1#`→`1` is also completely accurate and maximally general but it overlaps with the first classifier in the `11*11*` subspace. This partial competition also influences the niche distribution.

Figure 5.20 shows the resulting niche distributions of all 36 niches in the problem. Note how the zero niches (left-hand side) have a lower frequency than the correspondingly specific one niches (e.g. `0##0#######` vs. `1##1#######`) due to the higher overlap with other competitors. Note also how the classifiers that specify three zeros in the two higher bits of the two numbers have a slightly smaller distribution than the other classifiers that specify three zeros. Again, the additional competition due to more overlaps in the former is responsible for the effect. With condensation (Figure 5.21), we can see again how the balanced GA application results in a niche size distribution with smaller deviation. The distributions stay closer to the mean consequently preventing niche loss more effectively.

Due to the overlapping final solution, it is not immediately possible to derive the probability of reproduction for a particular niche. However, it is possible to derive a general probability of reproduction for a subset of niches that are non-overlapping with the other

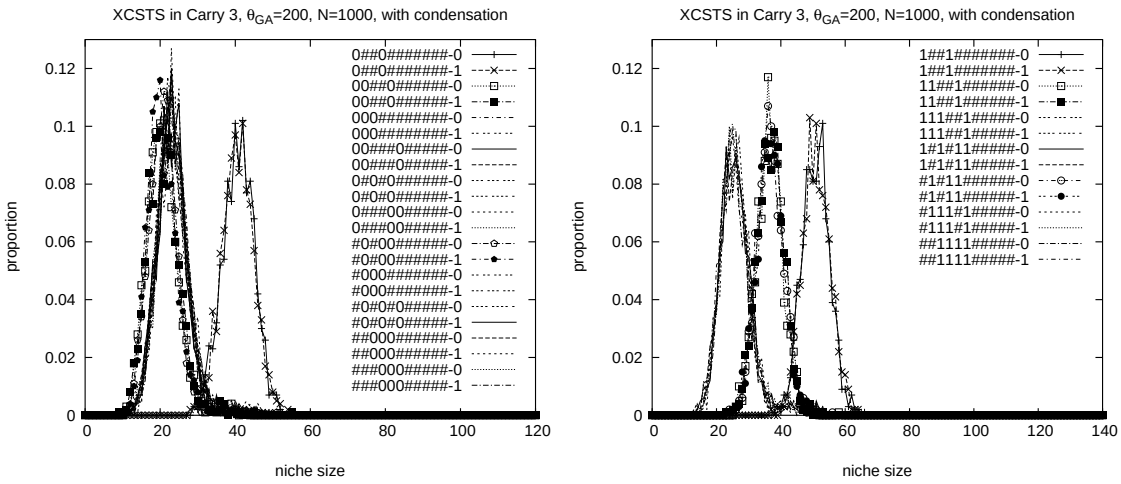


Figure 5.21: Also in the carry-3 problem, which requires an overlapping final solution representation, a higher GA threshold results in a niche size distribution with smaller deviation.

niches. In the carry problem, the zero niches clearly do not overlap with the one niches so that the overall distribution of the zero and one niches can be estimated with our theory. The overall reproductive probability p for all representatives of all one niches can be estimated by summing the probability that the highest bits are both one (0.25), plus the event that both second highest bits but only one of the highest bits are one (0.125), plus the event that both third-highest positions are one and only one of the two higher positions is one (0.0625). In sum, the result equals 0.4375. Figure 5.22 shows in the overall distribution. It appears that the action set size based deletion pushes the distribution slightly closer to each other which is also confirmed in the runs with a higher GA threshold (Figure 5.23). Further conducted results with proportionate selection did not suggest any significant differences (not shown).

In sum, we saw that solution spaces interfere with each other during selection in problems that require an overlapping solution representation. The overlap causes a decrease in niche sizes. However, the influence is not as significant as originally feared. Further extensions to balance the niches are imaginable such as taking into consideration the degree of overlap among competing (fitness sharing) classifiers. Nonetheless, the model is able to predict the general behavior of XCS's final solution. Additionally, the model can be used to estimate the probability of a niche loss. The next section derives this probability and extends the model to a general population size bound that ensures the maintenance of a low-error solution with high probability.

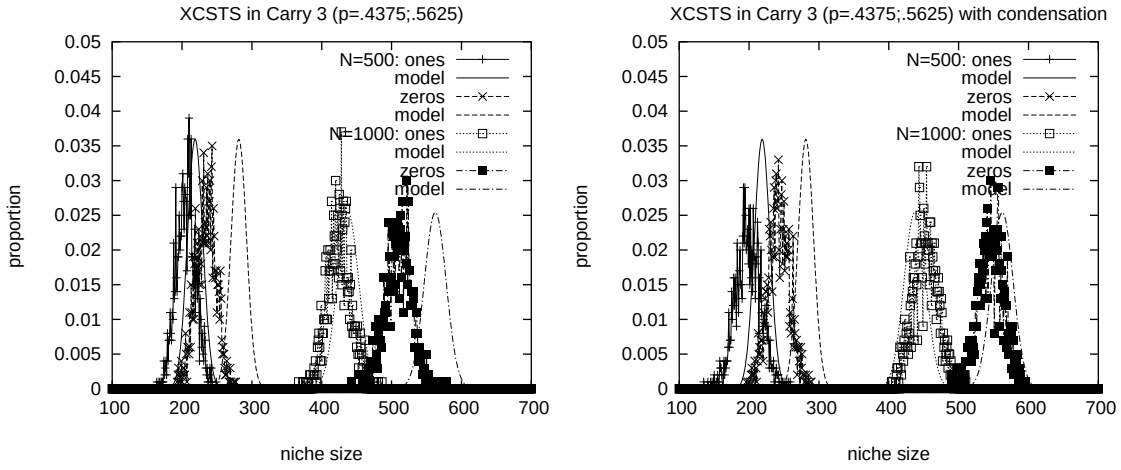


Figure 5.22: Combining all zero and all one niches and comparing the resulting (macro-)distributions shows that the model applies again. The action set size based deletion in conjunction with the overlapping nature of the problem cause the two distributions to move slightly closer together. As before, condensation eliminates overgeneral classifiers and thus focuses the population on niche representatives.

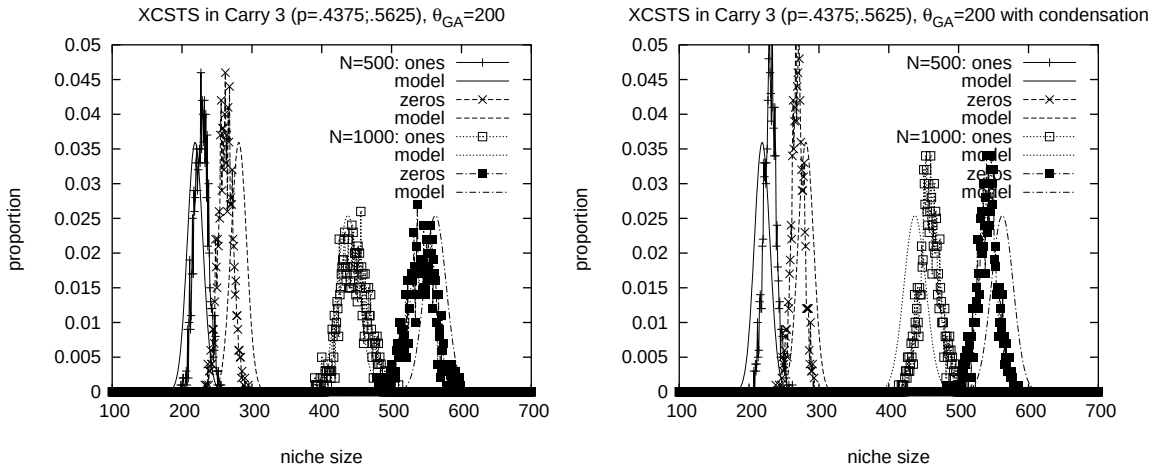


Figure 5.23: A higher GA threshold moves the lower and upper distributions closer together and slightly focuses the population.

5.5.4 Population Size Bound

The reported results show that our model for niche support in XCS can predict the distribution of niche size for problems involving non-overlapping niches. Effectively, our model provides an asymptotic prediction of niche support distribution. It applies once the problem has been learned and there is no further influence from genetic operators. Besides such predictive capabilities, we can use our model to derive a population size bound that ensures that

a complete model is maintained with high probability. In particular, from the probability u_0 we can derive a bound to guarantee that a niche is not lost with high probability.

In essence, the model can approximate the probability that a niche is lost. Using Equation 5.57, we can derive a bound for the population size N that ensures with high probability that XCS does not loose any of the problem niches, that is, any subsolutions. From the derivation of the probability of being in state u_0 (which means, that the respective niche was lost), which is $u_0 = (1-p)^N$, we see that the probability of loosing a niche decreases exponentially with the population size. Given a problem with 2^k problem niches, that is, the perfect solution $[O]$ is represented by 2^k schemata of order k, the probability of loosing a niche equates $u_0 = \left(1 - \frac{1}{2^k}\right)^N$.

Requiring a certainty θ that no niche will be lost (that is, $\theta = 1 - u_0$), we can derive a concrete population size bound

$$N > \frac{\log(1-\theta)}{\log(1-p)} > \frac{\log(1-\theta)}{\log(1-\frac{1}{2^k})}, \tag{5.58}$$

effectively showing that population size N grows logarithmically in the confidence value and polynomially in the solution complexity 2^k. Figure 5.24 shows the population size bound that assures niche support. Since the population size scales as the inverse of the probability of niche occurrence, the $\log - \log$-sale shows a straight line.

Thus, the bound confirms that once a problem solution was found, XCS is able to maintain the problem solution with high probability requiring a population size that grows polynomially in solution complexity and logarithmically in the confidence value. This bound confirms that XCS does not need more than a polynomial population size with respect to the solution complexity consequently pointing to the PAC learning capability of XCS confirming Wilson's original hypothesis (Wilson, 1998).

5.6 Towards PAC Learnability

The derivations of the problem bounds in the previous sections enables us to connect learning in the XCS classifier system to fundamental elements of computational learning theory (COLT). COLT is interested in showing how much computational power an algorithm needs to learn a particular problem. To derive an overall computational estimate of XCS's learning capabilities, we focus on the problem of learning *k-DNF* functions. In particular, we show

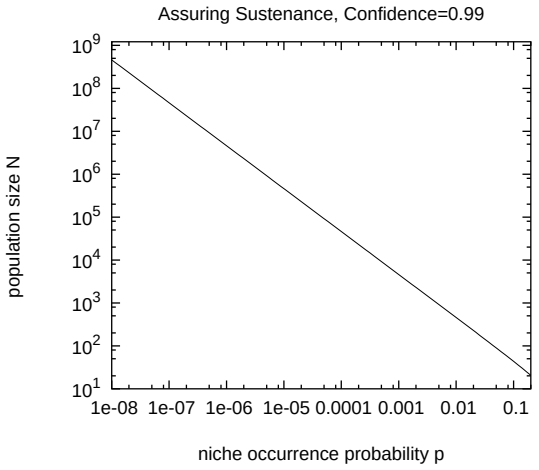


Figure 5.24: To sustain a complete solution with high probability, population size needs to grow as the inverse of the niche occurrence probability p. In a uniformly sampled, binary problem, p corresponds to $1/2^k$ where k specifies the minimal number of attributes necessary to do maximally accurate predictions in all solution subspaces.

that *k-DNF* problems that satisfy few additional properties are PAC-learnable (Valiant, 1984; Mitchell, 1997) by XCS. In essence, we also confirm Wilson's previous conjecture that XCS scales polynomially in time and space complexity (Wilson, 1998).

XCS is certainly not the most effective *k-DNF* learning algorithm. Servedio (2001) shows that an algorithm especially targeted to solve noise-free uniformly sampled *k-DNF* problems is able to reach a much more effective performance. However, this thesis does not only show that XCS is able to PAC-learn *k-DNF* problems. Rather, it shows that it is able to learn a large variety of problems including nominal and real-valued problems, noisy problems, as well as general RL problems. Restricting the problem to *k-DNF* problems, we can show that XCS is a PAC-learning algorithm confirming the effectiveness as well as the generality of XCS's learning mechanism.

To approach the PAC-learning bound, we reflect on the previous bounds evaluating their impact on computational complexity. The successive chapters provide a variety of evidence for XCS's successful and broad applicability as well as its effective learning and scale-up properties.

5.6.1 Problem Bounds Revisited

In Chapter 4, we analyzed how the evolutionary pressures in XCS bias learning towards the evolution of a complete, maximally accurate, and maximally general problem solution

ensuring *fitness guidance* and *appropriate generalization*. This chapter investigated the requirement on population size and learning time in order to supply better classifiers, make time to detect and grow those better classifiers until the final population is reached, and finally, to sustain the final problem solution with high probability. Satisfying these bounds, we can ensure with few additional constraints that XCS learns the underlying problem successfully.

We now revisit the bounds considering their resulting computational requirement.

Covering Bound. The covering bound ensures that the GA is taking place establishing a covering probability (Equation 5.1). To ensure a high probability of covering, the specificity can be chosen very low by setting the initial specificity (controlled by $P_\#$) as well as mutation rate sufficiently low. Given a fixed specificity that behaves in $O(\frac{n}{l})$ as necessary to supply better classifiers, as derived above (Equation 5.33), the population size can be bounded as follows using the approximation $x < -\log(1-x)$:

$$\frac{-\log(1-P(\texttt{cov.}))}{-\log\left(1-\left(\frac{2-\sigma[P]}{2}\right)^l\right)} < \frac{-\log(1-P(\texttt{cov.}))}{\left(1-\frac{n}{2l}\right)^l} < -\log(1-P(\texttt{cov.}))e^{n/2} < N \qquad (5.59)$$

Thus, to satisfy the covering bound, the population size needs to grow logarithmically in the probability of error and exponentially in the number of problem classes n. With respect to PAC-learnability, the bound shows that to ensure that the GA is successfully applied in XCS with probability $1-\delta_P$ (where $\delta_P = (1-P(\texttt{cov.}))$ the population size scales logarithmically in the error probability δ_P as well as exponentially in the number of problem classes.

Schema Bound. The *schema bound* ensures that better classifiers are available in the population. Given a problem with a minimal order of difficulty k_m and requiring a high probability that representatives of this order are supplied (Equation 5.3), we were able to bound the population size N in Equation 5.5 showing that population size N needs to grow logarithmically in the probability of error δ_P (that is, $\delta_P = 1 - P(\texttt{rep.exists})$) and exponentially in the order of the minimal order complexity k_m given a certain specificity and thus polynomial in concept space complexity.

Reproductive Opportunity Bound. In addition to the existence of a representative, we showed that it is necessary to ensure reproduction and thus growth of such representatives. This is ensured by the general reproductive opportunity bound which was shown to require

a population size growth of $O(l^{k_m})$ (Equation 5.22) with respect to the minimal order complexity k_m of a problem. The reproductive opportunity bound was generated with respect to one niche. However, since XCS is evolving the niches in parallel and the probability of niche occurrence as well as the probability of deletion underly a geometric distribution (memoryless property and approximately equal probabilities), we can assure with high confidence $1-\delta_P$, that all relevant niches receive reproductive opportunities. Thus, we can assure with high probability that lower-order representatives grow leading to higher accuracy within a complexity that is polynomial in the concept space complexity.

Time Bound. The time bound estimates the number of problem instances necessary to learn a complete problem solution using XCS. Given a problem that requires classifiers of maximal schema order k_d (a k_d-conjunction in a *k-DNF*) and given further the domino convergence property in the problem, the expected time until generation of an optimal classifier was approximated in Equation 5.32 yielding a population size requirement of $O(l2^{k_d+n})$ (Equation 5.34). The estimation approximates the expected time til creation of the best classifier of a problem niche of order k_d. As in the reproductive opportunity bound, we can argue that since XCS evolves all problem niches in parallel and since the generation of the next best classifier underlies a geometric distribution (memoryless property and equal probability), given a certain confidence value δ, the time until a particular classifier of order k is generated with high confidence δ grows within the same limits. Similarly, assuming a probability p of problem occurrence, we can bound the time requiring a maximal error in the final solution ϵ with low probability δ by $O(l\frac{1}{\epsilon}2^n)$.

Niche Support Bound. To ensure the support of the final solution, we finally established the niche support bound. Given a problem whose solution is expressed as a disjunction of distinct *subsolutions*, XCS tends to allocate distinct rules to each *subsolution*. To ensure a complete problem solution, it needs to be assured that all subsolutions are represented with high probability. Deriving a Markov model over the number of representatives for a particular niche, we were able to derive the steady state niche distribution given a niche occurrence probability p (Equation 5.57).

Requiring that all niches with at least niche occurrence probability p are expected to be present in the population with high probability, we were able to derive a bound on the population size N requiring that the probability of no representative in a niche with more than p occurrence probability is sufficiently low. With respect to PAC-learnability this bound requires that with high probability $1-\delta$ we assure that our solution has an error probability

of less than ϵ (where ϵ is directly related to p). Using this notation, we can derive the following equation from Equation 5.56 substituting ϵ for p and δ for u_0 using again the approximation $x < -\log(1-x)$:

$$\frac{\log \delta}{\log(1-\epsilon)} < -\frac{1}{\epsilon} \log \frac{1}{\delta} < N. \tag{5.60}$$

This bound essentially bounds the population size showing that it needs to grow logarithmically in $\frac{1}{\delta}$ and linear in $\frac{1}{\epsilon}$. Approximating ϵ by $(\frac{1}{2})^{k_d}$ assuming a uniform problem instance distribution, we see that to prevent niche loss, the population size needs to grow linearly in the concept space complexity 2^{k_d}.

5.6.2 PAC-Learning with XCS

With the bounds above, we are now able to bound computational effort and number of problem instances necessary to evolve with high probability $(1-\delta)$ a low error ϵ solution of an underlying *k-DNF* problem. Additionally, the *k-DNF* problem needs to be maximally of minimal order of difficulty k_m as discussed in sections 5.2 and 5.3.

Thus, Boolean function problems in *k-DNF* form with l attributes and a maximal order of problem difficulty k_m are PAC-learnable by XCS using the ternary representation of XCS conditions. That is, XCS evolves with high probability $(1-\delta)$ a low error ϵ solution of the underlying *k-DNF* problem in time polynomial in $1/\delta$, $1/\epsilon$, l, and concept space complexity.

The bounds derived in Section 5.6.1 show that the computational complexity of XCS, which is bounded by the population size N, is linear in $1/\delta$, $1/\epsilon$ and l^{k_m}. Additionally, the time bound shows that the number of problem instances necessary to evolve a low-error solution with high probability grows linearly in $1/\delta$, $1/\epsilon$, l and the solution complexity. Consequently, we showed that Boolean functions that can be represented in *k-DNF* and have a maximal order of problem difficulty k_m, are PAC-learnable by XCS using the ternary representation of XCS conditions as long as the assumptions in the bound derivations hold.

The following further assumptions about the interaction of the bounds have been made. First, crossover is not modeled in our derivation. While crossover can be disruptive as already proposed in Holland (1975), crossover may also play an important innovative role in recombining currently found subsolutions effectively as proposed in Goldberg (2002) and experimentally confirmed for XCS in Butz, Goldberg, and Tharakunnel (2003). The next chapter provides a more detailed investigation on the impact of crossover.

Second, the specificity derivation from the mutation rate assumes no actual fitness influ-

ence. Subtle interactions of various niches might increase specificity further. Thus, problems in which the specificity assumption does not hold might violate the derived reproductive opportunity bound. The experimental investigations in Chapter 7 on several Boolean functions provide further evidence on the influence of specificity and the resulting learning time and population size requirements.

Third, if the probability of reproduction p is approximated by $(\frac{1}{2})^{k_d}$, niche support assumes a non-overlapping representation of the final solution. Thus, overlapping solution representations require an additional increase in population size as evaluated in Section 5.5.

5.7 Summary and Conclusions

This chapter showed *when* XCS is able to learn a problem. Along our facetwise theory approach to LCSs, we derived population size, specificity, and time bounds that assure that a complete, maximally accurate, and maximally general problem solution can evolve and can be sustained.

In particular, we derived a *covering bound* that bounds population size and specificity to ensure proper XCS initialization making way for classifier evaluation and GA application. Next, we derived a *schema bound* that bounds population size and specificity to ensure *supply* of better classifiers. Better classifiers were defined as classifiers that have higher accuracy on average. They can be characterized as *representatives* of minimal order schemata or BBs—those BBs, that increase classification accuracy in the problem at hand. Next, we derived a *reproductive opportunity bound* that bounds specificity and population size to assure *identification* and *growth* of better classifiers. The subsequently derived *time bound* estimates the learning time needed to evolve a complete problem solution given the other bounds are satisfied. Finally, we derived a *niche bound* that assures the *sustenance* of a low-error solution with high probability.

Along the way, we defined two major problem complexities: (1) the minimal order complexity k_m, which specifies the minimal number of features that need to be specified to decrease class entropy (that is, increase classification accuracy), and (2) the general problem difficulty k_d, which specifies the maximal number of attributes necessary to specify a class distribution accurately. While the former is relevant for supply and growth, the latter is relevant for the sustenance of a complete problem solution.

Putting the bounds together, we showed that XCS can *PAC-learn* a restricted class of *k-DNF* problems. However, the reader should keep in mind that XCS is an online generalizing, evolutionary-based RL system and is certainly not designed to learn *k-DNF* problems

particularly well. In fact, XCS can learn a much larger range of problems including DNF problems but also multistep RL problems as well as datamining problems as validated in subsequent chapters.

Before the validation, though, we need to investigate the last three points of our facetwise LCS theory approach for single-step (classification) problems. Essentially, it needs to be investigated if search via mutation and recombination is effective in XCS and how XCS distinguishes between local and global solution structure. The next chapter consequently considers problems which are hard for XCS's search mechanism since whole BB structures need to be processed to evolve a complete problem solution. We consequently improve XCS's crossover operator using statistical methods to detect and propagate dependency structures (BBs) effectively.

Chapter 6

Effective XCS Search: Building Block Processing

The facetwise approach to GA theory stresses effective mixing and decision making among BBs. The last chapter showed that in XCS, BBs are subsets of specified attributes that increase accuracy. The reproductive opportunity bound additionally ensures that BBs are able to grow in the population making time for the identification and reproduction of schema representatives. Until now, we assumed that mutation is sufficient to generate better classifiers as investigated in the time bound. However, the GA literature suggests that competent crossover operators are mandatory to solve boundedly difficulty optimization problems in which small, lower-level BBs may mislead the population to a local optimum.

Thus, this section investigates problems that pose a similar BB-challenge to the XCS system. We create hierarchical classification problems that demand the effective processing of lower level BB structures. In effect, we face the third part of our proposed facetwise problem decomposition for LCS systems, that is, the necessity to enable optimal solution search: (1) Search via mutation needs to be effective; (2) Search via recombination needs to be effective; (3) Local problem solution structure may be different from global structure and thus needs to be taken into account into the recombinatory search operators.

Search via mutation was investigated in several of the previous chapters. We showed its influence on specificity as expressed in the specificity equation (Equation 4.8) as well as its influence in generating schema representatives and finding an optimal problem solution (time extension of schema supply bound, time bound, Chapter 5).

This chapter focuses on recombination as well as differences between local and global problem structure. To investigate the effectiveness of recombination, we identify BB-hard problems in classification problems. XCS is not able to solve these problems due to disruption caused by crossover. Mutation alone may solve the problem but may take very long. To

solve the problems effectively, a competent crossover operator is necessary that recombines BBs without disrupting them. Experiments with an informed crossover operator confirm this hypothesis but are unsatisfactory since BB structures cannot be assumed to be know beforehand.

Thus, we integrate structure extraction mechanisms previously successfully applied in the GA literature. However, since XCS reproduces in action sets and thus in problem subspaces, the methods need to be modified for the XCS system—respecting the difference in local problem solution structure in comparison to global problem solution structure as suggested in the eighth point of our facetwise LCS theory approach.

In particular, we introduce the formation of *marginal product models* used in the *extended compact GA* (ECGA) (Harik, 1999). The technique is able to identify non-overlapping dependency structures in a problem. Since the marginal product model can only model non-overlapping BBs, we also utilize dependency structures in the form of Bayesian decision trees as used in the Bayesian optimization algorithm (BOA) (Pelikan, Goldberg, & Cantu-Paz, 1999). The Bayesian model can also detect overlapping dependency structures.

We integrate the methods in XCS by extracting a dependency structure from the *global* population and using the gained structural knowledge to generate *local* offspring. The resulting enhanced XCS system is able to solve the identified BB-hard problems.

The remainder of this chapter first derives BB-hard problems for classification. The evaluations show that only an informed crossover operator can solve the problems reliably. Next, we introduce competent crossover operators derived from mechanisms used in ECGA and BOA to solve the BB-challenge without any prior structural information. Summary and conclusions put the results into a broader LCS perspective.

6.1 Building Block Hard Problems

As we have seen in the previous chapter, XCS relies on the supply of minimal order schemata that increase classification accuracy—the BBs in XCS. In the previous chapter, we evaluated the schema bound and reproductive opportunity bound in a problem in which one minimal order schema had to be present. The solution was found once the block was detected and reproduced that specified all k_m attributes correctly.

The question is now, if XCS is able to identify and process several of those BBs, represented by schemata of order k_m, effectively. Thus, we first revisit fitness guidance to understand BB processing in XCS even better. Next, we create hierarchical classification problems which consist of several BB structures. In order to solve the problems efficiently,

it is necessary to identify, reproduce, *and* recombine the blocks appropriately. Thus, fitness guidance needs to be exploited to successfully grow blocks as well as effective recombination operators need to be available to successfully combine blocks.

This section first provides further exemplar problems and the consequent fitness guidance in the problem. Next, we introduce hierarchical classification problems showing that a proper BB propagation algorithm is mandatory to solve these types of problems effectively. Section 6.2 introduces explicit BB-identification and propagation mechanisms to XCS.

6.1.1 Fitness Guidance Exhibited

As noted before, the strength of the fitness pressure in XCS depends on the problem at hand. A typical easy problem for the XCS mechanism is the count ones problem (Butz, Goldberg, & Tharakunnel, 2003), in which the majority of ones (or zeros) in the relevant attributes decides the class. The accuracy structure in the count ones problem is very similar to the fitness structure of a one-max problem in the GA literature. Each relevant bit raises accuracy. Thus, each relevant bit is progressively more specialized in the condition parts of the classifiers in XCS.

Table 6.1 shows some exemplar classifier condition parts and the corresponding average reward prediction and reward prediction error estimates for classifiers with action part 1 in the count ones problem. It can be seen that the specialization of progressively more ones or more zeroes decreases error and consequently fitness. Thus, fitness progressively pushes towards the specification of more ones (zeros) in the problem. In Butz, Goldberg, and Tharakunnel (2003) it was shown that uniform crossover can assure and improve successful learning of the count ones problem with many additional irrelevant bits due to its effective uniform recombination.

In comparison with the count-ones problem, the hidden parity problem (Kovacs, 1999) is harder because the specialization of one attribute of the parity bits does not raise accuracy. Only once all relevant attributes are specialized, accuracy raises effectively directly solving the problem. Thus, supply of classifiers that specialize all parity bits is necessary, as also shown in the previous chapter. Table 6.1 shows the four hidden parity problem (the fifth bit is irrelevant). Error only drops to zero once all four attributes are correctly specified. In the next section we show that a hierarchical parity, multiplexer problem forces XCS to propagate the lower level parity blocks effectively.

Finally, we again show the widely studied multiplexer problem (Wilson, 1995; Wilson, 1998), in which accuracy somewhat guides towards the correct specializations. Initially,

Table 6.1: Expected reward prediction and reward prediction error estimates for exemplar condition parts in several typically-used Boolean function problems for classifiers with action part $A = 1$.

5-Count-Ones Problem			Hidden 4-Parity Problem			6-Multiplexer Problem		
C	R	ε	C	R	ε	C	R	ε
`#####`	500.0	500.0	`#####`	500.0	500.0	`######`	500.0	500.0
`1####`	687.5	429.7	`1####`	500.0	500.0	`1#####`	500.0	500.0
`##1##`	687.5	429.7	`0####`	500.0	500.0	`0#####`	500.0	500.0
`0####`	312.5	429.7	`11###`	500.0	500.0	`##1###`	625.0	468.8
`####0`	312.5	429.7	`1##1#`	500.0	500.0	`##0###`	375.0	468.8
`11###`	875.0	218.8	`00###`	500.0	500.0	`##11##`	750.0	375.0
`##1#1`	875.0	218.8	`111##`	500.0	500.0	`##00##`	250.0	375.0
`00###`	125.0	218.8	`000##`	500.0	500.0	`0#1###`	750.0	375.0
`#0#0#`	125.0	218.8	`101##`	500.0	500.0	`0#0###`	250.0	375.0
`110##`	750.0	375.0	`1110#`	1000.0	0.0	`0#11##`	1000.0	0.0
`111##`	1000.0	0.0	`0100#`	1000.0	0.0	`001###`	1000.0	0.0
`11##1`	1000.0	0.0	`0000#`	0.0	0.0	`10##1#`	1000.0	0.0
`000##`	0.0	0.0	`1010#`	0.0	0.0	`000###`	0.0	0.0
`0##00`	0.0	0.0	`1111#`	0.0	0.0	`01#0##`	0.0	0.0

though, only the specialization of the value bits raises accuracy. Only once some value bits are specified in a classifier condition, specialization of the address bits decreases accuracy further. Table 6.1 clarifies the property in the 6-multiplexer case. When starting with complete generality ($P_{\#} = 1.0$), relying on mutation for the first specializations, specificity initially raises more in the value attributes of the classifiers. Only later, specificity in the address attributes takes over.

6.1.2 Building Blocks in Classification Problems

The above problems consist of BB structure that either consist of only one BB, as in the hidden parity problem, or many single-attribute BBs as in the count ones problem. The multiplexer problem is somewhat a hybrid since initial fitness guidance leads to the less important value bits and only later, fitness guides towards the specialization of the address bits. Thus, BB processing is somewhat easy in the count ones problem in that only one specialization needs to be identified at a time. This can be accomplished by mutation. More challenging is the hidden parity problem in which classifiers that specialize all relevant bits need to be available from the beginning.

However, what happens if we combine multiple hidden-parity problems? If there is a

hierarchical dependency between the subproblems, first, the hidden parity blocks need to be identified, and next, they need to be recombined effectively. This describes a hierarchical problem structure that makes a problem hard for simple crossover operators.

We construct such a problem structure using a two-level hierarchy. On the lower level, small Boolean functions are evaluated which provide the input to the higher level. Thus, the function evaluation is pursued in two stages. The evaluation of the functions on the lower level serve as input to the higher level. For example, we mainly use a parity, multiplexer combination in which the small lower-level blocks are evaluated by the parity function. The results are then fed into the higher-level multiplexer function deriving the overall class of the problem instance. Further information and visualizations on the hierarchical problem class are provided in the Appendix C.

Note that we are not interested in creating a problem to force BB processing for its own sake. In fact, many indications in nature and engineering suggest that typical natural problems are structured in a hierarchical, decomposable structure (Simon, 1969; Goldberg, 2002). Thus, we believe that the introduced problem is an important problem to solve with a general machine learning system.

How can XCS solve this problem? Clearly, the lower level parity blocks need to be identified first to enable the discovery of the higher level function. Table 6.2 shows exemplar conditions with corresponding average reward predictions and prediction errors for the hierarchical 3-parity, 6-multiplexer problem. In contrast to the plain multiplexer problem or count ones problem, in these hierarchical problems, the lower-level BBs (for example, parity blocks) need to be identified and then processed effectively. The next section shows that only if the detected blocks are not disrupted, XCS is able to solve the problem. Additionally, only if the BBs are recombined effectively, XCS can solve the problem efficiently.

Note that we focus in the remainder on XCS's performance in the parity, multiplexer and parity, count-ones combination. Nonetheless, any other type of Boolean function combination in the proposed hierarchical manner is possible. Additionally, it is not necessary that all BBs on the lower level are evaluated by the same Boolean function nor do they need to be of equal length. Certainly, though, all these potential manipulations may lead to different challenges with respect to the facetwise theory for LCSs.

6.1.3 The Need for Effective BB Processing

We tested XCS on the proposed hierarchical problem combining parity and multiplexer problem as well as parity and count ones problem. Results confirm that the parity, multiplexer

Table 6.2: Expected reward prediction and reward prediction error measures for exemplar condition parts in the hierarchical 3-parity, 6-multiplexer problem for classifiers with action part $A = 1$. For readability reasons, the lower level 3-parities are tightly coded and separated by spaces.

C						R	ε
###	###	###	###	###	###	500.0	500.0
111	###	###	###	###	###	500.0	500.0
#1#	###	#11	#1#	#11	###	500.0	500.0
###	###	111	###	###	###	625.0	468.8
###	#1#	###	100	##1	###	625.0	468.8
###	0##	###	###	000	###	375.0	468.8
###	111	###	010	###	###	750.0	375.0
##1	111	##0	100	#0#	###	750.0	375.0
101	###	111	###	###	###	750.0	375.0
###	000	###	###	000	###	250.0	375.0
101	111	###	100	###	###	1000.0	0.0
101	000	111	###	###	###	1000.0	0.0

combination is particularly challenging.

Hierarchical three-parity, six-multiplexer problem

Performance of XCS in the hierarchical 3-parity, 6-multiplexer problem is shown in Figure 6.1.[1] It can be seen that XCS is not able to solve the problem if uniform crossover is applied. Due to the disruptive effects of uniform crossover—as already suggested in Holland's original schema theory (Holland, 1975)—XCS is not able to process the lower level building blocks but rather disrupts them.

In addition to the usual crossover operators of uniform, one-point, and two-point crossover, we applied an informed crossover operator to investigate the potential of more competent recombination operators. The informed crossover operator is informed about the BB structure in the problem applying a BB-wise uniform crossover operator. BB-wise uniform crossover exchanges only complete BBs uniformly randomly similar to uniform crossover, which exchanges attributes uniformly randomly.

XCS with BB-wise crossover solves the problem effectively and nearly independently of

[1]If not stated differently, all results in this chapter are averaged over ten experiments. Performance is assessed by test trials in which no learning takes place and the better classification is chosen. During learning, classifications are chosen at random. If not stated differently, parameters were set as follows: $N = 20000$, $\beta = 0.2$, $\alpha = 1$, $\varepsilon_0 = 10$, $\nu = 5$, $\theta_{GA} = 25$, $\chi = 1.0$, $\mu = 0.01$, $\gamma = 0.9$, $\theta_{del} = 20$, $\delta = 0.1$, $\theta_{sub} = 20$, and $P_{\#} = 0.6$.

the mutation type used. Thus, a mechanism in XCS is necessary that is able to identify the lower-level BB structure. Once identification is successful, effective BB processing and recombination can be applied. Uniform crossover strongly disrupts BB propagation preventing learning (Figure 6.1a,c). The continuously high population size indicates that uniform crossover causes a high diversity in the population. However, BB-disruption prevents the growth of higher-order BBs (Figure 6.1b,d). Mutation alone is able to solve the problem but learning takes about three times as long as in the case of BB-wise uniform crossover operator (Figure 6.1a,c). The population sizes indicate that diversity stays much lower resulting in a lower macro-population size (Figure 6.1a,c). If the BBs are tightly coded, one-point and two-point crossover are able to recombine BBs effectively (Figure 6.1a). However, if the attributes are randomly distributed, the potential recombinatory benefit of one-point or two-point crossover is overshadowed by their disruptive effect delaying learning and population convergence (Figure 6.1c,d). Thus, one-point and two-point crossover show beneficial effects in the case of tightly-coded building blocks but disruptive effects in the loosely-coded case. Since the dependency structures cannot be expected to be tightly coded in general, competent crossover operators are mandatory.

Although mutation can be tuned to solve the hierarchical 3-parity, 6-multiplexer problem nearly as good as the informed crossover operator does (Figure 6.2b), the behavior is unsatisfactory: larger problems or the same problem with additional irrelevant attributes would make it impossible to set the mutation rate high enough due to the reproductive opportunity bound introduced in the last chapter. However, a small mutation rate strongly delays learning if only mutation is applied. Figure 6.2c,d shows XCS's dependence on mutation rate in further detail. The BB-wise uniform crossover operator stays nearly independent from the mutation operator (Figure 6.2a,b). It only relies on the supply of lower level BBs, which is usually ensured by the initial sufficiently large specificity. Thus, a competent crossover operator that detects BBs on the fly is highly desirable.

Hierarchical two-parity, eleven-multiplexer problem

Figure 6.3 confirms similar results in the hierarchical 2-parity, 11-multiplexer problem. In the runs, population size is set to $N = 15,000$. The problem is slightly easier since the lower-level building blocks are only of order two and the size of the optimal set $|[O]|$ is 2^9 instead of 2^{10}. Thus, uniform crossover is less disruptive and the problem remains solvable even with uniform crossover. However, the beneficial effect of building block wise crossover remains tremendous. Again, it can be seen how one-point and two-point crossover become

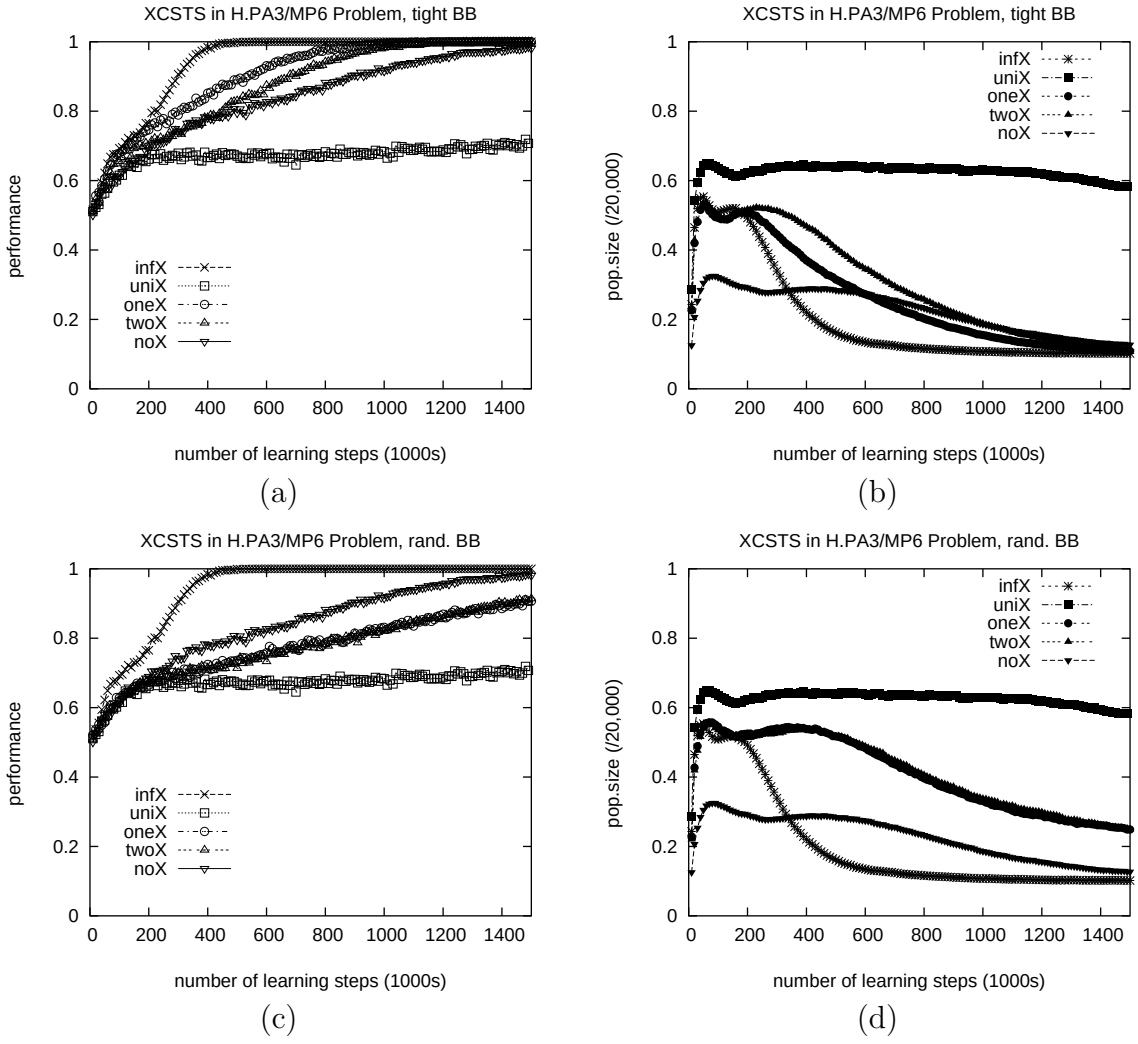


Figure 6.1: Performance (a,c) and population sizes (b,d) of XCS ($N = 20k$) in the hierarchical 3-parity, 6-multiplexer problem (infX = informed (i.e. BB-wise uniform) crossover, uniX = uniform crossover, oneX = one-point crossover, twoX = two-point crossover, noX = mutation only). Efficient BB recombination strongly improves XCS's performance. One-point and two-point crossover are only beneficial if the BBs are tightly coded. Although mutation alone is able to solve the problem, the time until the solution is found is much larger.

disruptive once the attributes are randomly distributed over the string.

Figure 6.4 exhibits the strong advantage of effective BB processing further. Performance with informed crossover stays nearly independent of mutation rate (Figure 6.4a,b). When mutation is set low and BB-wise uniform crossover is applied, the initial much stronger increase in population size indicates a much more effective exploitation of fitness pressure (Figure 6.4a). As in the hierarchical 3-parity, 6-multiplexer problem, without crossover

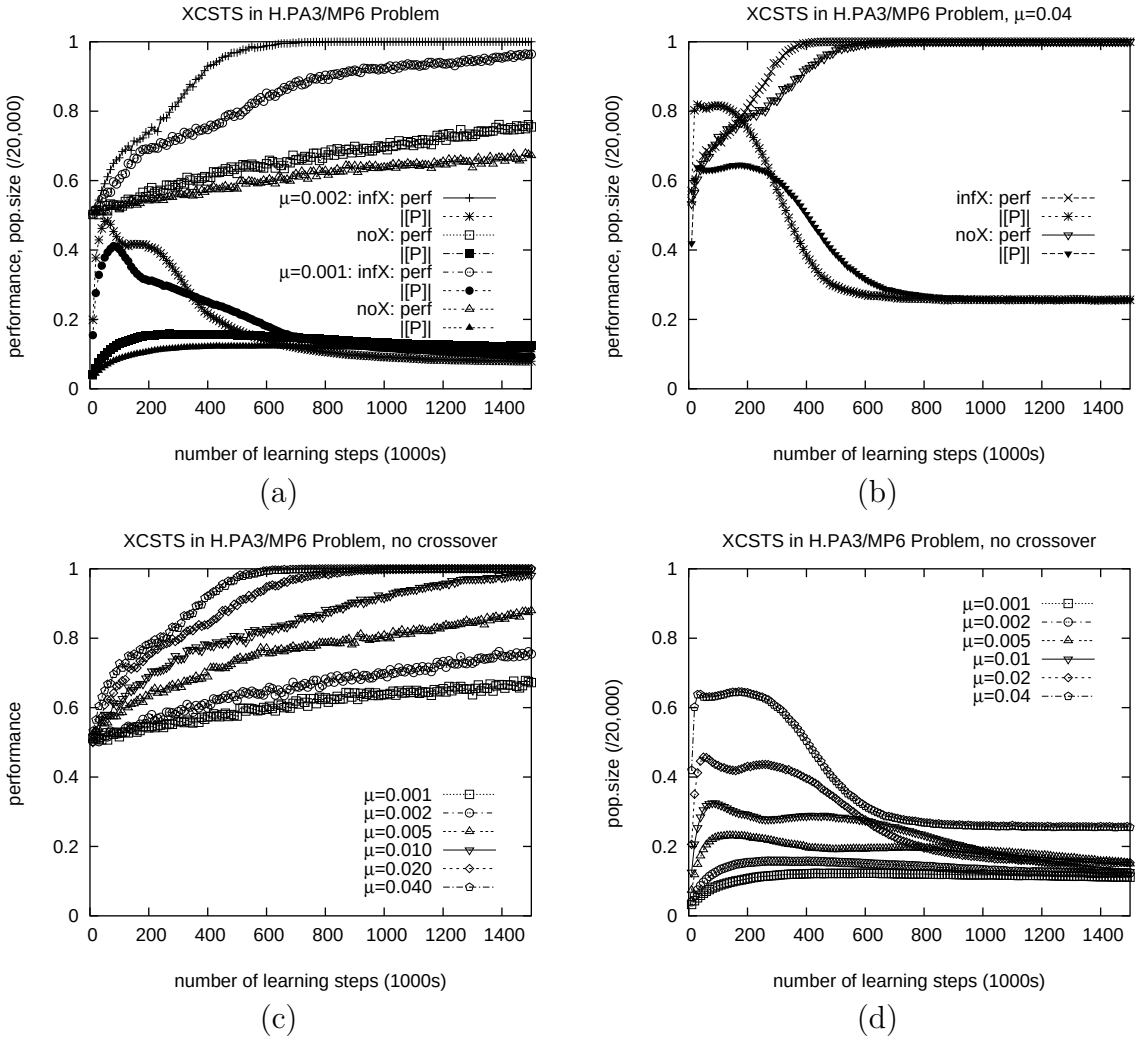


Figure 6.2: A low mutation rate strongly delays learning if effective recombination is not applied. With a mutation rate of $\mu = 0.001$, however, certain specialized attributes might get lost so that performance is delayed even with effective recombination (a). Higher mutation rates alleviate the problem (b) but may not be applicable in problems with more attributes. The comparison of different mutation rates exhibits XCS's dependence on a sufficiently high mutation rate for successful learning (c). Lower mutation rates cause lower diversity, lower specificity, and thus smaller population sizes (d).

performance strongly depends on an appropriate setting of mutation rate (Figure 6.4c,d).

Hierarchical parity, count-ones problem

Figure 6.5 confirms similar results in the hierarchical 3-parity, 5-count ones problem. In the runs, population size is set to $N = 15,000$. Note that the problem has as many niches as the

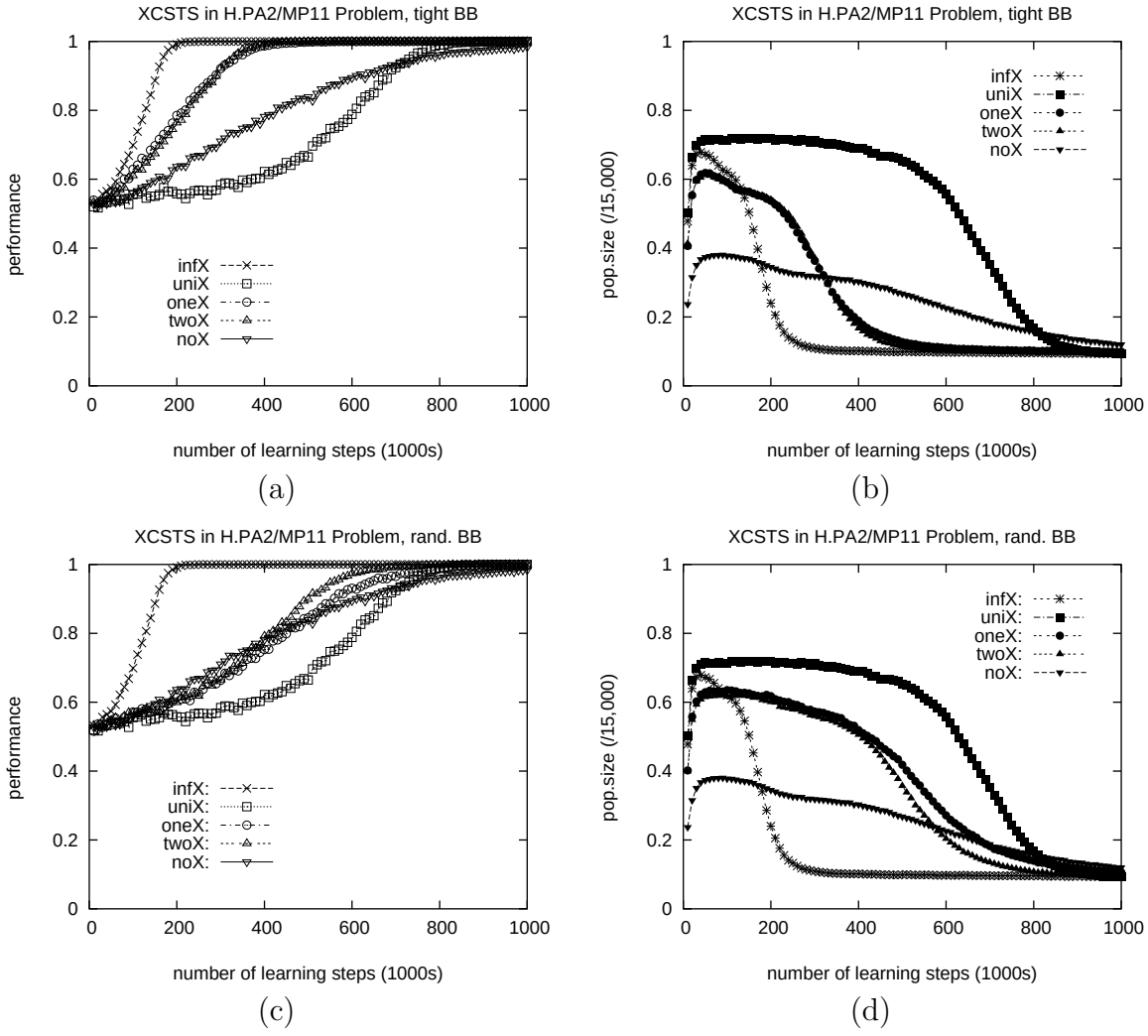


Figure 6.3: Performance (a,c) and population sizes (b,d) of XCS ($N = 15k$) in the hierarchical 2-parity, 11-multiplexer problem. Efficient BB recombination strongly improves XCS's performance. One-point and two-point crossover are only beneficial in the BBs are tightly coded. Although mutation alone is able to solve the problem, the time until the solution is found is much larger.

3-parity, 6-multiplexer problem. However, the smaller population size as well as the overlapping niches in the problem make it really hard for XCS to solve the problem completely optimally. Nonetheless, effective recombination significantly improves performance. As before, one-point and two-point crossover are only effective if the blocks are tightly coded. Otherwise, the operators are nearly as disruptive as uniform crossover. Mutation alone slowly improves performance but takes a very long time to evolve an accurate solution. The performance of BB-wise crossover is not reached by any of the other settings.

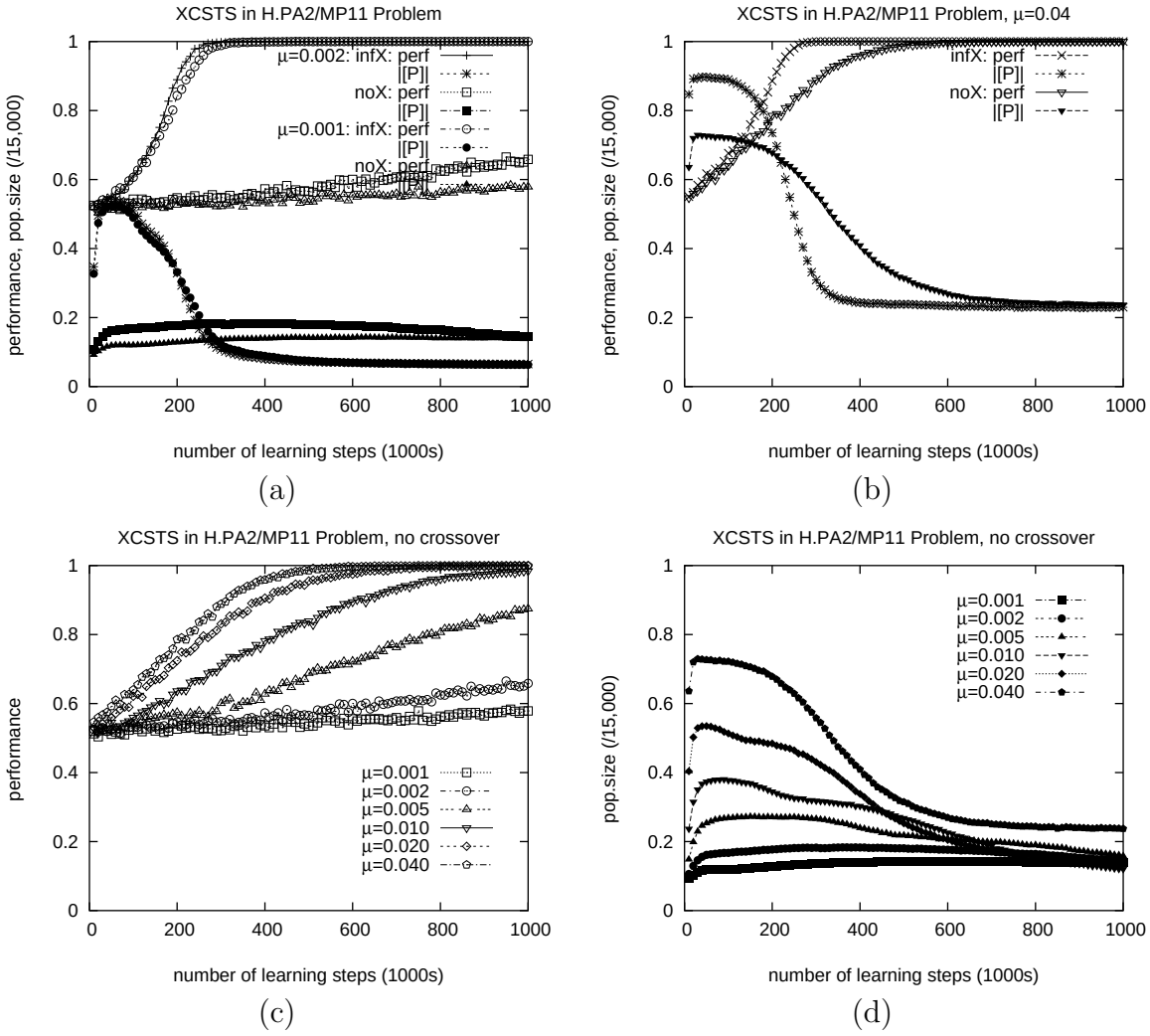


Figure 6.4: Similar to the hierarchal 3-parity, 6-multiplexer problem, low mutation rates strongly delay learning in the 2-parity, 11-multiplexer problem when only mutation is applied. The informed crossover operator, on the other hand, stays nearly independent of the mutation operator. Gradually increasing mutation rate show the direct influence of mutation on performance and population size if not crossover is applied (c,d).

6.2 Building Block Identification and Processing

Facing the BB-challenge within XCS it is necessary to develop a mechanism that learns effective recombination online. Most appropriate for this seems to be an estimation of distribution algorithm (EDA) approach modeling dependency structures and recombining them appropriately (Pelikan, Goldberg, & Lobo, 2002; Larrañaga, 2002).

The evolutionary component in XCS differs from the usual GA application in several

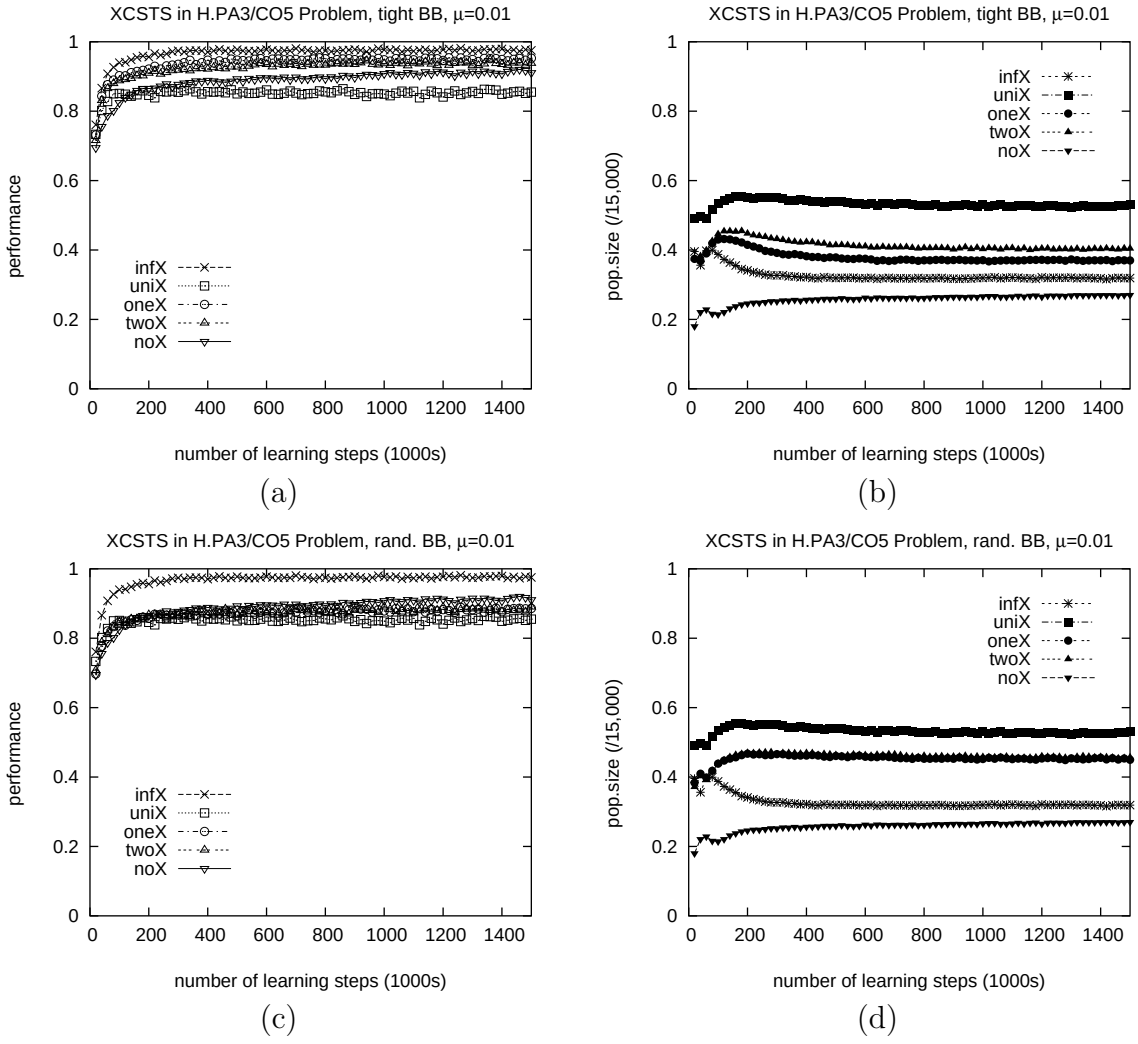


Figure 6.5: Performance (a,c) and population sizes (b,d) of XCS ($N = 15k$) in the hierarchical 3-parity, 5-count ones problem. Again, efficient BB recombination strongly improves XCS's performance. One-point and two-point crossover are only beneficial if the BBs are tightly coded. Mutation alone gradually improves performance but is much less effective that BB-wise crossover.

respects, though. Due to XCS's niched reproduction in action sets and since action sets are generally rather small compared to the whole population, structure extraction is hard to be applied successfully in an action set alone. On the other hand, extracting global structure results in global offspring generation, which may not reflect the local problem structure appropriately. Thus, the inclusion of an EDA mechanism in XCS is not straight-forward.

This section integrates the BB-identification mechanism in the ECGA (Harik, 1999) to identify and process lower-level dependency structures. Alternatively, we also show how to

integrate the more powerful Bayesian learning mechanism used in BOA (Pelikan, Goldberg, & Cantu-Paz, 1999). We show that both mechanisms are suitable to learn the *global* lower-level problem structure and can be used to generate or improve *local* classifier offspring. The generation and improvement of the local offspring depends on the current action set as the original XCS crossover operation does.

We first give an overview over the learning algorithms used in the ECGA as well as in BOA to learn the respective dependency structures. Next, we show how these mechanisms may be integrated into the XCS classifier system. We learn the dependency structures from the filtered and converted XCS population and then use the learned structures to sample and/or optimize offspring in local problem niches.

6.2.1 Structure Identification Mechanisms

Our investigations show that at least two structure identification mechanisms are suitable for competent BB processing in XCS: (1) *marginal product models* used for example in the ECGA mechanism (Harik, 1999), and (2) Bayesian decision tree structures used in BOA (Pelikan, Goldberg, & Cantu-Paz, 1999; Pelikan, 2002). The former is easier to understand and to apply but is limited to the identification of non-overlapping BBs only. The latter is more complicated but is able to model overlapping dependency structures as well.

Any structure extraction mechanism, however, faces the problem of accuracy vs. generality. That is, the generated model is intended to identify relevant dependencies but ignore spurious, irrelevant dependencies. Hereby, we rely on *Occam's razor* in that we want to find the model that codes the data structure most compactly. The usual approach to balance the two optimization factors is to apply the minimum description length principle (MDL) (Mitchell, 1997). Essentially, the MDL principle weighs accuracy with model complexity by combining the cost of describing the derived model with the resulting cost of encoding the modeled data using the model. Using information theoretic principles, the two influences can be appropriately balanced using entropy as the basic measure.

6.2.2 BB-identification in the ECGA

As mentioned above, the ECGA mechanism learns a non-overlapping BB-structure, termed a marginal product model. ECGA considers the best individuals in its current population (selected by any suitable selection mechanism, e.g. tournament selection) and builds the model from these individuals, that is, the data. For example, consider the simple population shown in Table 6.3. ECGA finds dependency structures in terms of feature subsets (that is,

Table 6.3: The illustrated example shows how the marginal product model learning mechanism detects structural properties in a population. While MC measures model structure complexity, CPC measures the compression possibly gained due to a more complex model.

problem instances	
11000	11111
11001	11110
11000	11110
00111	00001

marginal product model	MC	CPC	=
[1][2][3][4][5]	15	36.98	51.98
[1 2] [3] [4] [5]	18	30.49	48.49
[1 2] [3 4] [5]	21	22.49	43.49
[1 2 3 4] [5]	48	22.49	70.49
[1 2 3] [4 5]	30	30.49	60.49

the BBs). A block is essentially formed if the representation as a block, albeit more complex to express as a model, results in a sufficient reduction in the resulting data description complexity when using the model.

ECGA expresses these two complexity measures in terms of model complexity MC, which favors more compact models, and the resulting compressed population complexity CPC, which favors a more compact (accurate) population representation with respect to the used model. The MDL measure is simply the sum of MC and CPC. The two complexity measures are determined by

$$MC = \log N \sum_I 2^{S[I]} - 1, \tag{6.1}$$

$$CPC = N \sum_I E(M_I), \tag{6.2}$$

where N specifies the population size, I a dependency subset, $S[I]$ the number of attributes in subset I, M_I the probability distribution of all possible values in subset I, and $E(M_I)$ the entropy of a probability distribution. The measure MC exploits the fact that $\log N 2^{S[I]}$ bits are necessary to describe the probability distributions over each subset $S[I]$ ($2^{S[I]}$) probability entries. The measure CPC then determines the complexity of coding all N individuals with respect to the subsets, which is determined by the sum of the entropies over all subsets. Table 6.3 shows several potential model structures and the resulting MC and CPC measures. It can be seen how the MDL principle balances the model complexity with the resulting population description complexity.

The ECGA mechanism learns the marginal product model greedily minimizing the sum of MC and CPC. That is, if the scaled entropy decrease and thus the decrease of CPC due to a merge of two sets is larger than the consequent MC increase, the merge is performed.

Subsets are greedily merged until no more merge is able to decrease the MDL measure. In the ECGA, the model is learned every GA iteration. The offspring population is generated out of the derived dependency structure probabilistically sampling from the dependency structure. That is, each BB is considered independently when generating an offspring individual choosing the corresponding code probabilistically with respect to the determined probability distribution. The MDL mechanisms used to grow the dependency structure in XCS similar to the ECGA are taken from the available ECGA implementation (Lobo & Harik, 1999).

The ECGA mechanism has shown to be able to solve previously BB-hard problems, such as the typically used deceptive trap problems, effectively (Harik, 1999; Sastry & Goldberg, 2000). Due to its rather straight-forward approach and the various successful applications, it appears a valuable candidate for integration into XCS.

6.2.3 BB-Identification in BOA

BOA uses the more powerful representation of Bayesian networks in order to represent BB-structures. The overall learning mechanism is similar to the one applied in the ECGA, learning a Bayesian network from a selected subset of individuals and sampling from the Bayesian network. Due to the potentially much more complex Bayesian network structure, the generation and sampling mechanisms are not as straight-forward as in the ECGA.

Bayesian networks (BNs) (Howard & Matheson, 1981; Pearl, 1988; Buntine, 1991; Mitchell, 1997) combine statistics with graph theory generating a modular graphical model of the analyzed data. BNs can be used to estimate probability distributions as well as to do inference. A Bayesian network is defined by its structure and its (conditional) probabilities. The structure is usually encoded by a directed acyclic graph with the nodes corresponding to the features and the edges corresponding to conditional dependencies. The parameters are represented by a set of conditional probability tables (CPTs) specifying a conditional probability for each variable given any instance of the variables that the variable depends on.

The BN as a whole encodes a joint probability distribution given by

$$p(x) = \prod_{i=1}^{n} p(x_i|\Pi_i), \tag{6.3}$$

where $X = (X_0, \ldots, x_{n-1})$ is a vector of all the variables in the problem; Π_i is the set of parents of x_i (the set of nodes from which there exists an edge to x_i); and $p(x_i|\Pi_i)$ is the conditional probability of x_i given its parents Π_i. A CPT then codes the probability of

the values of x_i given the parental values. A directed edge relates the variables so that in the encoded distribution, a variable corresponding to a terminal node is conditioned on the parental variables. More incoming edges into a node result in a conditional probability of the variable with a condition containing all its parents.

As the ECGA structure assumes the independence of the blocks, also a Bayesian network encodes a set of (implicit) independence assumptions. Variables are assumed to be independent of each other given the values of the variables of all of their parents and none of their common descendants. The exact independence assumptions resulting from the BN structure can be found in the literature (Mitchell, 1997).

Conditional probability tables (CPTs) store conditional probabilities $p(x_i|\Pi_i)$ for each variable x_i. The number of conditional probabilities for a variable that is conditioned on k parents grows exponentially with k. For binary variables, for instance, the number of conditional probabilities is 2^k, because there are 2^k instances of k parents and it is sufficient to store the probability of the variable being 1 for each such instance. Figure 6.6 shows an example CPT for $p(x_1|x_2, x_3, x_4)$.

A greedy algorithm is usually used to learn a BN. The greedy algorithm starts with an empty BN. Each iteration, an edge is added to the network that improves quality of the network maximally. Network quality can be measured by any popular scoring metric for Bayesian networks, such as the Bayesian Dirichlet metric with likelihood equivalence (BDe) (Cooper & Herskovits, 1992; Heckerman, Geiger, & Chickering, 1994) or the Bayesian information criterion (BIC) (Schwarz, 1978). Learning terminates when no more improvement is possible.

The sampling of a Bayesian network can be done using probabilistic logic sampling (PLS) (Henrion, 1988). In PLS the variables are first ordered topologically so that every variable is preceded by its parents. The variables are then generated according to the topological ordering. As a result, once the value of a variable x_i is to be generated, its parents Π_i are assured to have been generated already. Thus, the probabilities of different values of x_i can be directly extracted from the CPT for x_i using the known values of Π_i.

Despite the encoded independence assumptions in a Bayesian network, identified dependencies may also contain regularities. Furthermore, the exponential growth of full CPTs with respect to the number of parents often obstructs the creation of models that are both accurate and efficient. Thus, often local structures are used in Bayesian networks to represent local conditional probabilities more efficiently than traditional full BNs (Chickering, Heckerman, & Meek, 1997; Friedman & Goldszmidt, 1999).

Pelikan (2002) uses decision trees to store the conditional probabilities of each variable

in a separate tree. Each internal (non-leaf) node in the decision tree for $p(x_i|\Pi_i)$ has a variable from Π_i associated with it and the edges connecting the node to its children stand for different values of the variable. For binary variables, there are two edges coming out of each internal node; one edge corresponds to 0 and the other edge corresponds to 1. For more than two values, either one edge can be used for each value, or the values may be classified into several categories and each category creates an edge.

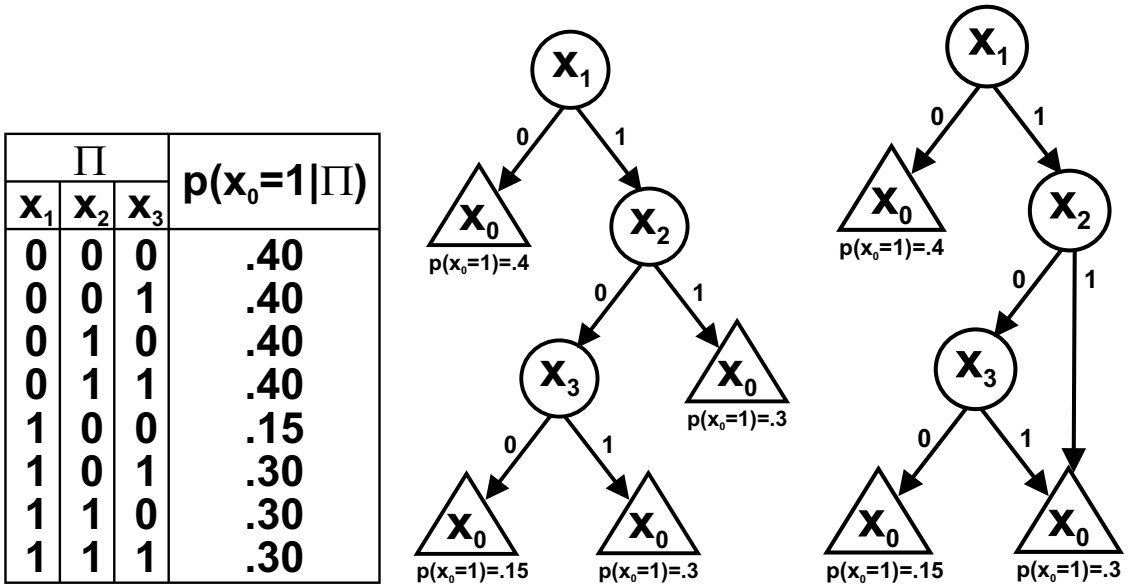
Each path in the decision tree for $p(x_i|\Pi_i)$ that starts in the root of the tree and ends in a leaf encodes a set of constraints on the values of variables in Π_i. Each leaf stores the value of a conditional probability of $x_i = 1$ given the condition specified by the path from the root of the tree to the leaf. A decision tree can encode the full conditional probability table for a variable with k parents if it splits to 2^k leaves, each corresponding to a unique condition. However, a decision tree enables the more efficient and flexible representation of local conditional distributions. See Figure 6.6b for an example decision tree modeling the conditional probability table presented earlier.

Pelikan (2002) uses also the (acyclic) decision graph feature allowing more edges to terminate in a single node enabling the sharing of children by several internal nodes. This makes the representation even more flexible and allows even more compact dependency structure representations. Figure 6.6c shows an exemplar decision graph.

To learn Bayesian networks with decision trees, a decision tree for each variable x_i is initialized to an empty tree with a univariate probability of $x_i = 1$. In each iteration, each leaf of each decision tree is split (as long as a topological ordering remains possible) determining the quality change of the current network measured by the applied metric. The best split is performed. Learning stops when no potential split is able to improve the current network.

To estimate model quality, a combination of the BDe (Cooper & Herskovits, 1992; Heckerman, Geiger, & Chickering, 1994) and BIC (Schwarz, 1978) metrics is used, where the BDe score is penalized with the number of bits required to encode parameters (Pelikan, 2002). For decision graphs, a merge operation is introduced to allow merging two leaves in any (single) decision graph. The Bayesian decision graph mechanism applied to XCS is taken from Pelikan's BOA implementation available on the net (Pelikan, 2001)

Similar to the ECGA approach, Bayesian networks model dependencies and independencies where a dependency is defined as a *non-linearity*. This non-linearity is identified by an entropy decrease as measured in the applied metric. Applying the introduced decision graph structure focuses the modeled dependencies in one decision graph on one feature modeling local, lower-level dependency structures, that is, the expected BBs in the problem.



(a) (b) (c)

Figure 6.6: A conditional probability distribution representation for $p(x_0|x_1, x_2, x_3)$ using a full-blown conditional probability table (a), as well as a decision tree (b) and a decision graph (c).

6.2.4 Learning Dependency Structures in XCS

Similar to the BB-identification mechanisms in ECGA and BOA, which search BBs in the current best subset of individuals, it is possible to learn dependency structures from the current population in XCS. However, two aspects need to be considered. (1) Selection from the global population is not straight-forward. (2) Classifiers need to be suitably transferred into binary.

As in ECGA and BOA, the dependency structure needs to be built from the current best individuals in the population of XCS. Despite the fitness sharing in XCS, relative accuracy may not be the appropriate measure since different problem niches may currently exhibit different learning stages so that the fitness may be misleading potentially favoring already converged subspaces. However, it is possible to require a certain classifier confidence for selection, similar to the thresholded application of subsumption. We use a filtering mechanism that extracts the most accurate classifiers out of the current population. The mechanism extracts those classifiers that have a minimum experience θ_{be}, a minimum numerosity θ_{bn}, and a minimum error $\theta_{b\varepsilon}$. The parameters were set to $\theta_{be} = 20$, $\theta_{bn} = 1$, $\theta_{b\varepsilon} = 400$ throughout the subsequent experiments filtering out the young and high-error classifiers. Since predictions below the average reward of 500 can be considered as predictions of the opposite class with

Table 6.4: Sample classifiers (from the multiplexer problem) and their corresponding binary encoding for the structure learning mechanism. Spaces are added for clarity. If an attribute is a don't care symbol, the second bit in the corresponding binary code is chosen randomly. The class bit is flipped, if the reward prediction is below 500.

C	A	R	ε	binary encoding
##11##	1	750.0	375.0	10 11 01 01 11 10 1
##00##	1	250.0	375.0	11 11 00 00 10 11 0
0#1###	0	250.0	375.0	00 01 11 11 11 10 1
0#0###	0	750.0	375.0	00 00 10 11 10 10 0
0#11##	1	1000.0	0.0	00 11 01 01 11 10 1
0#11##	0	1000.0	0.0	00 11 01 01 11 11 0
001###	1	1000.0	0.0	00 00 01 10 10 10 1
10##0#	0	1000.0	0.0	01 00 11 11 00 10 0
000###	1	0.0	0.0	00 00 00 11 10 10 0
01#1##	0	0.0	0.0	00 01 11 01 11 11 1

higher reward, we switch the class of those classifiers that predict a reward of less than 500. Note that this method can only be applied in classification problem in which only two types of reward (e.g. 1000/0) are possible.

Given a filtered population, it needs to be clarified how to translate the classifier population into a suitable representation to build the model. Don't care symbols may be simply coded by a third symbol in a ternary alphabet. However, don't care symbols do have a special meaning in that they match zero or one. Thus, to simplify model-building, we decided to code each condition attribute by two bits: The first bit encodes if the condition attribute is general (that is, don't care) or specific. The second bit encodes the value of the attribute. If the attribute is a don't care symbol, we choose zero or one uniformly randomly for the second bit. Finally, the classification part may yet play a special role and future work may build models for each classification separately. For now, we simply code the classification part as another bit. Table 6.4 shows a set of classifiers and the corresponding encoding that is used to learn the Bayesian network with decision graphs.

With a binary coded set of individuals at hand, we are able to learn the BB structure via the MDL-metric of the ECGA or the Bayesian decision graph structure via the Bayesian-network learning algorithm. Note that since XCS applies a steady-state niched GA, the dependency structure does not need to be re-built every time step. We re-build the network after a fixed number of time steps θ_{bs}, usually set to $10,000$ in our experiments. The threshold is only slightly problem dependent and does not appear to have a strong impact

on performance. In general, the lower the threshold, the more often the model is re-build potentially adjusting the model to newly detected dependencies faster but also potentially wasting computational resources for re-building the same dependency structure.

6.2.5 Sampling from the Learned Dependency Structures

As shown in Section 6.1, the recombination of the parents using common crossover operators may lead to disruptive effects potentially destroying important BB-structure. Once the dependency model is learned, XCS may use the model to recombine or directly generate offspring classifiers more effectively. As long as the learned model reflects important dependency structures, it can be expected that the resulting recombination is less disruptive and much more directed towards generating offspring that combines already successfully learned substructures effectively searching in the neighborhoods defined by the substructures.

As investigated in detail in the previous chapters, XCS generates offspring from parental classifiers selected from the current action set. This means that XCS reproduces classifiers that encode solutions with respect to the current problem instance. When using the globally learned dependency structures to generate offspring, we consequently need to adjust the structures to be able to generate local offspring. We investigate the following two options: (1) sampling classifiers using the model updated to the local probability distribution; (2) probabilistically improving the selected parental offspring classifier using the model with global or local probabilities.

Figure 6.7 shows the different potential methods for offspring generation by the means of a dependency model structure. Since XCS generates offspring in local niches, reproducing classifiers simply sampling from the global model is impossible since the classifier cannot be expected to reflect the solution structure in the current niche. Similarly, optimizing classifier structure by the means of the global model with global probabilities is expected to be disruptive as well since the optimization biased on the global probability structure again generates a classifier that reflects the global probability distribution. Even if the global model represents the dependencies in the population optimally, it may not be used to directly sample local offspring since it can only be expected to code lower-level BB information. Higher-level BB dependencies depend on the problem niche under investigation. Thus, it appears ineffective and very hard to grasp these higher-level dependencies in the global model.

Both offspring generation methods are introduced next. The latter is used only in conjunction with the learned Bayesian networks.

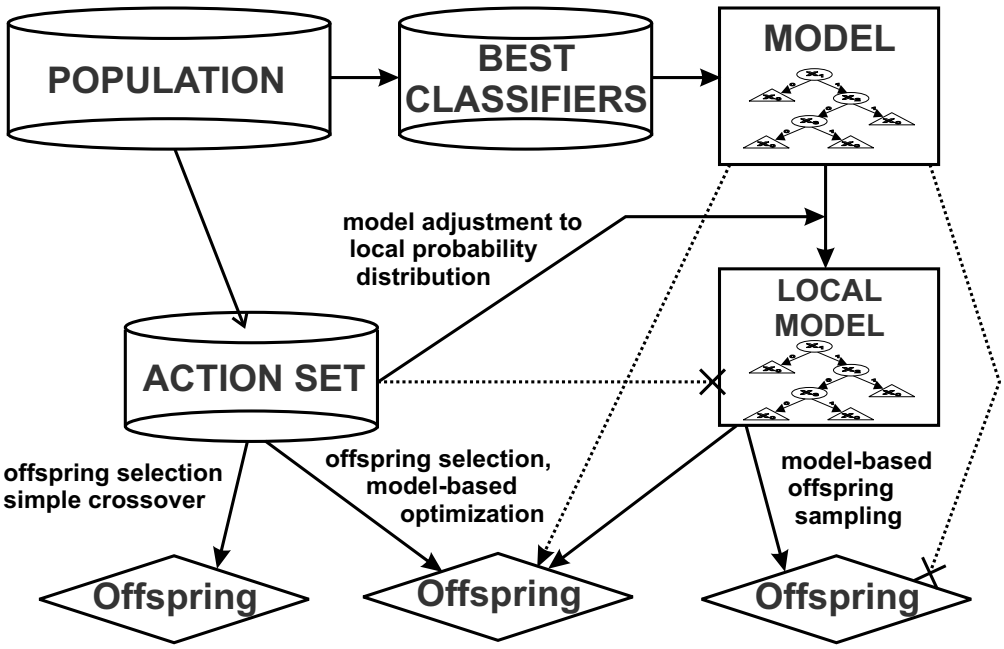


Figure 6.7: A probabilistic model of the problem structure can be build and used for offspring generation in many ways. Since action sets are usually too small to gather reliable statistics, a selected classifier subset from the global population should reflect problem structure most effectively. Once the model is formed, offspring may either be generated optimizing a parental classifier or sampling directly from the model. Sampling from the global model is inappropriate since the global distribution does not reflect the local solution structure. Setting the model distribution probabilities to the local probability distribution results in a model that encodes global dependency structures with respect to the local probability distribution. Thus, optimizing offspring by the means of the local model can be expected to be most cautious and most robust.

Sampling using local probabilities

Considering the effect on specificity of reproducing classifiers in an action set shows that offspring classifiers are expected to exhibit a specificity that corresponds to the average specificity of the action set. Fitness may increase this average specificity due its pressure towards higher accuracy, which often leads to an implicit specialization pressure as shown in Chapter 4.

Thus, to sample offspring using the learned dependency structure, the model probabilities need to reflect the local specificity distribution. Consequently, we update the probabilities in the applied model with respect to the best classifiers in the current action set. To achieve this, we select a subset of classifiers from the action set using the tournament selection mechanism. The selected subset (which may contain identical classifiers several times selecting with

replacement) is used to update the (conditional) probabilities. The updated dependency structure consequently reflects the detected global dependency structure but mimics the local probability distribution. The globally detected dependencies are thus combined with the local probabilities resulting in an offspring sampling mechanism that combines global with local problem knowledge.

The sampling is then achieved using the sampling mechanisms in the dependency structures explained above. Effectively, we use the globally detected dependency structures to sample local offspring. Hereby, the globally extracted structure biases the recombination. The current local probability distribution is used to sample offspring locally using the global structure. As a result, we recombine the locally important BBs effectively as long as the global dependency structure applies in the local problem niche.

Structure optimization

In the former case, we combined global model information about dependencies with the local probability distribution. Vice versa, it is also possible to use the global dependency structure and probability distribution to optimize the local probability structure. Hereby we have to be cautious to not overwrite the local information completely by the global information.

Essentially we apply a Markov Chain Monte Carlo (MCMC) approach (Neal, 1993) first introduced in the statistical physics literature in the 1950s in the so-called *Metropolis Algorithm*. An MCMC essentially iteratively and probabilistically changes a current probability distribution to an equilibrium distribution. MCMC iteratively evaluates potential changes of single attributes and decides probabilistically if the change should be made. Which probabilities are chosen to confirm an update is application dependent. Our method uses the likelihood of the change with respect to the global model.

Particularly, XCS chooses an offspring via tournament selection in the current action set. Instead of crossover, XCS then applies the MCMC mechanism to probabilistically optimize the classifier structure. Bits of the binary code of a classifier are chosen at random determining the likelihood of the structure before and after the change. To avoid zero likelihoods, all conditional probabilities are linearly normalized to values ranging from 0.05 to 0.95. Normalizing the likelihood before and after the change to one, the change then is committed with the probability of the normalized likelihood.

In effect, the MCMC pushes the selected local classifier towards the global probability distribution. The aim is to combine local *and* global structural information in the offspring classifier. Too many update iterations can be expected to be not useful since the resulting

classifier will reflect the global model structure. On the other hand, too few updates will have no effect at all. The subsequent experimental evaluations confirm these expectations.

To avoid using the global probability distribution, it is also possible to combine the two mechanisms above adjusting the dependency structure to reflect the local probability distribution using the consequent structure to probabilistically optimize a selected local offspring classifier. This has the advantage of avoiding the problem of over-optimization towards the global structure. Additionally, the freedom of sampling local offspring is further constrained since a parental offspring classifier is optimized constraining the search to an actual parental classifier. In effect, the combination might be the most cautious but also the most robust offspring optimization mechanism overall.

6.2.6 Experimental Evaluation

We evaluate XCS's performance on the above introduced hierarchical problems evaluating and comparing both offspring generation methods applying several different settings. XCS is set to learn a Bayesian network every 10,000 learning steps ($\theta_{bs} = 10,000$). The population XCS learns from is the filtered population as explained above. If the filtered population is empty, no model is learned. As long as no model is learned, XCS applies uniform crossover instead of the model-based crossover. Mutation is applied to the offspring classifiers generated by the model as before when simple crossover was applied. The results are averaged over 10 experiments. Other parameters are set as above except for the population size which is set to $N = 20,000$ in all runs as well as mutation which is set to $\mu = 0.01$ in the runs with normal crossover operators, to $\mu = 0.001$ in the XCS/BOA combination, and to either value as indicated in the subsequent figures in the XCS/ECGA combination. This population size seems large but the investigated problems are huge as well. For example, the hierarchical 3-parity, 6-multiplexer problem requires a final solution of 2^{10} classifiers so that the only twenty times larger population size appears reasonable.

Hierarchical parity, multiplexer problems

In Section 6.1 we saw that the evolution of a successful solution in the 3-parity 6-multiplexer problem strongly depends on the choice of mutation rate and crossover type. If a small mutation is chosen, effective BB recombination is mandatory and was achieved by an informed BB-wise crossover operator. If mutation is large, the problem was solvable but took longer than with the application of the informed crossover operator. However, we know that a large mutation value is not an option in larger problems in which a smaller mutation rate

is necessary to satisfy the covering challenge as well as the reproductive opportunity bound (Butz, Goldberg, & Tharakunnel, 2003). Thus, to solve the addressed problem, mutation needs to be set low and crossover needs to be effective.

Figure 6.8 shows XCS's performance in the hierarchical 3-parity, 6-multiplexer problem. In the ECGA comparison (Figure 6.8 a,b), we see that while BB-wise crossover learns the model slightly faster, ECGA reaches similar performance. The different settings refer to the number of selected classifiers used to adjust the model to the local probability distribution. Higher mutation rates are actually somewhat disruptive as also indicated by the resulting higher population sizes.

An additional specializing effect is observable that is partially a result of the binary recoding of the population for the model building and model-based offspring generation. Due to the random choice of the second bit of a don't care attribute, actual offspring may be generated that does not match the current action set. For example, consider the two simple classifiers `1#`→`1` `##`→`1`. The resulting binary codes may be `0111` and `1010`. Thus, dependent on the model dependencies, offspring may be generated with the code `0010`, which translates into `0#`→`1`. Although the average specificity is maintained, the offspring may not match the current action set consequently increasing diversity in the population. This additional diversity may be slightly disruptive as observed in the ECGA graphs. Note that we also ran experiments applying uniform crossover on the transferred binary code. The result was that the population was filled-up with apparently meaningless classifiers. No learning was observable in this case.

The application of the Bayesian model results in a similarly successful solution of the 3-parity, 6-multiplexer problem. The probabilistic optimization method even reaches higher performance slightly faster than the informed BB-wise crossover application (curve BOA: 0/18). However, if too many optimization steps are applied (0/90), the mechanism over-optimizes the offspring towards the global probability distribution and consequently over-specializes the population with respect to the current global model. This problem does not occur when offspring is sampled using the local probability distribution or if selected offspring is optimized using the local probability distribution. All settings exhibit similar performance nearly as good as the informed BB-wise crossover technique. In contrast to the ECGA combination, the BOA combination does not suffer from any over-specializations and all runs reach 100% performance reliably.

The hierarchical 2-parity, 11-multiplexer combination shows to be slightly easier to solve although the problem length is longer ($l = 22$ instead of $l = 18$ in the 3-parity, 6-multiplexer combination). XCS with the ECGA model builder solves the problem reliably in all settings.

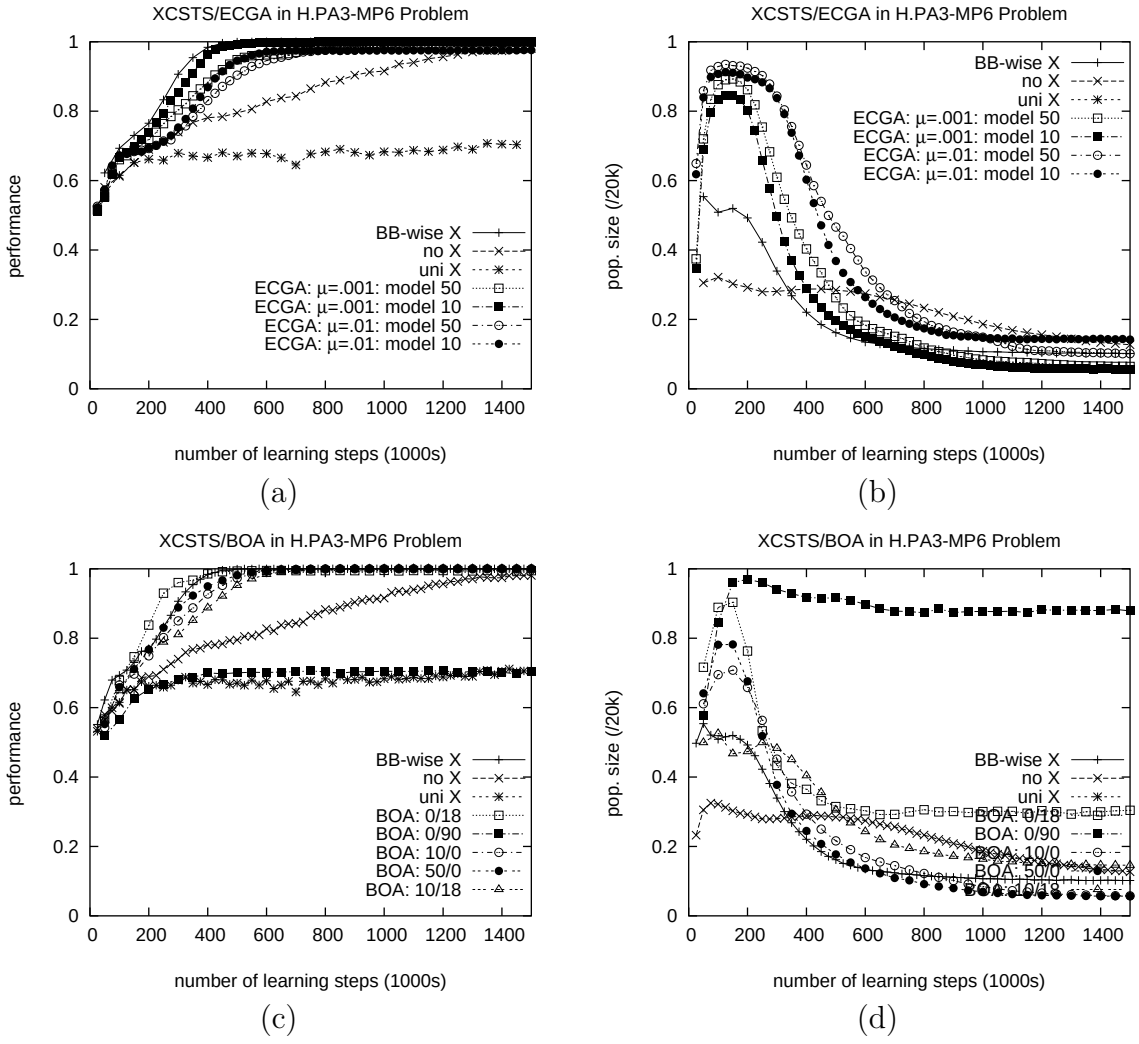


Figure 6.8: When applying ECGA or BOA to the hierarchical 3-parity, 6-multiplexer problem, recombination becomes effective and XCS is able to effectively learn a complete solution comparatively fast to the runs with BB-wise crossover. The application of the ECGA-based model learning mechanism (a,b) shows competent performance. The 50/10 variation refers to the number of selected classifiers used to set the probabilities to the local probability distribution. In the BOA-based model learning (c,d), the first number again refers to the number of selected classifiers used to set the probabilities to the local distribution (0 indicates that the global probability distribution is used). The second number refers to the probabilistic optimization steps applied to a selected parental classifier (0 indicates that offspring was sampled directly from the model).

If more classifiers are used to derive the current local probabilities, the consequent larger diversity appears to slightly delay convergence. In the XCS with Bayesian model runs, even the over-optimizing run with 90 probabilistic optimization steps towards the global model

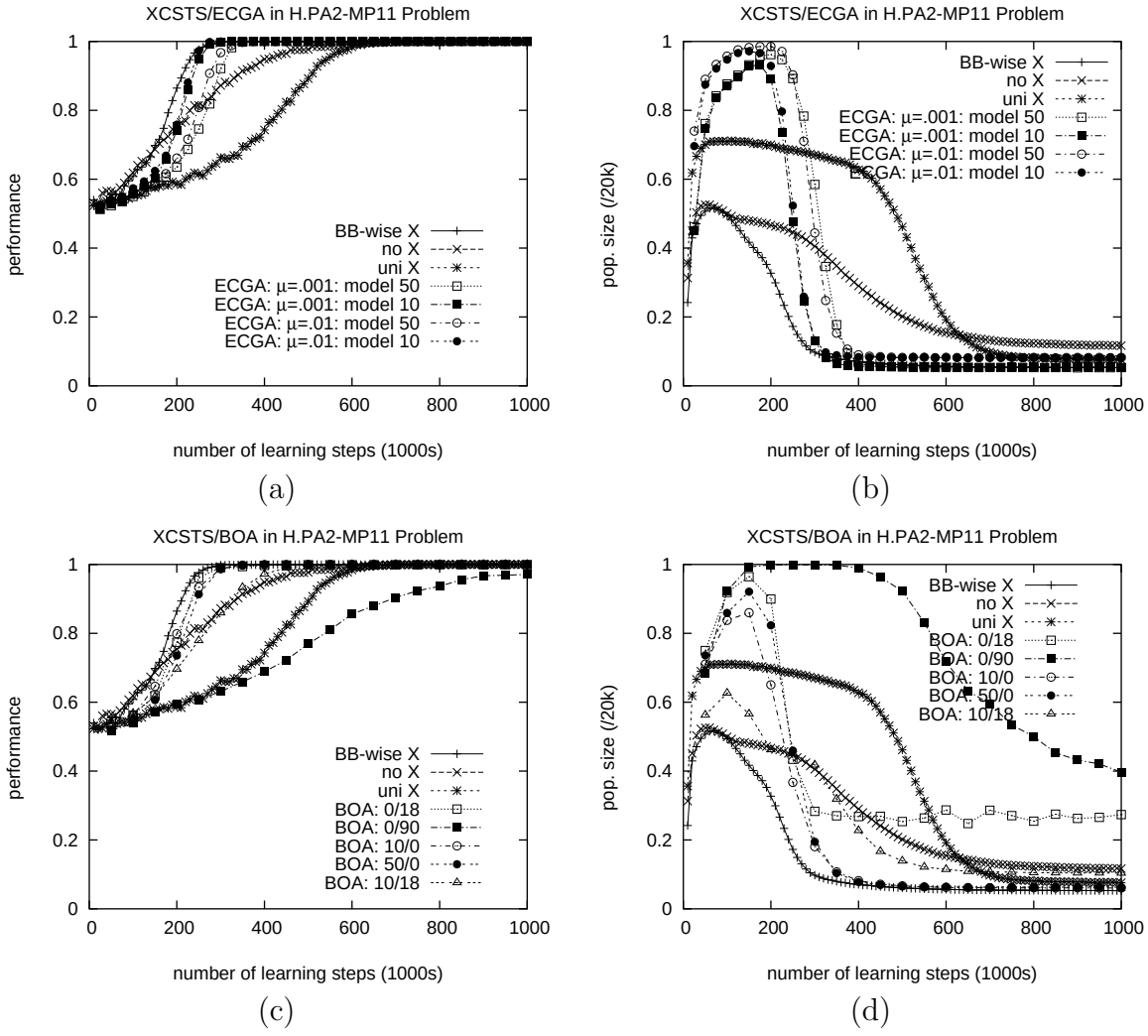


Figure 6.9: In the hierarchical 2-parity, 11-multiplexer problem, again the XCS combination with ECGA or BOA shows to solve the underlying problem nearly as good as when informed crossover is applied.

converges to a complete problem solution. However, note the huge population size that strongly decreases learning speed. When applying the optimization procedure with local probabilities, learning is slightly slower than when sampling or optimizing 18 steps using the global probabilities. This is the case due to the more cautious offspring generation causing less diversity in the population and thus a slightly slower but more cautious learning progress.

Hierarchical parity, count ones problem

Also the investigated hierarchical 3-parity, 5-count ones problem requires a final optimal solution size of 2^{10}. However, in this case the final population size is overlapping in that three out of five parity blocks need to be specified correctly to predict class zero or one accurately. XCS with ECGA model does not show any problems in solving the problem (Figure 6.10a,b). All runs converge to the near-optimal solution nearly as fast as the informed BB-wise crossover runs. Even with a lower mutation rate, performance is hardly influenced.

Similarly, the XCS runs with Bayesian network successfully learn the problem (Figure 6.10c,d). BOA is slightly slower than the ECGA combination in this case apparently because BOA detects spurious dependencies that may slow down the overall learning process. Since in this problem the propagation of all three BBs independently is nearly most effective, the Bayesian learning algorithm appears to over-model and thus delay the learning. Nonetheless, a near-optimal performance level is still reached very fast clearly outperforming the runs without crossover application as well as with uniform crossover application. As before, when probabilistically optimizing 90 steps using the global probabilities, performance is degraded indicating an overly strong bias towards the global probability distribution.

In sum, the results confirm that XCS can be successfully combined with a number of structural learners to improve offspring generation. The implemented XCS/ECGA and XCS/BOA combinations showed to be able to achieve performance similar to the performance with BB-wise uniform crossover, which relies on explicit problem knowledge. XCS/ECGA as well as XCS/BOA do not require any global problem knowledge and thus allow XCS to flexibly adjust its recombination operators dependent on the encountered problem. The next chapter provides further evidence for the generality of the model-building approach in XCS applying the techniques to several other typical Boolean function problems.

6.3 Summary and Conclusions

This chapter considered the last three aspects of the facetwise LCS theory approach in the XCS learning system for single-step problems. We showed that as in GAs, dependent on the problem, it might be necessary to process BBs in LCSs effectively to ensure reliable learning of a problem solution. Additionally, we highlighted the difference between LCSs and GAs in that problem structure may differ dependent on which problem subspace is currently under investigation. In essence, different attributes may be relevant in different problem structures so that different recombinatory mechanisms need to be applied dependent on the current

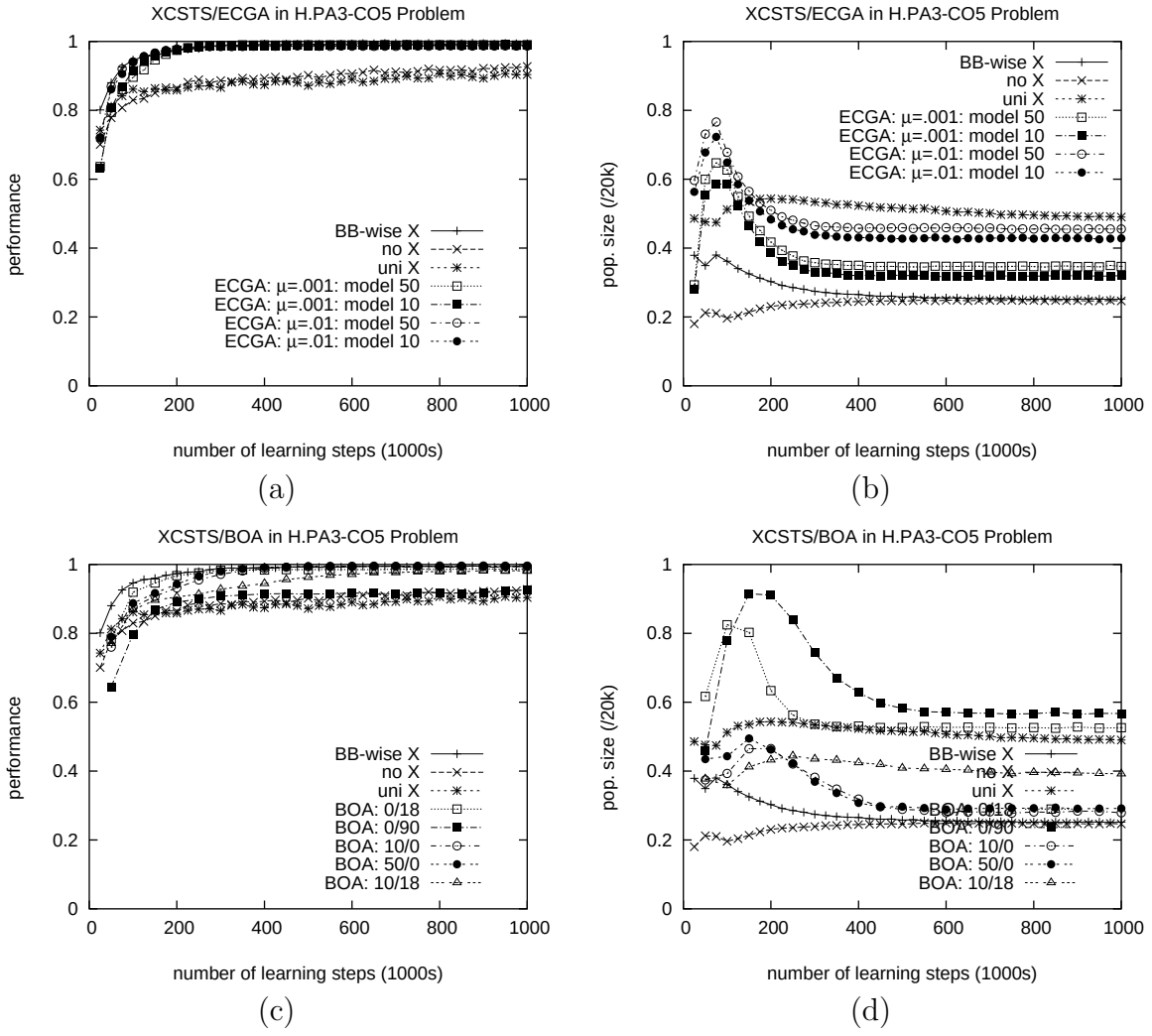


Figure 6.10: Also the hierarchical 3-parity, 5-count ones problem is effectively solvable with either model-based offspring generation method. The ECGA combination appears slightly more robust in this case indicating that the Bayesian-net might model unnecessary, spurious dependencies that delay convergence.

problem subspace.

To investigate the recombinatory capabilities of the XCS system, we introduced hierarchical binary classification problems combining parity blocks on the lower level with the multiplexer or count ones function on the higher level. XCS with simple crossover is not able to solve the resulting problems reliably. We observed the expected disruptive effects of simple recombination when applying uniform crossover as well as when applying one-point or two-point crossover with loosely coded blocks (randomly distributed over the population). Mutation alone is able to solve the parity, multiplexer problem combination—albeit severely

delayed in time—but it is not able to solve the parity, count ones problem combination satisfactorily in the available time.

The integration of the model-building and offspring generation mechanisms from the extended compact GA (ECGA) or the Bayesian model building algorithm (BOA) show that competent crossover operators can be integrated in the XCS learning structure. Since XCS applies a steady-state niche GA, the probabilistic model is not build every time step but it is build from the global model at predefined points in time. The model is then modified to reflect the local probability distribution in an action set at each time step to generate offspring respecting the local probability distribution but biasing recombination on the globally detected dependency structure. XCS with the integration of either model learner learned complete solutions to the hierarchical problems.

With respect to the final points of our facetwise theory approach for LCSs we showed that XCS respects the difference between global and local problem structure by reproducing in action sets. Recombination can be made more efficient by using global BB structure information. However, since the local problem structure needs to be respected but usually cannot be coded in the global dependency structure, offspring recombination needs to be adapted to the current local problem niche. Since knowledge about the current niche is represented in the local classifier population, that is, the current action set, model-based offspring generation needs to be biased on the classifier distribution in the current action set.

In conclusion, XCS with the integration of either model learner may be termed a *competent LCS*. That is, it is able to solve boundedly difficult problems—those with a minimal order complexity of k_m—effectively. The next chapter provides further evidence for the generality of the model-learning integration investigating XCS's performance in several Boolean function problems including noisy and very large problems.

Chapter 7

XCS in Binary Classification Problems

With all parts of the facetwise theory in place, this chapter applies XCS to several further binary classification problems experimentally confirming the usefulness of the introduced XCS enhancements as well as the theoretic learning behavior. In particular, we investigate further the effects of tournament selection, fitness pressure, the influence of noise, niching, model-based offspring generation and overlapping problems. [1]

7.1 Multiplexer Problem Analyses

The multiplexer problem is traditionally studied in the LCS literature due to its interesting function properties. As we saw in Chapter 4, the problem initially does not provide very strong fitness pressure. Even more severely, though, the fitness pressure initially suggests the specialization of the value bits instead of the address bits since only the value bits result in a decrease in average error (information gain). This phenomenon is illustrated in Figure 7.1 (left-hand side) confirming the initial faster rise in specificity in the value attributes before the specificity in the address attributes takes over.

This section investigates fitness guidance in the multiplexer problem as well as performance in large multiplexer instances including the 70-multiplexer and the layered 135-multiplexer. Next, we investigate very noisy multiplexer problem instances. Finally, we compare XCS's performance with the performance of XCS with the ECGA or the BOA model building enhancement.

[1] Throughout this chapter, if not stated differently, parameters are set as follows: $N = 2000$, $\beta = 0.2$, $\alpha = 1$, $\varepsilon_0 = 10$, $\nu = 5$, $\theta_{GA} = 25$, $\chi = 1.0$, $\mu = 0.01$, $\gamma = 0.9$, $\theta_{del} = 20$, $\delta = 0.1$, $\theta_{sub} = 20$, and $P_{\#} = 0.8$.

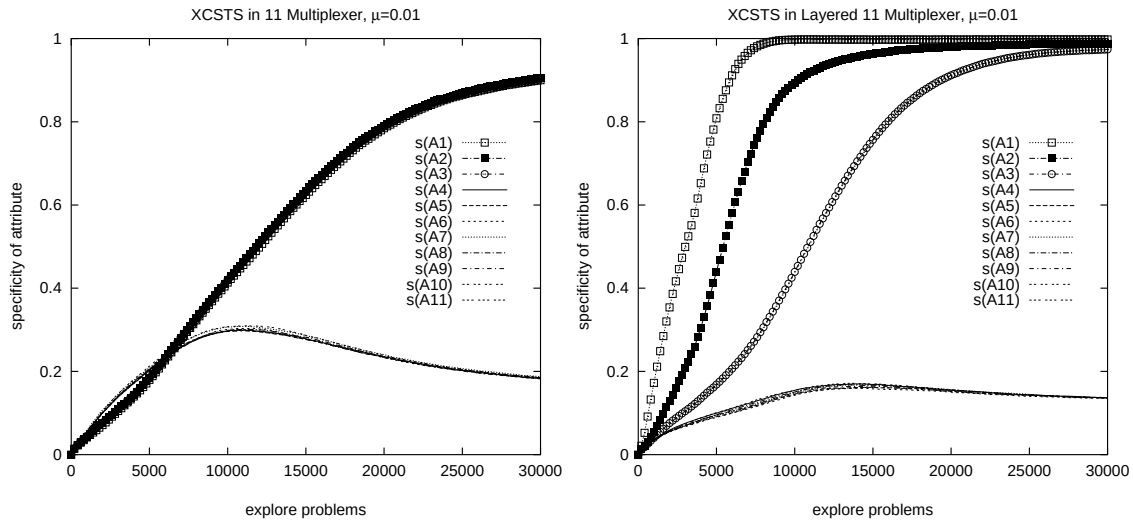


Figure 7.1: Plotting the specificity evolution in the 11-multiplexer, value bits initially gain more specificity indicating the initial misleading nature of the problem. In the layered 11-multiplexer on the other hand, additional fitness guidance biases the fitness pressure towards the specialization of the address bits from the beginning.

7.1.1 Large Multiplexer Problems

XCS was shown to be able to solve the 70-multiplexer even with proportionate selection (Butz, Kovacs, Lanzi, & Wilson, 2004). However, in Butz, Goldberg, and Tharakunnel (2003) it was shown that XCSTS solves the problem much faster and much more reliable. XCS reached 100% performance in approximately 4 000 000 learning iterations using a population size of $N = 50\,000$, a mutation rate $\mu = 0.04$ (niche mutation), and a sufficiently high initial specificity in the population ($P_{\#} = 0.75$).

Figure 7.2 shows the performance of XCSTS in the 70-multiplexer problem with a population size $N = 50\,000$ and a population size of $N = 20\,000$, a mutation rate of $\mu = 0.01$, and an initially completely general population ($P_{\#} = 1.0$). The curves are averaged over 25 experiments. The graphs show that XCSTS with a population size of 50 000 reliably solves the problem within 1 400 000 learning iterations. Decreasing the population size to 20 000 results in a much harder problem and all runs except one converged after 3 400 000 problems. The last run took more than 5 000 000 problems to find the optimal solution. Due to the small size of the population it is hard to give more accurate classifiers reproductive opportunities often loosing the detected higher accurate classifiers. No XCS run with roulette wheel selection was able to solve the problem within 6 000 000 learning steps in the applied parameter settings—the specializing fitness pressure apparently was never able to

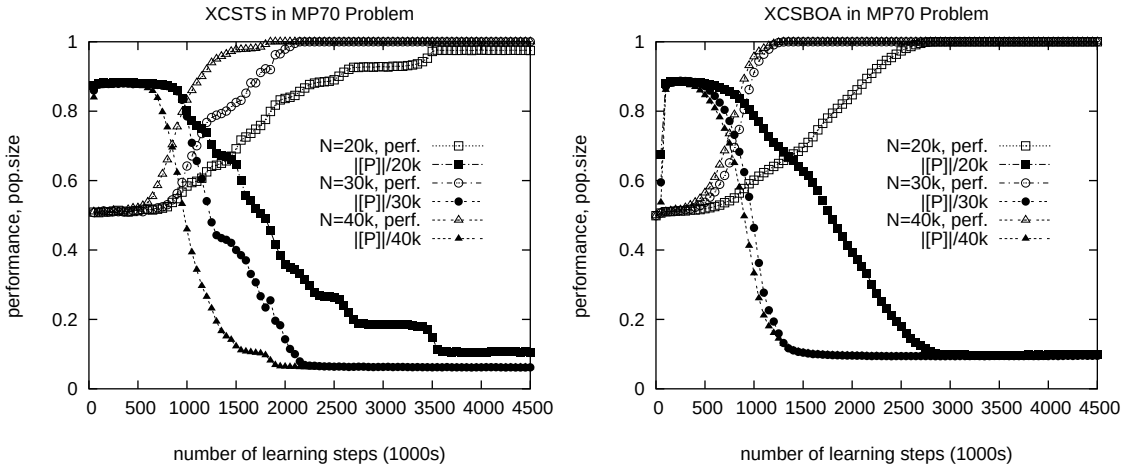


Figure 7.2: XCSTS reliably solves the very large 70-multiplexer problem. A smaller population size delays the learning progress. Substituting uniform crossover with the Bayesian network-based recombination approach results in more effective genetic search and thus faster and more reliable learning.

overcome the generalizing set pressure when starting with a completely general population and applying a smaller mutation rate than in the runs presented in Butz, Kovacs, Lanzi, and Wilson (2004).

In contrast to the multiplexer problem, the layered multiplexer problem provides strong fitness guidance leading to a domino-like convergence of specificities. In fact, each specialization of an address bit cuts the consequential reinforcement range in half strongly decreasing the expected average error. Thus, the layered multiplexer problem provides strong fitness guidance towards specializing the address bits. This observation is confirmed when plotting the specificity progress of each attribute in the 11-multiplexer problem. Figure 7.1 (right-hand side) shows how the specificities of each attribute increase progressively. The value bits are less misleading than in the normal multiplexer (left-hand side) and gain specificity slower than the more important address bits. This fitness guidance is certainly very helpful for the XCS learning mechanism.

Figure 7.3 shows the performance of XCSTS in the layered 70 and 135-multiplexer problems. Comparing the performance in the layered 70-multiplexer problem with that of the normal 70-multiplexer problem in Figure 7.2, we see that XCSTS is able to solve the 70-multiplexer problem much faster if reward is layered confirming the successful exploitation of the available fitness guidance. With this additional fitness guidance, XCSTS is also able to solve the layered 135-multiplexer problem successfully. Due to the additional fitness guidance, the evolutionary process is much more directed causing less additional specializations

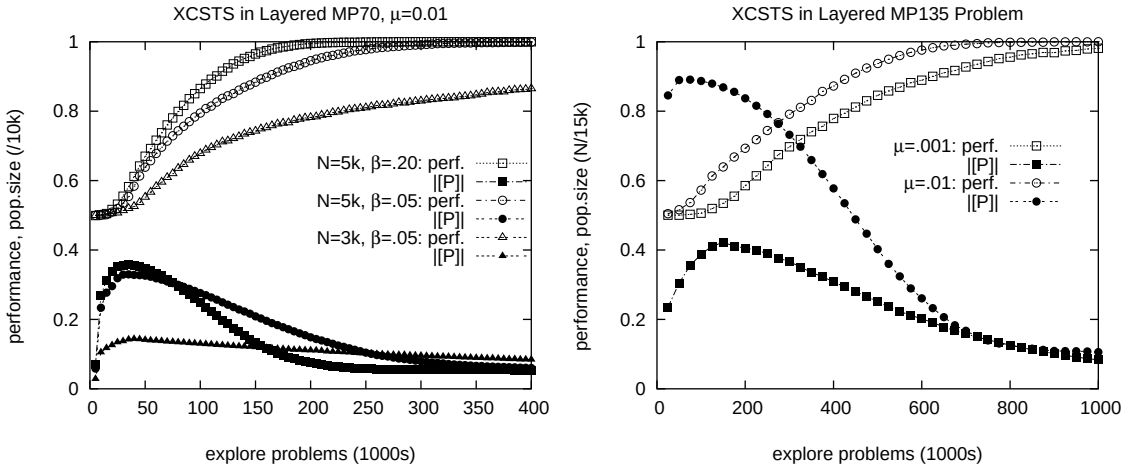


Figure 7.3: Due to the additional fitness guidance available in the layered multiplexer problem, XCS is able to solve the 70 (left-hand side) as well as the 135 (right-hand side) problem much faster than the corresponding normal multiplexer problem instances.

of unnecessary attributes effectively decreasing the specificity in the value attributes. The specificity decrease lowers the reproductive opportunity bound allowing the evolution of a solution with a population size as low as $N = 20k$.

7.1.2 Very Noisy Problems

When adding Gaussian noise with a large standard deviation σ, also XCSTS has a hard time to evolve an accurate population. Figure 7.4 shows that the speed of learning is strongly decreased when a noise of $\sigma = 500$ is added. Perfect performance can only be reached if the learning rate β is lowered, effectively decreasing the noise in the parameter estimations of the classifiers. Even higher noise values prevent XCSTS from reaching 100% knowledge reliably (Figure 7.5). When setting $\sigma = 700$, hardly any learning is observable.

It is possible to determine the actual expected standard deviation of classifiers in problems with additional Gaussian noise. Given two normal distributions $N_1(\mu_1, \sigma_1)$ and $N_2(\mu_2, \sigma_2)$ and noting that a classifier approximates the mean average deviation (MAD), a classifier that is only applicable in situation-classification combinations that yield payoff distributed by N_1 will have approximately a prediction error estimate of $0.8\sigma_1$ (since $MAD(N(\mu, \sigma)) = \sqrt{2/\pi}\sigma \approx 0.8\sigma$). The prediction error estimate of a classifier that encounters the first payoff distribution with probability p and the second payoff distribution with probability $(1 - p)$

can then be approximated as follows:

$$\begin{aligned}
\sigma^2 = E(X^2) - E(X)^2 = \\
= pE(X_1^2) + (1-p)E(X_2^2) - (p\mu_1 + (1-p)\mu_2)^2 \\
= pE(X_1^2) + (1-p)E(X_2^2) - (p\mu_1)^2 - ((1-p)\mu_2)^2 - 2p(1-p)\mu_1\mu_2 \\
= p(E(X_1^2) - \mu_1^2) + p\mu_1^2 - (p\mu_1)^2 + (1-p)(E(X_2^2) - \mu_2^2) + (1-p)\mu_2^2 - \\
((1-p)\mu_2)^2 - 2p(1-p)\mu_1\mu_2 \\
= p\sigma_1^2 + (1-p)\sigma_2^2 + p(1-p)\mu_1^2 + p(1-p)\mu_2^2 - 2p(1-p)\mu_1\mu_2 \\
= p\sigma_1^2 + (1-p)\sigma_2^2 + p(1-p)(\mu_1 - \mu_2)^2
\end{aligned} \tag{7.1}$$

This prediction error approximation enables us to estimate the difficulty of the problem and to predict if a problem is still solvable at all. In the case of $\sigma = 600$, the completely general classifier, for example, would experience a mean of 500 and a standard deviation of $(0.5 \cdot 600^2 + 0.5 \cdot 600^2 + 0.25 \cdot 1000^2)^{0.5} = 781$ so that the problem appears to be still solvable theoretically since the maximally accurate classifier experiences a standard deviation of $\sigma = 600$. It also shows how much more difficult the problem is than the problem without any noise. Without any noise, accurate classifiers experience a standard deviation of $\sigma = 0$ while the completely general classifiers experience a standard deviation of $\sigma = 500$. The smaller the differences in the experienced standard deviations, the more the evolutionary process will be misled since the deviations are only approximated by the means of temporal difference learning techniques. Even stronger approximation mistakes are expectable and even inevitable in "young" classifiers that experienced only few parameter updates so far.

7.1.3 Model Learning Mechanisms in the Multiplexer

Although we showed in the last chapter that XCS's performance can be improved using competent crossover operators that use a statistical analysis of the lower-level dependency structures in the investigated problem, it was not shown how robust the mechanisms are in other, non-hierarchical problems. Thus, we apply both mechanisms in the multiplexer problem comparing it with a normal, uniform-crossover application.

Figure 7.6 shows the performance and population size curves in the 20-multiplexer problem. Both model learning mechanisms slightly delay learning of a complete and accurate problem solution. Especially when using a larger classifier subset to set the model probabilities to the local solution, performance is delayed. This appears to be due to the diversification effect of the chosen binary encoding as discussed in the previous chapter. As observed before,

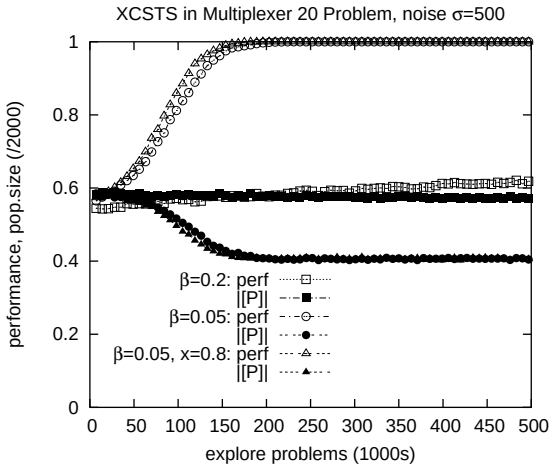


Figure 7.4: A lower learning rate β helps to derive accurate-enough fitness estimates in highly noisy problems. Crossover appears slightly disruptive.

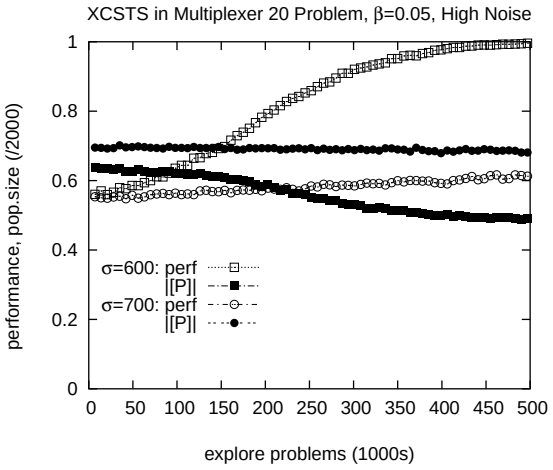


Figure 7.5: XCSTS is able to solve the Boolean multiplexer problem up to a noise level of $\sigma = 600$.

when applying too many optimization steps with respect to the global probability structure (setting 0/90), performance is severely delayed due to inappropriate specializations. In this still relatively small problem, the ECGA mechanism benefits from the additional specificity pressure due to a higher mutation rate.

In the 37-multiplexer problem, performance patterns strongly change. With the relatively small population size of $N = 5000$, learning is strongly delayed in the XCSECGA application due to apparent unnecessary over-specializations as indicated by the large early initial population sizes as well as the better performance when mutation rate is reduced. In effect, the reproductive opportunity bound appears to kick-in delaying the successful reproduction of better offspring significantly. On the other hand, the BOA mechanism is still able to generate offspring appropriately especially when probabilistically optimizing selected offspring regardless if from the local probability distribution or from the global probability distribution. A similar optimization mechanism may be combined with the non-overlapping model in the ECGA, which might yield similar performances. However, it can be expected that the additional capability to model overlapping dependency structures in the Bayesian-model may prove to be necessary to solve these larger multiplexer instances effectively.

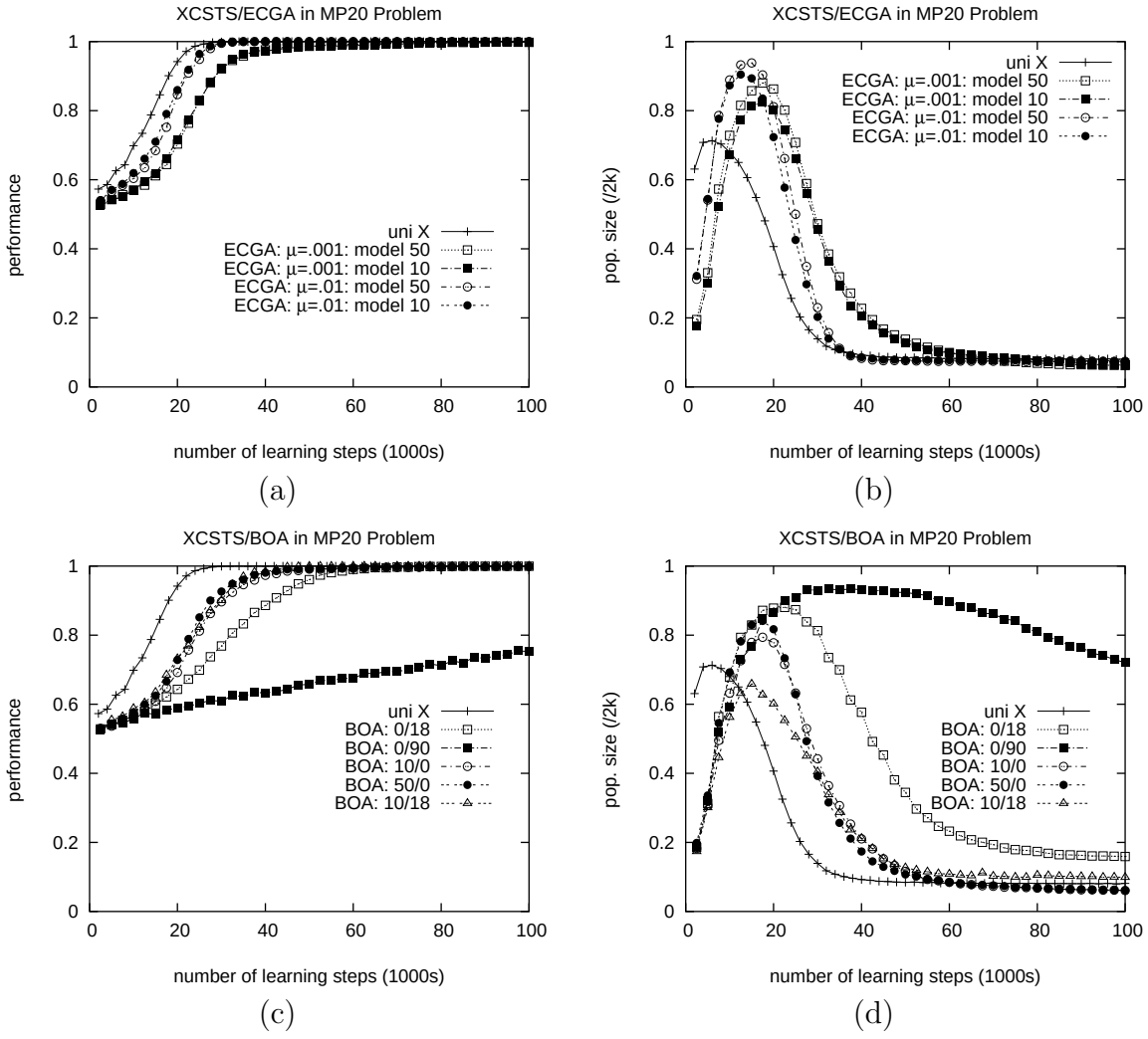


Figure 7.6: Both, the ECGA-based (a,b) as well as the BOA-based (c,d) model learning mechanisms, slightly affect XCS's performance delaying the successful learning of a complete model. However, neither mechanism prevents successful learning. The XCS-BOA combination was run with a mutation rate $\mu = 0.001$ and different model learning settings are applied as in the previous chapter.

7.2 The xy-Biased Multiplexer

The xy-biased multiplexer problem was introduced elsewhere (Butz, Kovacs, Lanzi, & Wilson, 2001; Butz, Kovacs, Lanzi, & Wilson, 2004) to investigate fitness guidance. The problem combines the difficulty of the multiplexer problem iteratively. A first multiplexer function with x address bits chooses the y multiplexer that decides on the current class of the problem. Again, the problem is somewhat hierarchical but not as clearcut as the hierarchical

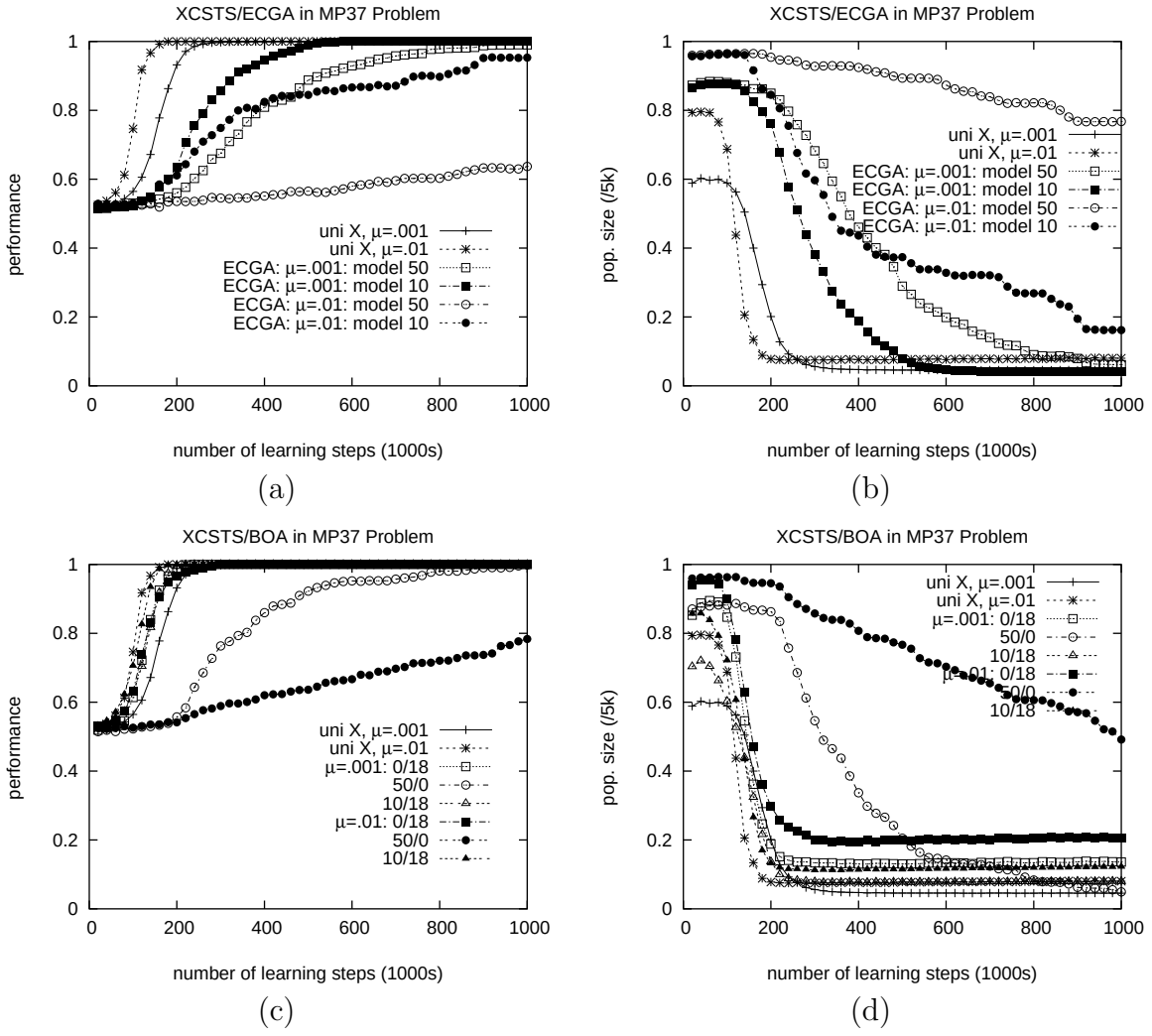


Figure 7.7: Particularly when sampling from a probability distribution reflecting the distribution of a larger classifier subset, performance is delayed in the 37 multiplexer problem. Classifier optimization instead of direct model sampling appears to be more appropriate in these larger problem instances.

problems we introduced earlier. In fact, the xy-biased multiplexer would be easy to solve by a typical hierarchical clustering approach (Duda, Hart, & Stork, 2001) in that the first x-bits partition the space focusing on the currently responsible classifier. However, learning the partition online is very hard since a specialization of one of the x bits does not necessarily yield any information gain (that is, increase in accuracy).

The xy-biased multiplexer is biased because the y multiplexer is slightly modified in that if all address bits are zero (or one) the result is a zero (one) regardless of the value bits, dependent if the biased multiplexer is zero (or one) biased, respectively. This biases the

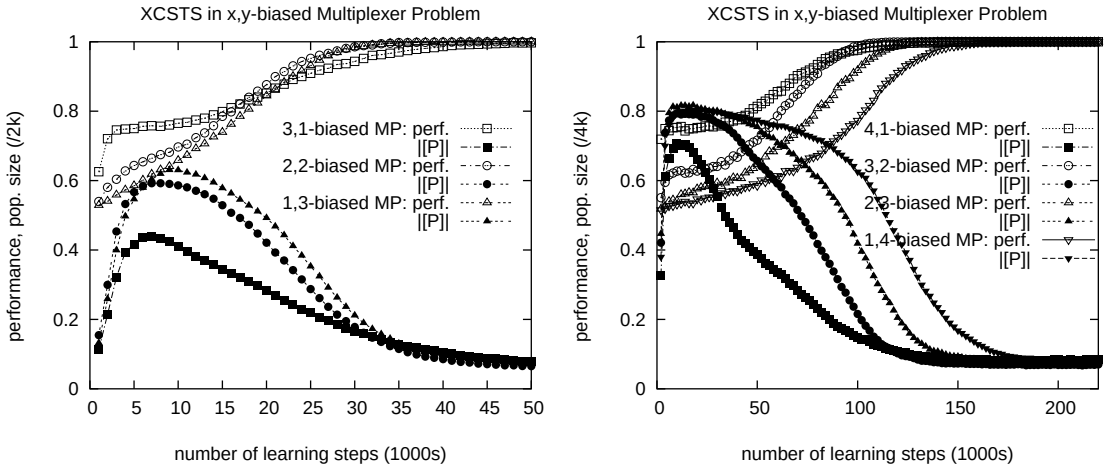


Figure 7.8: The x,y-biased multiplexer runs confirm the strong exploitation of fitness guidance in XCS. Population size was set to $N = 2000$ in the smaller runs on the left-hand side and to $N = 4000$ in the larger runs on the right hand side.

problem in that the specialization of an address bit can increase accuracy slightly. Further information on the problem can be found in Appendix C.

Figure 7.8 shows performance curves in various instances of the xy-biased multiplexer problem. Curves are average over 50 (left-hand side) and 20 runs (right-hand side). To have an idea of how complex the final solutions are we use the function $|[O]|$ introduced by Kovacs and Kerber (2001). This measure defines the complexity of a problem as the size of the minimal, accurate, non-overlapping population that covers all environmental niches accurately. We note that $|[O]|(3,1) = 48$, $|[O]|(2,2) = 56$, and $|[O]|(1,3) = 60$, suggesting that $(3,1)$ is the simplest problem, while $(1,3)$ is the most difficult one. The three plots on the left-hand side of Figure 7.8 however do not necessarily confirm a direct relation to the complexity $[O]$. In fact, the $(3,1)$ setting takes longest to converge but has the smallest $|[O]|$ value. Also the problem length l does not reflect problem complexity: while $(3,1)$ has the smallest input length $l = 19$, it only reaches higher performance initially but later hardly differs in performance gain from $(2,2)$ where $l = 22$ or from $(1,3)$ where $l = 21$. Additionally, the plateau in the $(3,1)$ curve is not explainable, either.

The explanation of the observed phenomena comes from the biased nature of the problem. In the $3,1$ problem, the specification of the first attribute allows an accurate prediction with 75% since the referred multiplexers are either all zero biased or all one biased. Since the biased multiplexer are of length 2, they are with 75% zero or one dependent on their bias. XCS detects this problem bias easily and very fast. However, the further specialization of the exception cases is much harder and delays the progress towards 100% performance. In

the $(2,2)$ and $(1,3)$ case, the bias is not as strong although $(2,2)$ still undergoes an initial performance boost.

Somewhat similar performance observations can be made in the larger runs shown on the right-hand side of Figure 7.8 in which population size was set to $N = 4000$. In the $(4,1)$ setting, the expected performance boost in the beginning is clearly observable. Also the $(3,2)$ exhibits the smaller initial performance boost due to the specialization of the first attribute. Setting $(1,4)$ has the smallest initial performance gain but later gains momentum and reaches complete performance only slightly delayed. The results confirm the exploitation of fitness guidance in XCS as well as its capability of processing various, distributed problem niches in parallel.

Similar performance influences caused by the problem structure can be seen in the much more difficult $(5,1), (4,2), (3,3), (2,4), (1,5)$ problem instances (Figure 7.9). Performance is averaged over 20 runs here and population size was set to $N = 15k$. As in the $(4,1)$ case, performance reaches the 75% level very fast in the $(5,1)$ problem. However, with simple uniform crossover, the problem is not solved even after $1.5M$ learning steps. The Bayesian search mechanism alleviates this problem and assures accurate learning. The tough problem in the $(5,1)$ problem is that the minimal order of difficulty is large ($k_m \approx 5$) since the first bit of the five address bits is easily detected but a specialization of any or even all of the other bits does not increase accuracy (given the first bit) as long as not at least one value bit is correctly set.

This difficulty (k_m) decreases so that in the simpler settings, XCS with the Bayesian search is outperformed by normal XCS recombination. Thus, the problem becomes simpler with respect to evolutionary search but becomes harder with respect to initial accuracy. The extreme initial gain in accuracy in the $(5,1)$ problem can cause disruption in the later learning progress. The Bayesian approach detects relevant dependencies much more effectively alleviating the disruptive effects of simple, uniform crossover.

7.3 Count Ones Problems

We introduced the count ones problem in Butz, Goldberg, and Tharakunnel (2003) in order to investigate the benefits of recombination in the XCS classifier system. In fact, the count ones and especially the layered count ones problem can be compared to the one-max problem in genetic algorithms in that each relevant attribute increases fitness, that is accuracy, independent of each other. Further details on the problem can be found in Appendix C.

Investigating the specificity behavior of XCS classifiers in the layered 10/5 count ones

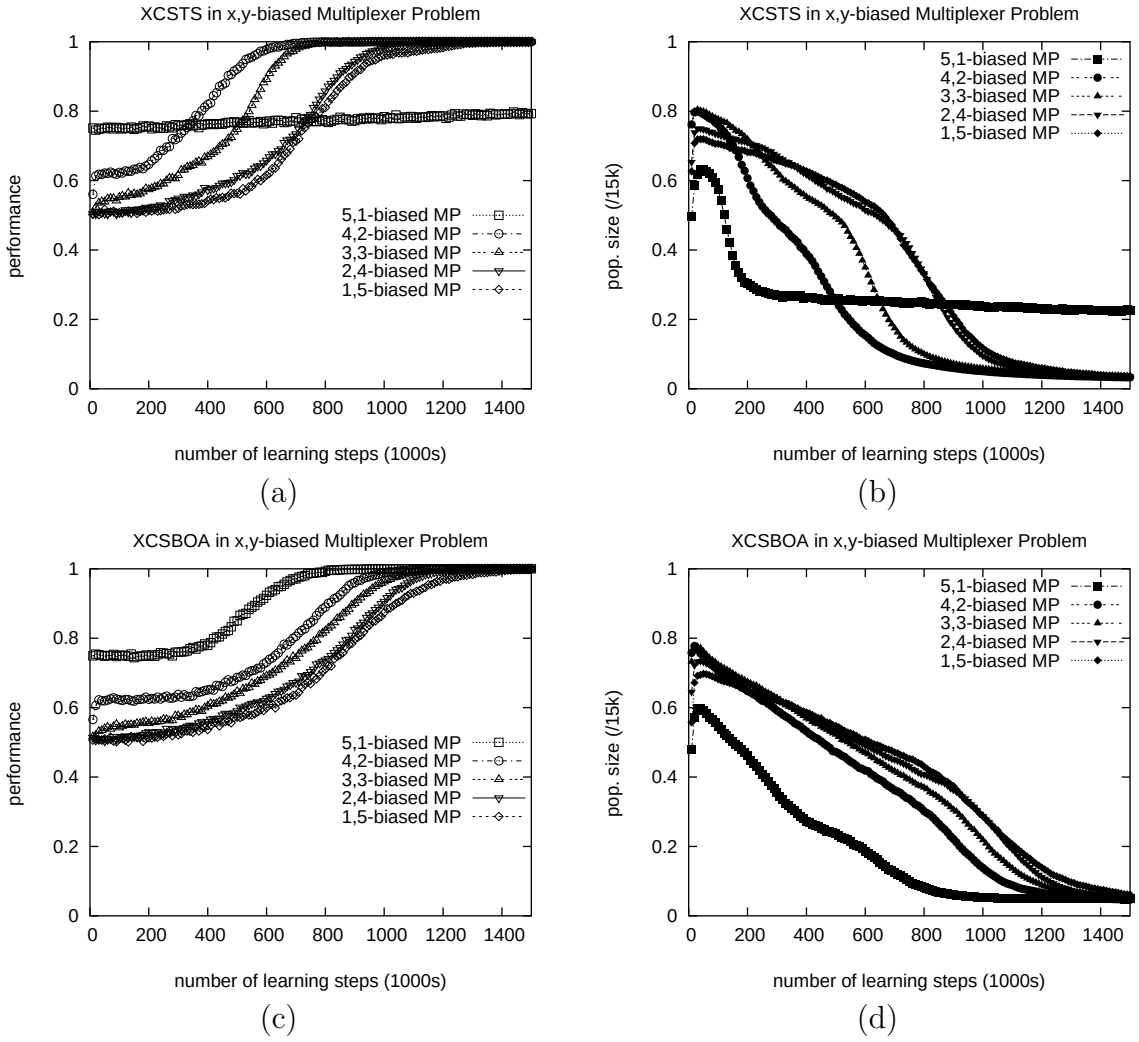


Figure 7.9: Larger biased multiplexer problem instances are particularly challenging since the minimal order of problem difficulty increases. The Bayesian search approach can detect and propagate lower-level dependency structures more effectively.

problem (referring to the problem length $l = 10$ and the number of relevant attributes $k = 5$ in the problem), Figure 7.10 shows that the specificity in all relevant attributes behaves similar while the specificity in all other attributes remains similarly low. In the count ones problem (left-hand graph), specificity reaches a level slightly above 3/5 since the optimal solution requires the specialization of three of the five attributes. The specificity stays slightly above the 3/5 because of the continuous specialization pressure caused by the application of the free mutation operator. In the layered count ones problem, 100% specificity is reached since all five attributes need to be specified to predict the resulting payoff level accurately. The performance line indicates that specificity converges shortly after complete performance

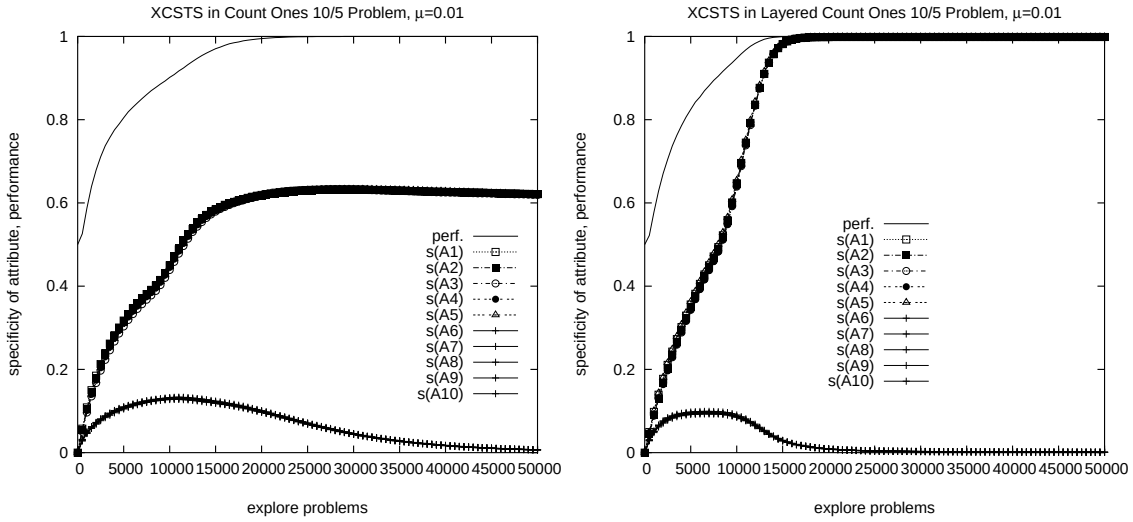


Figure 7.10: The specificity curves in the 10/5 count ones problem (left-hand side) and the 10/5 layered count ones problem (right-hand side) exhibit the strong fitness guidance towards specializing the relevant attributes.

is reached.

Comparing once more proportionate selection with tournament selection, Figure 7.11 shows that also in the count ones problems tournament selection is able to exploit fitness guidance much more effectively. Especially in the layered count ones problem, XCSTS solves the problem much more robustly. Additionally, crossover has a strong influence on performance since uniform crossover is very effective in the count ones problems.

Finally, we ran XCSTS in larger count ones problems with a problem length of $l = 100$ and seven relevant variables. Figure 7.12 shows that a lower mutation rate is mandatory to solve the problems effectively. Thus, XCS with roulette wheel selection fails to solve these larger instances since the necessary lower mutation rate is not sufficient to overcome the continuous generalization pressure. With tournament selection, though, the fitness pressure alone is strong enough and a complete problem solution is evolved. However, evolution is only successful if crossover is applied. Mutation alone may still solve the problem but performance is strongly delayed. Due to successful recombinatory events combining detected substructures, the large chunks of seven bits are detected more efficiently.

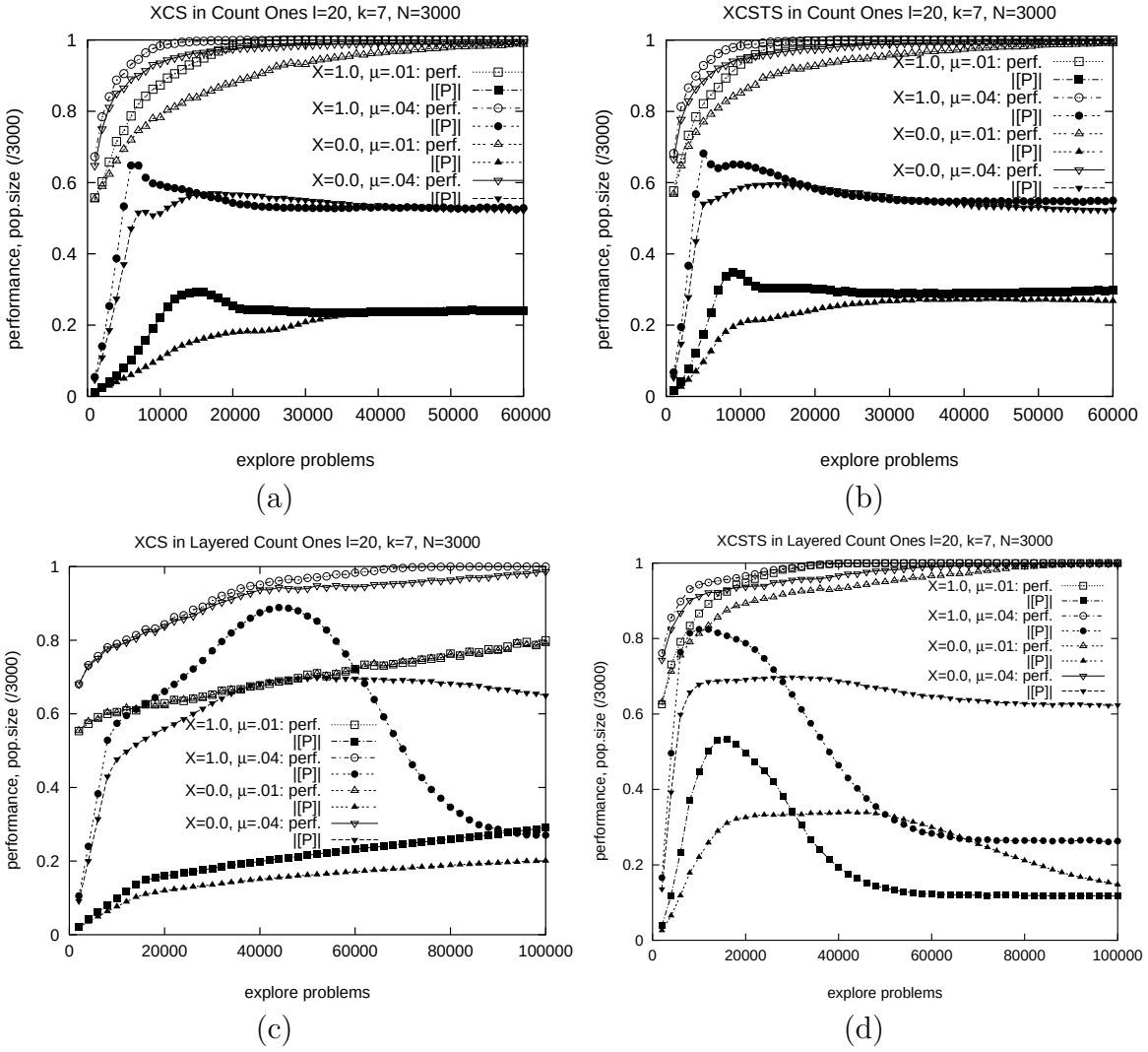


Figure 7.11: Performance results in the 20/7 count ones (a,b) and layered count ones (c,d) of XCS with proportionate selection (a,c) and XCSTS (b,d) once again confirm the superiority of tournament selection. Additionally, recombination is very beneficial in these problems.

7.4 Carry Problem

As a final Boolean classification problem we ran further runs in the Carry problem. The interesting part of the problem is that a simple solution is found easily in any instance of the carry. However, the hard part is the detection of the carry bit the further back in the string it is pushed. That is, while the easiest case of a carry is simply specified by requiring that the two left-most bits are set to one, the hardest instances are those in which the length of one number plus one bit need to be specified in order to assure that a carry occurs. Thus, performance is expected to reach high values in any instance of the carry problem. However,

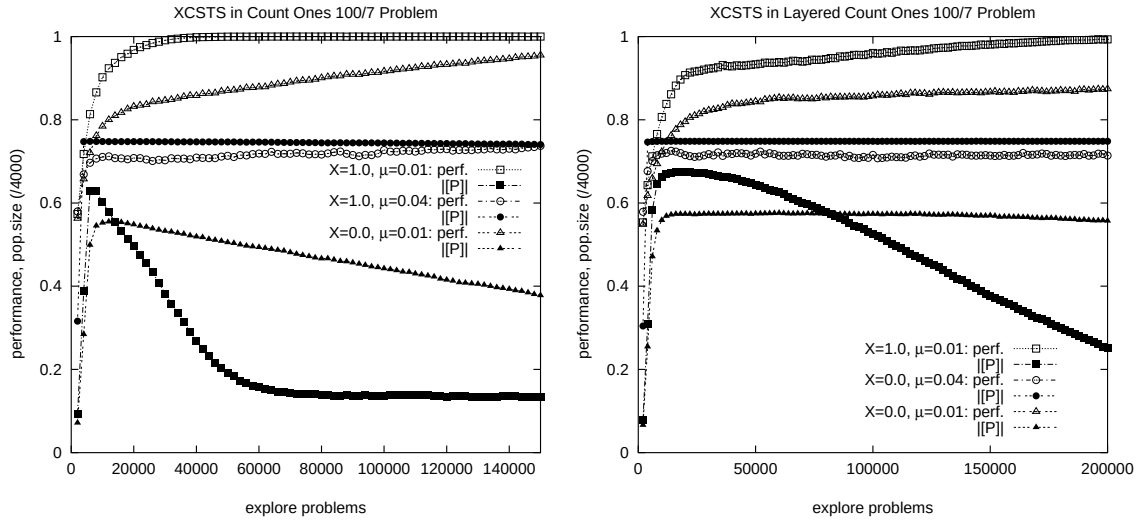


Figure 7.12: In the larger 100/7 count ones (left-hand side) and layered count ones problems (right-hand side), a low mutation rate is necessary to enable reproductive opportunities. Population size is set to $N = 3000$.

to reach values close to 100% becomes progressively more difficult. Additionally, due to the overlapping representation of the final problem solution, niching may be more important as investigated in the previous chapter. Further information on the carry problem can be found in Appendix C.

The observations are confirmed in runs conducted in the carry 4,5,6, and 7 problems with population sizes of $N = 2k$, $N = 4k$, $N = 8k$, and $N = 16k$, respectively, shown in Figure 7.13. XCS solves the problems fast, as expected, to a high performance level. However, 100% performance is reached only in the carry 4 problem. All other runs reach more than 99% performance at the end of the shown runs. Clearly, the highly improbably exceptional cases are the ones hard to detect. This is also expected from the derived theory since the occurrence frequency of the exceptional cases decreases exponentially in the size of the carry, time until those exceptional cases are learned as well increases exponentially in the size of the carry as predicted by the time bound in Chapter 5. Setting the GA application threshold to a higher value was expected to assure a more reliable maintenance of the infrequently occurring problem niches. However, the time bound seems to have an even stronger influence on performance in the shown settings so that a larger GA threshold only delays performance increase but does not yield any more accurate performance in the undertaken runs.

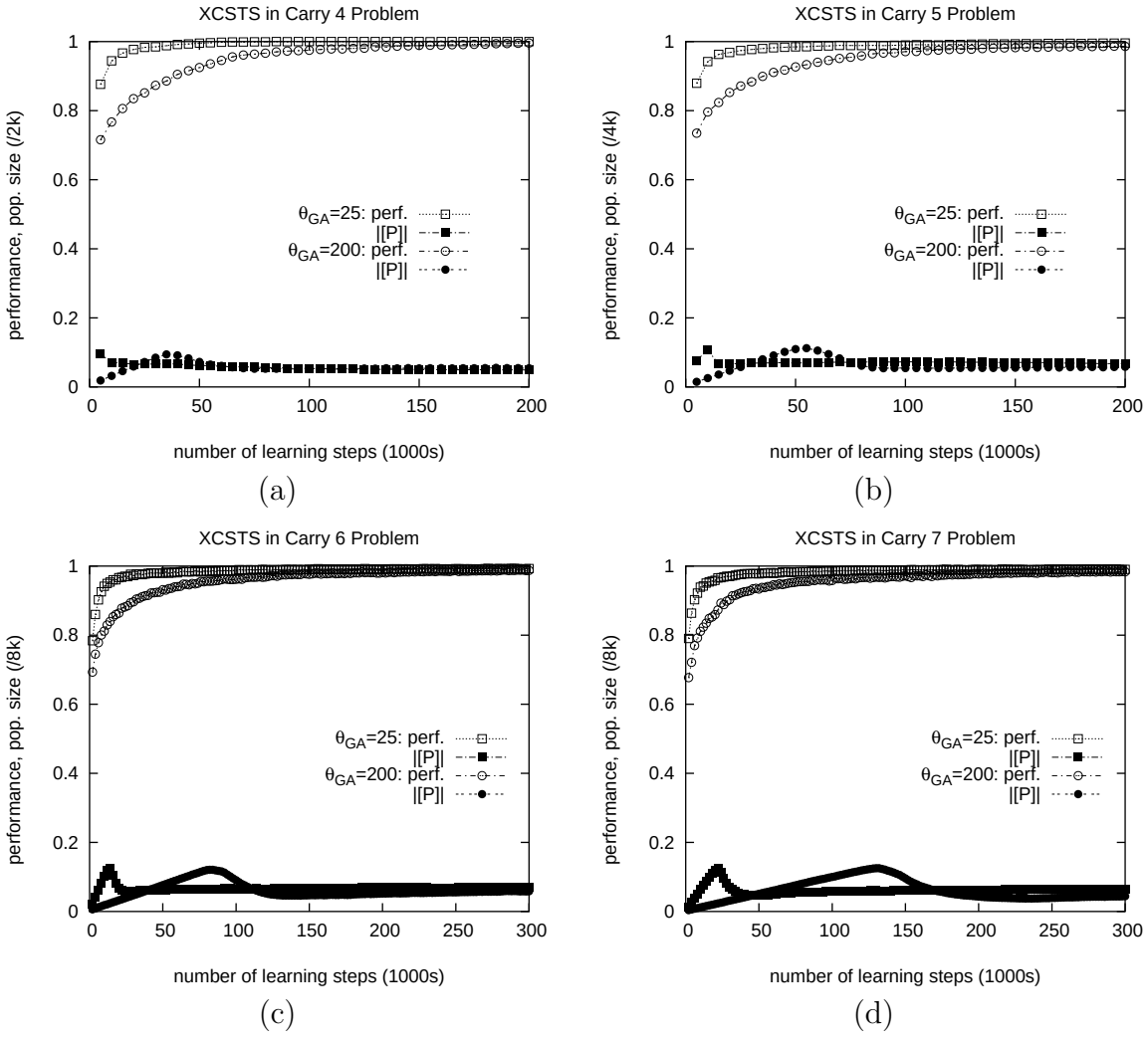


Figure 7.13: The carry problem poses the challenge of overlapping niches as well as very infrequently occurring subproblems.

7.5 Summary and Conclusions

This chapter has shown that XCS is applicable in a wide variety of classification problems. The problems range from non-overlapping to overlapping solution representations as well as from equally sized solution subspaces to highly unequally sized ones. Additionally, the problems were shown to possibly only provide noisy reinforcement feedback. XCS was able to solve all these problems with an appropriate setting of population size and mutation rate. All other parameters did not need to be adjusted as long as starting out with a sufficiently general population controlled by the $P_{\#}$ parameter.

This confirms that despite the large number of parameters in the XCS system, most

parameters have only a slight influence on performance and can be set to their standard values. Only population size and mutation rate may need to be set appropriately dependent on the complexity of the problem. Thus, a future research goal is to adapt population size as well as mutation rate on the fly during a run when performance stalls due to over-specialized classifiers and a too small population size.

In conclusion, this section confirmed that XCS is capable of solving a large variety of Boolean function problems as proposed by the derived facetwise theory in the last chapters. The reader needs to be reminded that XCS, however, is neither restricted to solve such Boolean function problems nor particularly designed to solve these Boolean function problems. In fact, in all the undertaken experiments feedback is solely provided in terms of reinforcement instead of direct, supervised feedback. Additionally, XCS learns online from one problem instance at a time. Thus, XCS is very much suitable to learn in more cognitive scenarios in which feedback is available only in terms of reinforcement and learning is required to be undertaken online. These conclusions are confirmed also in the two subsequent chapters in which XCS is applied to several (multistep) MDP problems in which reward backpropagation is necessary as well as in a variety of real-world data problems in which the handling of real valued and nominal attributes is required. Additionally, in both scenarios problem instance sampling is not uniform over the problem space.

Chapter 8

XCS in Multi-Valued Datamining Problems

This chapter investigates XCS's performance in various real- and/or nominal valued datasets. The application to other than binary valued problems requires a modification of the XCS classifier system condition parts as well as its genetic operators including covering, mutation, and crossover.

Besides the representational and operator enhancements, we revisit the established facetwise XCS theory in the real-valued and nominal problem domain. Essentially, it is necessary to revise the specificity definition for the new representation of classifier condition parts. Also the frequency of niche occurrence needs to be redefined due to the real-valued features and should be based on volume times sampling distribution.

Our performance analysis shows that XCS is able to learn competitively in various datasets taken from the UCI repository and other sources. In the analysis, we compare XCS's performance with many other established machine learning systems.

The main objective of this chapter is to confirm that XCS is a valuable learning system. The reader should keep in mind, though, that XCS is an online learning RL system and not a pure datamining system. Thus, the competitive results confirm XCS's learning competence. The real potential of the XCS mechanism, however, may rather lie in more reinforcement-based, online learning tasks as addressed in the subsequent chapters.

This chapter first introduces the necessary enhancements to XCS to be able to handle nominal values, real values, as well as a mixture of both. Section 8.2 outlines the consequently necessary theory modifications to ensure covering, schema supply, reproductive opportunities, and solution support in real-valued domains. Next, we investigate XCS's performance in a large set of datamining problems. Summary and conclusions discuss the results from a broader perspective.

8.1 XCS Enhancements

XCS for real valued problem domains was introduced elsewhere (Wilson, 2000; Wilson, 2001b). The resulting system, often referred to as XCSR, is modified in the representation of a condition as well as the operation of mutation and covering. Stone and Bull (2003) showed that the originally proposed center-spread method biases the learner towards learning solution value boundaries, which is not desirable in the general case. Thus, we apply the method that codes a condition part as a conjunction of intervals, as described in Wilson (2001b).

Particularly, the condition part consists of a conjunction of intervals represented by upper and lower boundaries (l_i, u_i). A condition matches a current real-valued problem instance $s \in \mathcal{S}$ if the problem instance lies within all intervals specified by the classifier condition.

Mutation mutates the lower or upper bound with probability μ. If mutation is applied, the bound is increased or decreased by a value uniformly randomly picked between zero and m_0. In Wilson's original paper parameter m_0 was an absolute value equal in all problem features. To be more independent of the problem domain, we chose to define m_0 as the relative fraction of the feature range. For example, if a feature may take values between 0 and 10 and $m_0 = 0.2$, then the mutation interval ranges between zero and two. If mutation mutates an attribute below the lower value boundary of the problem or above the upper value boundary, the new bound is set to the lower or upper boundary, respectively. If mutation mutates the lower boundary above the upper boundary or vice versa, the boundary values are swapped.

The covering operator is defined similar to the binary covering operator. Given a currently uncovered problem instance, each feature is covered generating an interval for which the lower bound is chosen uniformly randomly between zero and r_0 distance from the current value and the upper bound is similarly chosen above the current value. Although r_0 could also be defined relative to the problem domain as done for mutation, due to its importance solely in the beginning of a run we chose not to do so.

Note that the enhancements are directly taken from Wilson's publication (except for the relative mutation). Certainly, other types of mutation such as a Gaussian mutation approach as used in evolution strategies (Rechenberg, 1973; Bäck & Schwefel, 1995) might result in additional learning performance improvement. However, such enhancements are left for future research work.

In the case of nominal values, we apply the same enhancements in that we convert the nominal values into an integer representation. The operators are then applied similar to the

real-valued case only that mutation, if applied, increases or decreases the boundary by at least one unit. In the case of only two nominal values, mutation is applied equivalently to the binary case.

8.2 Theory Modifications

Facing a new representation, the notion of niche occurrence, representatives, as well as specificity needs to be redefined. We are making use of the notion of volume to define problem subspaces as well as specificity. To keep the notation simple, we assume that each real-valued attribute ranges between zero and one.

With this constrained, we can define the volume of a classifier condition C by

$$\texttt{Vol}(C) = \prod_{i=1}^{l} (u_i - l_i). \tag{8.1}$$

The volume effectively defines the size of the subspace in which the classifier matches. Assuming a uniform random distribution of problem instances over the whole problem space, the definition of volume coincides with the probability of matching a problem instance. Consequently, the completely general classifier, in which all upper boundaries are equal to one and all lower boundaries are equal to zero, has a volume of one.

8.2.1 Evolutionary Pressure Adjustment

With respect to the previous specificity equation, few adjustments are necessary. Set pressure applies similarly to the binary case. Assuming a Gaussian distribution over specificities (which is exactly the case when sampling uniformly randomly and ignoring boundary effects), the resulting expected specificity in an action set (match set) is determined by multiplying the Gaussian with the probability of matching (which is equal to one minus the generality, which is equal to one minus the volume). Thus, the specificity in an action set can be expected to be similarly decreased.

Mutation, however, does not have any effect on specificity in the used definition when disregarding boundary effects. Since the increase of a boundary is as likely as the decrease of a boundary, the overall interval size is expected to stay the same. However, if a boundary is very close to the lower bound (upper bound) or essentially equal, mutation has a tendency to specialize since further generalization has no effect. Similarly, if the current interval is very small, mutation tends to generalize. This was also investigated in Stone and Bull (2003)

from a somewhat different perspective. As in the binary case, crossover has no effect on generality in the real valued case.

In sum, the previously analyzed set pressure is also present in real valued problems. Since mutation does not have an impact on specificity except when close to the boundaries, XCS can be expected to evolve fairly general classifiers as long as no fitness pressure applies.

Subsumption propagates syntactically more general classifiers as in the binary case. Since especially in datamining problems problem instances are usually not uniformly distributed over the problem space, the semantic generalization pressure (that is, the set pressure) does not apply as prominently as in the binary problems investigated. Thus, subsumption pressure may have stronger generalization impacts as long as the problem allows the evolution of completely accurate classifiers.

8.2.2 Population Initialization: Covering Bound

With the notion of volume, we can derive the probability that a covering classifier exists given a certain population size N and the covering operator r_0. In particular, a randomly generated classifier has an average volume of r_0^l since the interval starts on average half r_0 below the current value and stop half r_0 above the current value disregarding boundary effects. Note that if a value would be circular in that the lower value equates the highest value, then there were no boundary effects and r_0 would be the exact expectable interval.

Classifiers in a randomly initialized population can be consequently expected to match with probability r_0^l. Similar to the covering bound in binary domains, we can derive a covering bound for real-valued domains as:

$$P(\texttt{cover}_r) = 1 - (1 - r_0^l)^N \tag{8.2}$$

To ensure covering in real-valued domains, operator r_0 should consequently be set high enough. For example, requiring a certain confidence θ that a current problem instance is covered in the population given a fixed population size N, r_0 should be set as:

$$r_0 > \sqrt[l]{1 - \sqrt[N]{1-\theta}} \tag{8.3}$$

Due to boundary effects, some intervals generated during covering can be expected to be smaller so that the actual value of r_0 should be chosen slightly larger.

8.2.3 Schema Supply and Growth: Schema and Reproductive Opportunity Bound

To derive the schema and reproductive opportunity bound for the real-valued domain, it is necessary to re-define a representative of a certain problem niche. Our intuitive definition in Chapter 5 may carry over in that a classifier may be considered a representative if it is at least as specific as the niche it represents. On the other hand, a slightly more general classifier may also be considered a representative if it does not overlap with any problem instance that belongs to another class. In general, since decision boundaries are never as clearcut as in the binary domain, the definition of a representative becomes less exact.

Additionally, the minimal order of problem difficulty k_m is not definable. In fact, it appears to vanish since the real valued representation allows any kind of overlap so that the class distribution of a classifier cannot be approximated as easily. The maximally difficult problem appears to be a highly separated checker-board problem as illustrated in Figure 8.1. As long as the covering operator is not chosen sufficiently specific, XCS can be expected to get stuck with an overgeneral, highly inaccurate representation since there is no specialization pressure towards evolving a more distributed problem representation unless a near-exact subsolution is identified (that is, one small rectangle in the checker problem).

The problem becomes even more clearly pronounced in the problem shown in Figure 8.2. Starting for example with maximally general classifiers, XCS can be expected to evolve the shown class boundary $y = .4375$. However, the small additional positive cases in the upper left part of the problem space are very hard to identify and separate from the rest of the problem space. In the shown problem, the classifier that considers the complete space x and the space above $y = .4375$ is more accurate than any other classifier that specifies a subspace above $y = .4375$ and comprises the positive cases in the upper left except for a classifier that singles out the positive cases (rectangles). Thus, fitness guidance is missing and the problem's minimal order specificity k_m becomes apparent.

In this problem case, default hierarchies, which were suggested a long time ago for LCS systems (Holland, 1975), may do the job. Since the accuracy is high in a classifier that identifies the upper part of the problem space, it might be maintained as the default rule. However, since it is not always correct, classifiers that identify subspaces and particularly incorrectly specified problem cases may be able to solve the problem completely accurately. Another option would be to induce highly specialized classifiers in exceptional cases in which a general near-accurate classifier has an incorrect prediction. However, an inappropriate hunt for a classification of incorrect, noisy problem instances needs to be avoided. Given

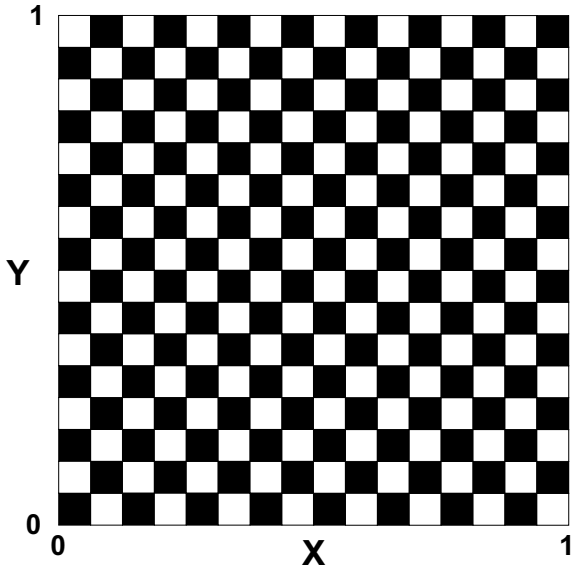


Figure 8.1: In a highly scattered checker board problem, it becomes extremely hard for XCS to evolve accurate classifiers when starting with overgeneral ones.

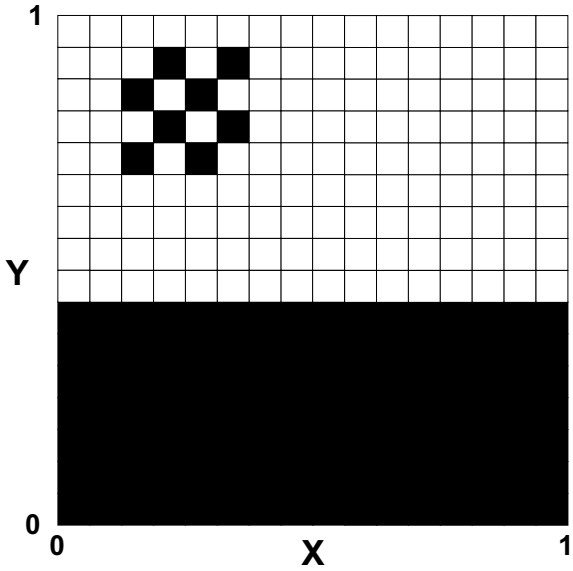


Figure 8.2: Starting from the overgeneral side, XCS faces a challenge if the problem space is highly unequally distributed and overgeneral classifiers prevent the evolution of more specialized classifiers.

knowledge of the amount of noise in a problem, it would be possible to apply more directed search operators. In the general reinforcement-based learning case, however, such special operators can be expected to do more harm than to result in any performance improvement.

8.2.4 Solution Sustenance: Niche Frequencies

Once an accurate representation evolved, the niche bound carries over rather directly. Given a niche of a certain volume and given further the maximally accurate classifier that covers that niche, the classifier will undergo the same Markov-chain reproduction and deletion process identified in Chapter 5. Thus, similar to the binary case, we can define the problem difficulty as the volume of the smallest problem niche. Since the volume equates the probability of niche occurrence p (as long as problem instances are uniformly randomly sampled from the whole problem space), the population size bound with respect to the niche bound is identical to the one derived for the binary case.

8.2.5 Obliqueness

Certainly there is an additional problem with respect to real-valued problem domains which is the problem of obliqueness. Our current representation of condition parts does not allow us to represent oblique decision boundaries. XCS approximates an oblique boundary using a piece-wise approach separating the problem space into small hypercubes. Again the system faces the problem of reproductive opportunities and overgenerality. Since the boundaries are oblique, specializations may not result directly in an improvement of accuracy. If a more specialized classifier was found, the competition with the slightly less accurate classifier is a challenging one requiring several evaluations before the superiority of the more specialized classifier is detected. In effect, the more accurate classifier is likely to be deleted before undergoing any reproductive events.

The exact formulation of this problem and the resulting population size requirements are not investigated any further herein. Instead, the next section investigates XCS's performance in its current form. The results confirm that even in its current form, XCS shows to be able to solve datamining problems machine learning competitively.

8.3 Datamining

Due to the variable properties of the investigated datasets including real values, nominals, and binary features, we use a hybrid XCS/XCSR approach that can handle any feature combination—as done before elsewhere (Bernadó, Llorà, & Garrell, 2002; Bernadó-Mansilla & Garrell-Guiu, 2003). In essence, each condition attribute is handled separately during matching, mutation, and covering dependent on the type of problem feature.

The datamining analysis has two main objectives: (1) to compare XCS's performance of tournament selection with that of proportionate selection; (2) to compare XCSTS's performance with that of other machine learning algorithms. We first introduce the datasets that we investigated. Next, we present the results.

8.3.1 Datasets

In Table 8.1 we show the datasets we have selected including datasets from the University of California at Irvine (UCI) repository (Blake, Keogh, & Merz, 1998) as well as a few other datasets. The other datasets are the `led.noise10` dataset which codes the seven lines of an LED display in binary. Additional 10% noise is added to the instances (features and class) to evaluate the approximation of the algorithm. The set was frequently used in the literature

to investigate the learning performance on a dataset with added artificial noise. Llorà and Goldberg (2002) showed that the maximal performance achievable with 10% noise is 75% accuracy. Secondly, the `tao` problem was previously investigated with the XCS system and other learning systems (Bernadó, Llorà, & Garrell, 2002; Bernadó-Mansilla & Garrell-Guiu, 2003). The problem is a dataset sampling uniformly randomly from the tao figure where white areas are assigned class zero and black areas class one. The dataset consequently has only oblique decision boundaries making it hard to learn with linear separators such as the interval coding we use in the XCS system.

Table 8.1 shows that the investigated datasets have various properties consisting of real, integer, nominal, and binary features. Hereby, nominal and integer features that only have two values are counted as binary features. Each dataset has a different number of features, of problem instances, and of solution classes. Also the number and distribution of instances per class vary. Finally, the number of missing values in the datasets vary.

To evaluate the performance of a learner, we apply stratified 10-fold cross-validation experiments (Mitchell, 1997).

With respect to the XCS application, we need to mention that XCS currently uses a simple strategy for missing values in a dataset simply assuming a match to a missing value. Other strategies might be superior to this one as also suggested by our recent datamining comparison with the Pittsburgh-style learning classifier system GAssist (Bacardit & Butz, 2004). Additionally, nominal values are coded as integers. This might lead to performance drawbacks since the resulting integer ordering might be inappropriate with respect to the problem. Other set-based approaches might be more appropriate but are not investigated further herein. It also needs to be remembered that we code nominal features with only two values identical to binary features in contrast to previous datamining investigations with XCS (Bernadó, Llorà, & Garrell, 2002; Bernadó-Mansilla & Garrell-Guiu, 2003).

8.3.2 Results

Table 8.2 compares XCS's performance with proportionate selection with XCS's performance with tournament selection (XCSTS). The table clearly confirms the superiority of tournament selection proving initial faster learning as well as better convergence properties. Additionally, the results show that $100k$ learning steps may not be enough to reach optimal performance. Besides the different number of learning steps and the different initializations, we also tested XCS on two covering parameter settings causing the initial population to be either very general ($r_0 = 100$) or very specific ($r_0 = 4$). It can be seen that a more general initialization

Table 8.1: The dataset properties indicate the number of problem instances (#Inst), the number of features (#Fea), of real-valued features (#Re), of integer-valued features (#In), of nominal features (#No), of binary features (#Bi), of classes (#Cl) as well as the percentage of instances belonging to the smallest class (%CMi) and the majority class (%CMa), of instances with missing values (%MVi), of features with missing values (%MVf), and of missing value values (#MVv).

Domain	#Inst.	#Fea	#Re	#In	#No	#Bi	#Cl	%CMi	%CMa	%MVi	%MVf	%MVv
anneal	898	38	6	0	13	19	5	0.9	76.2	0.0	0.0	0.0
audiology	226	69	0	0	8	61	24	0.4	25.2	98.2	10.1	2.0
autos	205	25	15	0	6	4	6	1.5	32.7	22.4	28.0	1.2
balance-scale	625	4	4	0	0	0	3	7.8	46.1	0.0	0.0	0.0
breast-cancer	286	9	0	0	6	3	2	29.7	70.3	3.1	22.2	0.3
breast-w	699	9	0	9	0	0	2	34.5	65.5	2.3	11.1	0.3
bupa	345	6	6	0	0	0	2	42.0	58.0	0.0	0.0	0.0
cmc	1473	9	2	0	4	3	3	22.6	42.7	0.0	0.0	0.0
colic	368	22	7	0	13	2	2	37.0	63.0	98.1	95.5	23.8
credit-a	690	15	6	0	5	4	2	44.5	55.5	5.4	46.7	0.6
credit-g	1000	20	7	0	11	2	2	30.0	70.0	0.0	0.0	0.0
diabetes	768	8	8	0	0	0	2	34.9	65.1	0.0	0.0	0.0
glass	214	9	9	0	0	0	6	4.2	35.5	0.0	0.0	0.0
heart-c	303	13	6	0	4	3	2	45.5	54.5	2.3	15.4	0.2
heart-c1	296	13	6	0	4	3	2	45.9	54.1	0.0	0.0	0.0
heart-h	294	13	6	0	4	3	2	36.1	63.9	99.7	69.2	20.5
heart-statlog	270	13	13	0	0	0	2	44.4	55.6	0.0	0.0	0.0
hepatitis	155	19	2	4	0	13	2	20.6	79.4	48.4	78.9	5.7
hypothyroid	3772	29	6	1	2	20	4	0.1	92.3	100.0	27.6	5.5
ionosphere	351	34	34	0	0	0	2	35.9	64.1	0.0	0.0	0.0
iris	150	4	4	0	0	0	3	33.3	33.3	0.0	0.0	0.0
kr-vs-kp	3196	36	0	0	2	34	2	47.8	52.2	0.0	0.0	0.0
labor	57	16	8	0	5	3	2	35.1	64.9	98.2	100.0	35.7
led.noise10	2000	7	0	0	0	7	10	8.9	10.9	0.0	0.0	0.0
lymph	148	18	0	3	6	9	4	1.4	54.7	0.0	0.0	0.0
mushroom	8124	22	0	0	18	4	2	48.2	51.8	30.5	4.5	1.4
new-thyroid	215	5	5	0	0	0	3	14.0	69.8	0.0	0.0	0.0
primary-tumor	339	17	0	0	3	14	21	0.3	24.8	61.1	29.4	3.9
segment	2310	19	19	0	0	0	7	14.3	14.3	0.0	0.0	0.0
sick	3772	29	6	1	2	20	2	6.1	93.9	100.0	27.6	5.5
sonar	208	60	60	0	0	0	2	46.6	53.4	0.0	0.0	0.0
soybean	683	35	0	0	19	16	19	1.2	13.5	17.7	97.1	9.8
splice	3190	60	0	0	60	0	3	24.0	51.9	0.0	0.0	0.0
tao	1888	2	2	0	0	0	2	50.0	50.0	0.0	0.0	0.0
vehicle	846	18	18	0	0	0	4	23.5	25.8	0.0	0.0	0.0
vote	435	16	0	0	0	16	2	38.6	61.4	46.7	100.0	5.6
vowel	990	13	10	0	1	2	11	9.1	9.1	0.0	0.0	0.0
waveform-5000	5000	40	40	0	0	0	3	33.1	33.8	0.0	0.0	0.0
wdbc	569	30	30	0	0	0	2	37.3	62.7	0.0	0.0	0.0
wine	178	13	13	0	0	0	3	27.0	39.9	0.0	0.0	0.0
wpbc	198	33	33	0	0	0	2	23.7	76.3	2.0	3.0	0.1
zoo	101	17	0	1	1	15	7	4.0	40.6	0.0	0.0	0.0

is often advantageous preventing over-specializations. However, in some datasets such as the tao problem an initial more specialized population is advantageous. This observation confirms the facetwise theory extension from the last section. The oblique boundaries in the tao problem are very hard to improve upon once a general solution was learned. Starting with an initially fairly specific population assures that the oblique boundaries are estimated more accurately consequently reaching and maintaining a higher performance level.

We compared XCSTS with a number of other machine learning programs including C4.5 (revision 8) (Quinlan, 1993), the Naive Bayes classifier (John & Langley, 1995), PART (Frank & Witten, 1998), the instance based learning algorithm with one and three nearest neighbor setting (Aha, Kibler, & Albert, 1991; Mitchell, 1997) and the support vector machine implementation SMO with polynomial kernels of order one and three and with radial basis kernels (Platt, 1998). To establish a performance baseline, we also provide results for the simple majority class algorithm which chooses the majority class as the classification throughout. Moreover, to provide another, more challenging baseline result, we also ran the simple 1-R classifier that learns one rule that is conditioned on the most significant problem features, ranking all features according to their train-set error and dividing continuous values straightforwardly requiring a minimal number of six instances of the optimal class in the interval (Holte, 1993). All machine learning methods were run using the WEKA machine learning package (Witten & Frank, 2000).

Performance comparisons confirm that XCS performs competitively to these other machine learning algorithms (tables 8.3 and 8.4). Statistics are based on eight stratified 10-fold crossvalidation runs (that is, 80 experiments in each dataset). XCSTS learned for $500k$ steps in the presented results. As can be expected given the number of different datasets, XCSTS outperforms the other learners in some datasets whereas it is outperformed in others. Interestingly, XCSTS is outperformed in the soybean dataset by all other learning mechanisms (except majority and 1-R). Considering the soybean properties, two observations can explain this finding: (1) The soybean dataset consists of lots of nominal features and the conversion to an integer representation may be unfavorable in this dataset; (2) Many instances have missing values in this problem. The missing value strategy currently implemented in XCS of simply assuming a match in a missing feature might be inappropriate in this dataset.

8.4 Summary and Conclusions

This chapter has shown that XCS can be successfully applied to various datamining problems yielding competitive performance in comparison to various other machine learning al-

Table 8.2: Crossvalidation performance of XCS comparing tournament selection (TS) with proportionate selection (PS) in diverse datasets after $100k$ learning iterations. Initialization is either general ($r_0 = 100$) or specific ($r_0 = 4$). The last column shows results of longer XCSTS runs ($500k$ learning iterations) comparing the results with the shorter runs. The ♦ (◊) and ■ (□) symbol indicate in which problems the first (second) column performance is significantly better/worse on a significance level of .99 and .95 (pairwise t-test), respectively. The final two lines summarize the performance scores.

Database	TS($r_0 = 100$)	TS($r_0 = 4$)	PS($r_0 = 100$)	PS($r_0 = 4$)	TS($r_0 = 100$,$t = 500k$)
anneal	91.2±2.7	91.7±2.9	92.9±2.7 ◊	91.4±3.2	97.7±2.0 ◊
audiology	73.8±12.6	74.1±12.8	73.5±12.6	73.9±12.8	79.6±12.3 ◊
autos	64.7±9.6	13.4±6.9 ♦	62.0±10.3 ■	14.7±7.3	71.2±9.9 ◊
balance-scale	84.6±3.3	82.0±3.5 ♦	83.6±4.0 ■	83.6±3.0 ◊	81.1±3.8 ♦
breast-cancer	71.8±6.6	68.5±8.0 ♦	72.6±5.3	69.0±7.6	70.1±8.0 ■
breast-w	96.2±2.2	96.5±1.9	96.1±2.4	96.2±2.1	95.9±2.3
bupa	67.2±7.9	58.7±8.5 ♦	65.8±7.7	56.0±8.2 ♦	67.1±7.5
cmc	50.1±4.7	53.6±3.6 ◊	43.3±5.7 ♦	53.5±3.7	52.4±3.6 ◊
colic	84.5±5.8	84.8±5.5	84.3±6.2	84.8±6.1	84.0±5.8
credit-a	86.2±3.9	85.3±4.1 ■	85.7±3.9	85.1±3.8	85.6±3.5
credit-g	71.4±3.8	72.5±3.1	71.8±3.6	71.4±2.9 ♦	70.9±4.3
diabetes	74.3±4.5	67.3±3.5 ♦	74.4±4.5	65.6±2.6 ♦	72.4±5.3 ♦
glass	70.6±8.2	70.7±8.4	54.5±9.7 ♦	70.8±9.7	71.8±8.9
heart-c1	77.7±6.8	68.9±8.6 ♦	77.7±7.5	65.6±8.9 ♦	76.5±7.9
heart-c	77.4±7.9	68.1±8.5 ♦	77.0±7.6	66.0±8.2	77.2±6.9
heart-h	79.4±7.7	70.8±6.9 ♦	80.8±7.1 □	70.1±6.3	77.8±8.0
heart-statlog	77.3±7.6	67.9±9.0 ♦	77.9±7.1	68.0±8.3	75.3±8.1
hepatitis	81.8±8.6	80.3±7.7	79.6±9.3 ■	79.9±8.4	80.7±9.2
hypothyroid	98.7±0.7	98.9±0.7	98.1±1.3 ♦	98.5±0.7 ♦	99.5±0.4 ◊
ionosphere	90.7±5.3	57.1±6.8 ♦	89.7±4.8	57.2±6.1	90.1±4.7
iris	92.6±7.2	95.0±5.0 ◊	84.7±12.8 ♦	95.2±5.0	94.7±5.1 □
kr-vs-kp	98.7±0.7	99.0±0.6 ◊	97.7±0.8 ♦	98.3±0.8 ♦	98.9±0.6 □
labor	80.9±15.5	86.6±15.3 ◊	70.0±12.0 ♦	86.4±14.7	83.5±14.8
led.noise10	72.1±3.7	72.1±3.8	67.4±5.3 ♦	67.5±5.4 ♦	72.7±3.1
lymph	81.7±9.5	83.5±9.5	81.2±10.2	81.9±10.2	79.8±10.2
mushroom	99.8±0.4	100.0±0.0 ◊	99.1±0.5 ♦	100.0±0.0	100.0±0.0 ◊
new-thyroid	94.8±4.5	94.8±4.8	93.9±5.2	94.4±4.6	95.4±4.6
primary-tumor	39.7±7.5	39.9±7.3	39.4±7.7	40.2±8.0	39.8±8.5
segment	92.5±1.7	16.8±1.1 ♦	92.2±1.9	16.8±1.1	95.8±1.3 ◊
sick	94.2±1.0	98.4±0.6 ◊	93.9±0.1 ♦	98.2±0.7	94.9±1.7 ◊
sonar	77.3±8.1	81.6±7.9 ◊	75.0±8.6	80.7±9.7	77.9±8.0
soybean	76.1±5.9	85.1±3.8 ◊	70.3±6.6 ♦	82.9±4.2 ♦	85.1±4.4 ◊
splice	88.9±4.2	28.0±1.0 ♦	73.6±14.4 ♦	28.1±0.9 ◊	93.8±1.5 ◊
tao	84.3±4.1	94.4±1.6 ◊	82.1±4.7 ♦	93.8±1.6 ♦	86.4±4.5 ◊
vehicle	73.8±4.4	25.1±0.5 ♦	73.4±4.0	25.1±0.5	74.3±4.7
vote	95.6±3.2	95.8±3.1	96.3±2.8 ◊	96.1±3.0	95.7±3.1
vowel	56.1±7.5	33.3±4.4 ♦	51.8±6.9 ♦	33.8±4.9	66.0±6.6 ◊
waveform-5000	82.5±1.7	33.8±0.0 ♦	82.2±1.5	33.8±0.0	82.5±1.5
wdbc	95.8±2.6	92.9±3.3 ♦	95.3±2.7	91.4±4.2 ♦	96.0±2.5
wine	95.8±4.7	97.1±4.0 ◊	93.6±5.8 ♦	95.4±5.3 ♦	95.6±4.9
wpbc	73.0±8.4	23.7±1.8 ♦	73.0±8.7	23.8±2.0	74.3±8.9
zoo	94.6±6.9	93.1±6.9	92.2±7.0 ♦	92.9±7.5	95.1±6.1
score 99%		17/10	15/2	11/2	2/12
score 95%		18/10	18/3	11/2	3/14

Table 8.3: Comparing XCSTS's performance with other typical machine learning algorithms confirms its competitiveness. The ♦ (◊) and ■ (□) symbol indicate in which problems XCSTS performed significantly better/worse on a significance level of .99 and .95 (pairwise t-test), respectively. The final two lines summarize the performance scores.

Database	XCSTS	Majority	Main Ind.	C4.5	Naive Bayes	PART
anneal	97.7±2.0	76.2±0.8 ♦	83.6±0.5 ♦	98.6±1.1	86.6±3.4 ♦	98.4±1.3
audiology	79.6±12.3	26.9±5.5 ♦	49.6±10.2 ♦	81.2±11.0	76.8±15.4	83.3±12.6 ◊
autos	71.2±9.9	32.8±2.1 ♦	61.4±10.1 ♦	82.0±8.7 ◊	56.8±9.9 ♦	74.6±8.5
balance-scale	81.1±3.8	46.1±0.2 ♦	57.9±4.0 ♦	77.9±4.0 ♦	90.5±1.8 ◊	82.4±4.0
breast-cancer	70.1±8.0	70.3±1.2	67.0±6.8	73.8±6.2 □	72.7±8.1	69.8±7.1
breast-w	95.9±2.3	65.5±0.3 ♦	91.9±2.9 ♦	94.5±2.6 ♦	96.0±2.1	94.7±2.4 ■
bupa	67.1±7.5	58.0±0.8 ♦	55.9±7.1 ♦	65.0±8.6	54.8±8.3 ♦	64.3±8.2
cmc	52.4±3.6	42.7±0.2 ♦	47.5±3.7 ♦	51.6±4.1	50.4±4.1	50.0±4.0 ♦
colic	84.0±5.8	63.1±0.8 ♦	81.5±6.3	85.2±5.4	78.3±6.4 ♦	84.4±5.4
credit-a	85.6±3.5	55.5±0.4 ♦	85.5±4.1	85.4±4.2	77.8±4.2 ♦	84.0±4.5
credit-g	70.9±4.3	70.0±0.0	66.0±3.4 ♦	71.1±3.6	75.1±3.7 ◊	70.0±3.9
diabetes	72.4±5.3	65.1±0.3 ♦	72.2±4.2	74.2±4.6	75.6±4.9 ◊	73.9±4.6
glass	71.8±8.9	35.6±1.4 ♦	56.6±9.4 ♦	67.4±8.8	48.1±8.2 ♦	68.3±8.5
heart-c1	76.5±7.9	54.1±0.9 ♦	73.7±7.2	76.4±8.2	83.3±6.0 ◊	78.3±7.2
heart-c	77.2±6.9	54.5±0.7 ♦	72.7±7.0 ■	77.2±6.9	83.5±6.2 ◊	78.1±7.1
heart-h	77.8±8.0	64.0±0.9 ♦	80.7±7.1	79.5±7.9	84.3±7.5 ◊	80.3±7.7
heart-statlog	75.3±8.1	55.6±0.0 ♦	72.0±8.0	77.8±8.0	83.9±6.4 ◊	77.4±7.3
hepatitis	80.7±9.2	79.4±1.5	82.5±7.0	79.0±9.2	83.6±9.2	79.8±7.8
hypothyroid	99.5±0.4	92.3±0.3 ♦	96.3±0.9 ♦	99.5±0.3	95.3±0.6 ♦	99.5±0.4
ionosphere	90.1±4.7	64.1±0.6 ♦	81.8±6.0 ♦	90.0±5.3	82.5±7.1 ♦	90.7±5.2
iris	94.7±5.1	33.3±0.0 ♦	93.3±5.9	94.8±5.9	95.5±5.1	94.4±6.4
kr-vs-kp	98.9±0.6	52.2±0.1 ♦	67.1±1.8 ♦	99.4±0.4 ◊	87.8±1.9 ♦	99.1±0.6
labor	83.5±14.8	64.7±3.1 ♦	72.3±14.1 ♦	79.2±14.7	94.1±8.7 ◊	78.1±15.2
led.noise10	72.7±3.1	10.8±0.1 ♦	19.5±1.0 ♦	74.8±2.8 ◊	75.5±2.8 ◊	74.1±3.0
lymph	79.8±10.2	55.0±2.9 ♦	74.8±11.9	77.7±11.1	83.0±8.6	76.9±10.0
mushroom	100.0±0.0	51.8±0.0 ♦	98.5±0.4 ♦	100.0±0.0	95.8±0.7 ♦	100.0±0.0
new-thyroid	95.4±4.6	69.8±1.6 ♦	91.4±6.5 ♦	92.5±6.3 ♦	97.0±3.3	94.0±5.5
primary-tumor	39.8±8.5	25.3±3.2 ♦	28.0±4.9 ♦	43.1±7.3	50.7±8.7 ◊	41.8±7.6
segment	95.8±1.3	14.3±0.0 ♦	64.4±2.6 ♦	96.8±1.1 ◊	80.1±1.8 ♦	96.5±1.2 ◊
sick	94.9±1.7	93.9±0.1 ♦	96.2±0.8 ◊	98.7±0.5 ◊	92.8±1.3 ♦	98.6±0.6 ◊
sonar	77.9±8.0	53.4±1.2 ♦	62.1±9.5 ♦	73.8±8.5	67.8±9.6 ♦	74.4±9.5
soybean	85.1±4.4	13.5±0.5 ♦	39.8±2.6 ♦	91.9±3.2 ◊	92.8±2.8 ◊	91.4±3.1 ◊
splice	93.8±1.5	51.9±0.1 ♦	63.4±1.6 ♦	94.2±1.3	95.5±1.3 ◊	92.6±1.4 ♦
tao	86.4±4.5	50.0±0.0 ♦	72.3±3.0 ♦	95.6±1.3 ◊	80.9±2.9 ♦	94.2±2.1 ◊
vehicle	74.3±4.7	25.6±0.3 ♦	52.8±4.7 ♦	72.4±4.1	45.1±4.7 ♦	72.0±4.0 ■
vote	95.7±3.1	61.4±0.4 ♦	95.6±3.1	96.5±2.8	90.1±4.5 ♦	96.0±3.0
vowel	66.0±6.6	9.1±0.0 ♦	33.5±4.5 ♦	80.1±3.6 ◊	63.2±4.9	78.5±4.0 ◊
waveform-5000	82.5±1.5	33.8±0.0 ♦	53.9±1.9 ♦	75.2±2.0 ♦	80.0±1.5 ♦	77.9±1.7 ♦
wdbc	96.0±2.5	62.7±0.4 ♦	88.5±3.5 ♦	93.5±3.2 ♦	93.4±3.5 ♦	94.1±3.0 ♦
wine	95.6±4.9	39.9±1.7 ♦	78.0±9.7 ♦	93.3±6.5	97.2±4.0	92.4±6.1 ♦
wpbc	74.3±8.9	76.3±1.8	68.8±6.1 ♦	72.7±8.6	67.1±10.4 ♦	74.2±9.0
zoo	95.1±6.1	41.5±5.9 ♦	43.1±4.9 ♦	93.0±6.8	95.7±6.0	93.7±6.7
score 99%		38/0	29/1	5/8	19/12	5/6
score 95%		38/0	30/1	5/9	19/12	7/6

Table 8.4: Continued comparison of XCSTS with other machine learning algorithms. The ♦ (◊) and ■ (□) symbol indicate in which problems XCSTS performed significantly better/worse on a significance level of .99 and .95 (pairwise t-test), respectively. The final two lines summarize the performance scores.

Database	XCSTS	Inst.b.1	Inst.b.3	SMO(poly.1)	SMO(poly.3)	SMO(radial)
anneal	97.7±2.0	99.1±1.0 ◊	97.2±1.6	97.2±1.5	99.2±0.8 ◊	91.9±1.8 ♦
audiology	79.6±12.3	77.4±9.4	71.1±12.4 ♦	84.4±11.7 ◊	82.3±12.3	46.8±10.1 ♦
autos	71.2±9.9	73.5±9.3	67.4±10.0	70.6±9.1	77.2±9.1 ◊	45.5±6.8 ♦
balance-scale	81.1±3.8	78.5±4.2 ♦	86.7±3.0 ◊	87.7±2.4 ◊	90.7±2.8 ◊	87.8±2.7 ◊
breast-cancer	70.1±8.0	68.6±7.8	73.7±5.1 □	70.1±7.3	66.8±7.6	70.3±1.2
breast-w	95.9±2.3	95.6±2.1	96.6±2.0	96.7±1.9	95.9±2.1	96.0±2.2
bupa	67.1±7.5	62.2±7.8 ■	62.5±7.6	57.9±1.5 ♦	59.1±3.4 ♦	58.0±0.8 ♦
cmc	52.4±3.6	44.3±3.8 ♦	46.7±3.6 ♦	48.9±3.6 ♦	49.1±3.7 ♦	42.8±1.1 ♦
colic	84.0±5.8	79.0±5.7 ♦	81.4±5.7	82.8±6.2	77.6±6.7 ♦	84.1±5.3
credit-a	85.6±3.5	81.6±4.7 ♦	85.0±4.1	84.9±3.8	81.8±4.6 ♦	85.5±4.1
credit-g	70.9±4.3	72.0±3.9	72.7±3.4	74.9±3.7 ◊	70.6±4.0	70.1±0.2
diabetes	72.4±5.3	70.4±4.4	73.9±4.5	77.0±4.3 ◊	76.9±4.1 ◊	65.1±0.3 ♦
glass	71.8±8.9	70.3±9.1	70.6±8.4	57.0±7.7 ♦	65.7±8.9 ♦	35.6±1.4 ♦
heart-c1	76.5±7.9	75.8±6.9	82.2±6.5 ◊	83.5±6.1 ◊	76.6±7.8	82.8±6.6 ◊
heart-c	77.2±6.9	75.7±7.7	82.3±6.5 ◊	83.4±6.5 ◊	77.9±6.8	83.2±6.3 ◊
heart-h	77.8±8.0	78.6±6.8	82.2±7.4 □	82.7±8.0 ◊	78.0±7.5	82.0±7.4 □
heart-statlog	75.3±8.1	75.1±7.5	79.2±7.2 □	83.8±6.3 ◊	79.0±6.5 ◊	82.9±6.5 ◊
hepatitis	80.7±9.2	80.9±8.3	80.2±8.6	85.6±8.7 □	82.7±7.9	79.4±1.5
hypothyroid	99.5±0.4	91.4±1.0 ♦	93.2±0.8 ♦	93.6±0.5 ♦	93.9±0.6 ♦	92.3±0.3 ♦
ionosphere	90.1±4.7	86.7±5.1 ♦	86.0±5.0 ♦	87.8±5.0	88.0±5.8	75.7±4.7 ♦
iris	94.7±5.1	95.4±5.2	95.2±5.4	96.3±4.6	92.9±6.0	90.6±9.0 ■
kr-vs-kp	98.9±0.6	90.5±1.6 ♦	96.5±1.1 ♦	95.8±1.2 ♦	99.6±0.4 ◊	91.4±1.6 ♦
labor	83.5±14.8	85.7±13.6	93.6±9.1 ◊	93.0±10.7 ◊	91.8±11.5 □	64.7±3.1 ♦
led.noise10	72.7±3.1	65.3±6.2 ♦	74.8±2.7 ◊	75.4±2.8 ◊	74.1±2.9	75.4±2.9 ◊
lymph	79.8±10.2	81.8±9.7	82.3±8.7	86.0±8.0 ◊	83.4±9.0	81.5±10.4
mushroom	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	99.8±0.1 ♦
new-thyroid	95.4±4.6	96.6±4.1	94.2±4.8	89.6±6.0 ♦	89.4±6.3 ♦	69.8±1.6 ♦
primary-tumor	39.8±8.5	34.3±7.5 ♦	46.0±7.6 ◊	47.9±7.8 ◊	42.3±7.2	25.3±3.2 ♦
segment	95.8±1.3	97.1±1.2 ◊	96.1±1.3	92.9±1.4 ♦	96.4±1.2	82.0±2.2 ♦
sick	94.9±1.7	96.1±0.8 ◊	96.1±0.8 ◊	93.9±0.1 ♦	95.9±0.7 ◊	93.9±0.1 ♦
sonar	77.9±8.0	86.4±7.0 ◊	83.5±7.7 ◊	77.5±9.1	85.1±8.0 ◊	68.6±7.4 ♦
soybean	85.1±4.4	90.2±3.1 ◊	91.1±3.0 ◊	92.7±2.7 ◊	93.2±2.7 ◊	87.9±2.6 ◊
splice	93.8±1.5	75.8±2.6 ♦	77.6±2.5 ♦	93.1±1.6	96.9±1.0 ◊	96.3±1.1 ◊
tao	86.4±4.5	96.6±1.2 ◊	96.1±1.3 ◊	84.0±2.7 ♦	84.3±2.7 ■	83.6±2.7 ♦
vehicle	74.3±4.7	69.4±4.0 ♦	70.0±4.4 ♦	74.4±4.1	83.0±3.7 ◊	41.7±3.8 ♦
vote	95.7±3.1	92.3±3.9 ♦	93.0±3.9 ♦	95.9±3.0	94.9±3.3	94.7±3.2
vowel	66.0±6.6	99.1±0.9 ◊	97.1±1.7 ◊	70.5±4.0 ◊	99.2±0.9 ◊	31.0±5.3 ♦
waveform-5000	82.5±1.5	73.5±1.7 ♦	77.6±1.6 ♦	86.4±1.4 ◊	81.5±1.6 ♦	85.4±1.5 ◊
wdbc	96.0±2.5	95.3±2.7	97.1±1.9 □	97.7±1.9 ◊	97.3±2.1 ◊	92.3±2.9 ♦
wine	95.6±4.9	95.2±4.4	96.7±3.9	98.6±2.6 ◊	97.2±3.6	41.3±3.2 ♦
wpbc	74.3±8.9	71.4±9.9	74.1±8.7	76.4±3.7	78.4±8.6	76.3±1.8
zoo	95.1±6.1	97.5±4.7	93.3±6.9	96.4±5.6	95.0±6.6	73.4±9.1 ♦
score 99%		13/7	9/11	9/17	8/13	23/8
score 95%		14/7	9/15	9/18	9/14	24/9

gorithm. The comparison included the decision tree learner C4.5, the rule extraction mechanism PART, the Naive Bayes classifier, as well as the support vector machine algorithm SMO. None of the algorithms clearly outperformed XCS nor was any of the algorithms clearly outperformed by XCS (except majority and 1-R). The comparison with simple majority classification and the simple 1-R algorithm clearly confirmed successful learning in the XCS classifier system.

Additionally, we showed that the facetwise LCS theory established in the previous chapters can be extended to real-valued features, nominal features, as well as a combination of different types of features. In order to extend the theory, it is necessary to adjust the specificity measure to the representation at hand. In a real-valued representation, volume can be equated with generality, which corresponds to one minus specificity. With the redefinition at hand, the other parts of the theory can be adapted fairly easily.

It should be emphasized once more that XCS is not really an algorithm that is designed to do classification. In fact, all the XCS results in this chapter were obtained in a reinforcement learning framework. In contrast to the other learners that all apply statistical analyses on the whole train-dataset, XCS learns a competitive solution online receiving and classifying one training instance at a time. Feedback is provided in terms of reinforcement. Although the provided two reinforcement levels (0 or 1000) clearly distinguish correct from incorrect classifications, the learning mechanism in XCS is designed to solve other types of problems as well including the multistep reinforcement learning problems investigated in the next chapter.

Chapter 9

Reinforcement Learning Problems

The performance investigations of XCS in various classification problems including diverse Boolean function problems as well as datamining problems showed that the facetwise LCS theory can predict XCS's behavior accurately. We confirmed the importance of appropriate selection pressure, population sizing and specificity, as well as the dependence on problem properties such as the minimal order complexity, infrequent niche occurrences, or overlapping solution representations. The datamining applications confirmed XCS's machine learning competitive learning behavior in terms of accuracy and solution generality.

Thus, it is now time to face the last part of our facetwise LCS theory approach: the additional challenges in multistep problems. In particular, (1) effective behavior needs to be assured, (2) problem sampling issues may need to be reconsidered, and (3) reward needs to be propagated most accurately.

As shown in our XCS introduction, XCS learns an approximation of the underlying Q-value function using Q-learning. Thus, to enable learning of the Q-value function, behavior needs to assure that all state transitions are experienced infinitely often in the long run. However, since Q-learning is *policy independent*, the behavioral policy does not need to depend on the current Q-value estimates enabling other behavioral biases. Since learning was shown to be most successful if all problem niches are encountered equally frequently, a behavioral bias that strives to experience all environmental states equally often can be expected to be beneficial. However, in this chapter, we are interested in *whether* XCS can learn optimal behavior and an optimal Q-function approximation rather than if learning can be further optimized by improving behavioral issues.

Most recently, Kovacs and Yang (2004) confirmed that more uniform problem sampling can strongly improve performance in multistep problems. Introducing an additional state-transition memory to XCS that is used to resample previously seen transitions, uniform

state-transition sampling is approximated increasing XCS's learning speed. Several sampling issues are also visible in our experimental investigations in several maze and blocks-world scenarios. In effect, population sizes need to be adjusted to prevent niche loss and ensure reproductive opportunities especially early in the run.

Most importantly, though, this chapter considers the importance of an accurate reward propagation. In essence, we implement gradient-based update techniques in the XCS classifier system derived from results in the function approximation literature in RL (Sutton & Barto, 1998). These mostly neural-based function approximation techniques have been shown to be highly unstable if direct gradient methods are used to implement Q-learning (Baird, 1995). Accordingly, residual gradient methods have been developed to improve robustness (Baird, 1999).

The aim of this chapter is to explore XCS's performance in multistep problems. We investigate the possibility of applying gradient-based update methods in the XCS system improving learning reliability and accuracy. We show how LCSs are related to neural function approximation methods and use similar gradient-based methods to improve stability and robustness. In particular, we apply XCS to large multistep problems using gradient-based temporal difference update methods reaching higher robustness and stability. The system also becomes more independent from learning parameter settings.[1]

The remainder of this chapter first gives an overview over (residual) gradient approaches in the RL-based function approximation literature. Next, we incorporate the function approximation techniques in the XCS classifier system. The subsequent experimental study confirms the expected improved learning stability and reliability in various multistep problems including noisy problems and problems with up to ninety additional randomly-changing irrelevant attributes. Summary and conclusions reconsider the relation of the XCS system to tabular-based RL and to function-approximation-based RL.

9.1 Q-learning and Generalization

Tabular Q-learning is simple and easy to implement but it is infeasible for problems of interest because the size of the Q-table (which is $|S| \times |A|$) grows exponentially in the number of problem features (attributes). This is a major drawback in real applications since the bigger the Q-table the more experiences required to converge to a good estimate of the optimal Q-table (Q^*) and the more memory required to store the table (Sutton & Barto, 1998).

[1]Related publications of parts of this chapter can be found elsewhere (Butz, Goldberg, & Lanzi, 2004b; Butz, Goldberg, & Lanzi, 2004c).

To cope with the complexity of the tabular representation, the agent must be able to *generalize* over its experiences. That is, it needs to learn a good approximation of the optimal Q-table from a limited number of experiences using a limited amount of memory. In RL, generalization is usually implemented by function approximation techniques: The action value function $Q(\cdot,\cdot)$ is seen as a function that maps state-action pairs into real numbers (i.e., the expected payoff); gradient descent techniques are used to build a good approximation of function $Q(\cdot,\cdot)$ from on-line experience.

Already in Chapter 1 we showed the Q-table for Maze1 in Table 1.2. From another perspective, we can order the payoff levels to generate a surface of different payoff levels. In the Maze1 problem, for example, there are 40 state-action pairs in total. Assuming a discount factor of $\gamma = 0.9$, 16 of them correspond to a payoff of 810, 21 correspond to a payoff of 900, and only three state-action pairs correspond to a payoff of 1000 (assuming a discount factor of $\gamma = 0.9$). The goal of function approximation techniques is to develop a good estimate of this payoff surface.

When applying gradient descent to approximate Q-learning, the goal is to minimize the error between the desired value that is estimated by "$r + \gamma \max_{a' \in A} Q(s', a')$" and the current payoff estimate given by $Q(s, a)$ as shown in Equation 1.4. For example, when applying function approximation techniques parameterized by a weight matrix W using gradient descent, each weight w changes by Δw at each time step t,

$$\Delta w = \beta(r + \gamma \max_{a' \in A} Q(s', a') - Q(s, a)) \frac{\partial Q(s, a)}{\partial w} \tag{9.1}$$

where β is the learning rate and γ is the discount factor. Following the notation used in th RL-approximation literature, $Q(\cdot,\cdot)$ represents the approximated action value function not the Q-table (Baird, 1995; Baird, 1999; Sutton & Barto, 1998). This weight update depends both (i) on the difference between the desired payoff value associated with the current state-action pair (that is, $r + \gamma \max_{a' \in A} Q(s', a')$) and the current payoff associated with the state-action pair $Q(s, a)$ and (ii) on the gradient component represented by the partial derivate of the current payoff value with respect to the weight. The gradient term estimates the contribution of the change in w to the payoff estimate $Q(s, a)$ associated with the current state-action pair. Function approximation techniques that update their weights according to the equation above are called *direct algorithms*.

While tabular reinforcement learning methods can be guaranteed to converge, function approximation methods based on direct algorithms, like the previous one, have been shown to be fast but unstable even for problems involving few generalizations (Baird, 1995; Baird,

1999). To improve the convergence of function approximation techniques for reinforcement learning applications, another class of techniques, namely *residual gradient algorithms*, have been developed (Baird, 1999). Residual gradient algorithms are slower but more stable than direct ones and, most importantly, they can be guaranteed to converge under adequate assumptions. Residual algorithms extend direct gradient descent approaches, which focuses on the minimization of the difference "$(r + \gamma \max_{a' \in A} Q(s', a') - Q(s, a))$," by adjusting the gradient of the current state with an estimate of the effect of the weight change on the successor state. The weight update Δw for Q-learning implemented with a *residual gradient* approach becomes the following:

$$\Delta w = \beta(r + \gamma \max_{a' \in A} Q(s', a') - Q(s, a)) \left[\frac{\partial Q(s, a)}{\partial w} - \gamma \frac{\partial}{\partial w} \left(\max_{a' \in A} Q(s', a') \right) \right], \tag{9.2}$$

where the partial derivative $\frac{\partial}{\partial w}(\max_{a' \in A} Q(s', a')$ estimates the effect that the current modifications of the weight have on the value of the next state. Note that since this adjustment involves the next state, the discount factor γ must also be taken into account.

Direct approaches and *residual gradient* approaches are combined in *residual algorithms* by using a linear combination of the contributions given by the former approaches to provide a more robust weight update formulation. Let ΔW_d be the update matrix computed by a direct approach, and ΔW_{rg} be the update matrix computed by a residual gradient approach, then, the updated matrix for residual approach ΔW_r is computed as

$$\Delta W_r = (1 - \phi)\Delta W_d + \phi \Delta W_{rg}.$$

By substituting the actual expressions of ΔW_d and ΔW_{rg} for Q-learning with *direct* and *residual gradient* approach (see Baird, 1999 for details) we obtain the following weight update for Q-learning approximated with a *residual approach*:

$$\Delta w = \beta(r + \gamma \max_{a' \in A} Q(s', a') - Q(s, a)) \left[\frac{\partial Q(s, a)}{\partial w} - \phi\gamma \frac{\partial}{\partial w} \left(\max_{a' \in A} Q(s', a') \right) \right]. \tag{9.3}$$

Note that the residual version of Q-learning turns out to be basically an extension of *residual gradient* approach where the contribution of the next state ($\frac{\partial}{\partial w} \max_{a' \in A} Q(s', a')$) is weighted by the parameter ϕ. The weighting can be either adaptive, or it can be computed from the weight matrices W_d and W_{rg} (Baird, 1999).

With the reinforcement learning knowledge in mind, we now turn to the XCS classifier system investigating how XCS approximates the Q-function and how gradient-based updates

can be incorporated into XCS.

9.2 Ensuring Accurate Reward Propagation: XCS with Gradient Descent

As shown throughout this thesis, XCS uses Q-learning techniques but can also be compared to a function approximation mechanism. In this section, we analyze the similarities between, tabular Q-learning, Q-learning with gradient descent, and XCS. We show how to fuse the two capabilities adding gradient descent to XCS's parameter estimation mechanism.

9.2.1 Q-Value Estimations and Update Mechanisms

As explained above, while tabular Q-learning iteratively approximates Q-table estimates, a function approximation approach estimates the Q-table entries by the means of a weight matrix. Using the direct (gradient descent) approach, each weight w in the matrix W is modified by the quantity Δw determined by Equation 9.1. Hereby, the gradient component $\frac{\partial Q(s,a)}{\partial w}$ is used to guide the weight update.

As seen above, XCS exploits a modification of Q-learning applying direct and independent prediction updates to all classifiers in the to-be updated action set $[A]$. To see the relation directly, we restate the update rule here again for the multistep case (see also Equation 3.4).

$$R \leftarrow R + \beta(r + \gamma \max_{a \in [A]} P(a) - R). \tag{9.4}$$

We can note that while tabular Q-learning only updates one value at each learning iteration (updating $Q(s, a)$), XCS updates all classifiers in the action set $[A]$. In fact, each position in the Q-table is represented by the corresponding prediction array value (Equation 3.1). Comparing the weight update for gradient descent (Equation 9.1) and the update for classifier predictions (Equation 9.4) we note that in the latter no term plays the role of the gradient. Classifier prediction update for XCS was directly inspired by *tabular* Q-learning (Wilson, 1995) disregarding any gradient component.

9.2.2 Adding Gradient Descent to XCS

To improve the learning capabilities of XCS we add gradient descent to the equation for the classifier prediction update in XCS. As noted above, the value of a specific state-action pair is

represented by the system prediction $P(\cdot)$, which is computed as a fitness weighted average of classifier predictions (Equation 3.1). In general, learning classifier systems consider the rules that are active and combine their predictions (their *strength*) to obtain an overall estimate of the reward that should be expected. In this perspective, the classifier predictions play the role of the weights in function approximation approaches. The gradient component for a particular classifier cl_k in the to-be-updated action set $[A]$ can be estimated by computing the partial derivate of $Q(s,a)$ with respect to the prediction p_k of classifier cl_k:

$$\frac{\partial Q(s,a)}{\partial w} = \frac{\partial}{\partial R_k}\left[\frac{\sum_{cl_j\in[A]} R_j F_j}{\sum_{cl_j\in[A]} F_j}\right] =$$

$$= \frac{1}{\sum_{cl_j\in[A]} F_j}\frac{\partial}{\partial R_k}\left[\sum_{cl_j\in[A]} R_j F_j\right] = \frac{F_k}{\sum_{cl_j\in[A]} F_j}. \tag{9.5}$$

Thus, for each classifier the gradient descent component corresponds to its relative contribution (measured by its current relative fitness) to the overall prediction estimate.

To include the gradient component in XCS's classifier prediction update mechanism, prediction R_k of each classifier $cl_k \in [A]$ is now updated using

$$R_k \leftarrow R_k + \beta(r + \gamma \max_{a\in A} P(a) - R_k)\frac{F_k}{\sum_{cl_j\in[A]} F_j}. \tag{9.6}$$

All other classifier parameters are updated as usual (see Chapter 3).

Due to the contribution-weighted gradient-based update, the estimate of the payoff surface, that is, the approximation of the optimal action-value function Q^*, becomes more reliable. As a side effect, the evolutionary component of XCS can work more effectively since the classifier parameter estimates are more accurate. The next section validates this supposition.

9.3 Experimental Validation

We now compare XCS's performance without and with gradient descent on several typically used maze problems as well as a blocks world problem. The results confirm that the incorporation of the gradient-based update mechanism in XCS allows the system to reliably learn a large variety of hard maze problems. In addition, we show that XCS is able to ignore irrelevant attributes requiring only a linear increase in population size. Finally, we apply XCS to the maze problems with additional noise in the actions.

The results confirm that XCS is a reliable reinforcement learning algorithm that is able to propagate gradient-based reinforcement backward and evolve a very general, accurate representation of the underlying MDP problem.

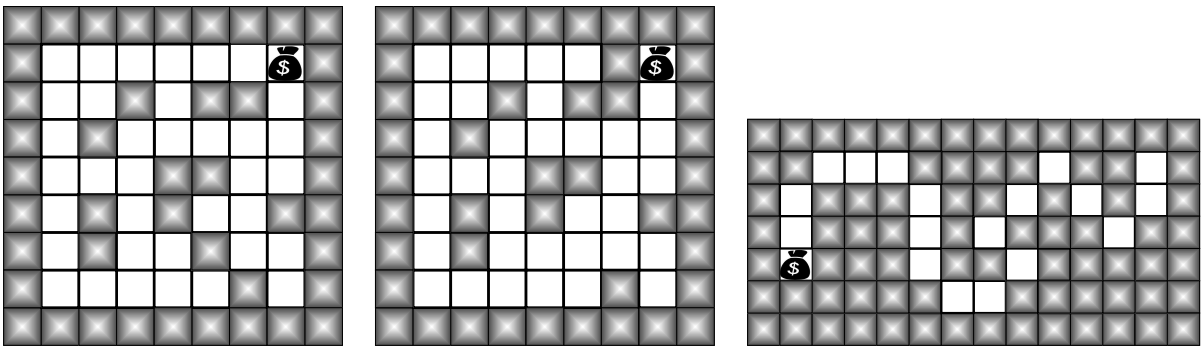
Parameters were set slightly different from the ones in the previous chapter slightly increasing the population size and decreasing the don't care probability to prevent niche loss and ensure the capability of handling large problem spaces, respectively. If not stated differently parameters were set to $N = 3000$, $\beta = 0.2$, $\alpha = 1$, $\varepsilon_0 = 1$, $\nu = 5$, $\theta_{GA} = 25$, $\chi = 1.0$, $\mu = 0.01$, $\gamma = 0.9$, $\theta_{del} = 20$, $\delta = 0.1$, $\theta_{sub} = 20$, and $P_{\#} = 0.8$ throughout the experiments. Uniform crossover was applied. Tournament selection is applied throughout this section with a tournament size proportion of $\tau = 0.4$ of the action set size. Results are averaged over fifty experiments in the usual case but over twenty experiments in the case of additional sixty or ninety random attributes.

The experiments alternated between learning (exploration) and testing (exploitation) trials. During a learning trial, actions were executed at random whereas during testing, the best action was chosen deterministically according to the prediction array values. Additionally, during testing the evolutionary algorithm was not triggered but reinforcement updates were applied (as used in the literature (Wilson, 1995; Lanzi, 1999a)). If a trial did not reach the goal position after fifty steps, the trial is counted as a fifty step trial and the next trial starts. Note that Lanzi (1999a) has shown that such a reset method can improve learning performance since it assures that the search space is explored more equally. For the sake of comparison between gradient and non-gradient-based updates in XCS, however, the advantage applies to both systems and is therefore irrelevant. Additionally, we assure that a trial undergoes a maximum number of fifty learning steps so that the actual number of learning steps is bound by the fifty steps.

9.3.1 Multistep Maze Problems

We apply XCS to three different maze problems previously studied with the XCS classifier system (Lanzi, 1999a) and other LCS systems (Butz, 2002). The investigated Maze problems are shown in Figure 9.1. Hereby, symbol `T` stands for an obstacle (or a tree) and symbol `F` (for food) stands for a position that triggers reward. Maze5 and Maze6 are very similar in nature. However, the Maze6 environment poses additional hard challenges since it is much less likely to reach the food position during a random walk. Woods14 is challenging due to the long backpropagation chain—in total eighteen steps that need to be back-propagated.

The maze problems are coded similar to the Maze1 and Maze2 problems, introduced in



(a) (b) (c)

Figure 9.1: Maze5 (a), Maze6 (b), and Woods14 (c) all pose different challenges to the XCS learning algorithms.

Chapter 1. A state in the environment is perceived as a binary feature vector of length sixteen coding each of the eight neighboring positions by two bits starting north and coding clockwise. An obstacle is coded by `01`, food is coded by `11`, and finally an empty position is coded by `00`. Code `10` has no meaning. A payoff of 1000 is provided once the food position is reached and the agent is reset to a randomly selected empty position in the maze.

Figure 9.2 compares performance in Maze5 (a,b) and Maze6 (c,d) without and with the gradient-based update technique. Clearly, XCSTS with gradient-based update learning is able to learn the optimal path to the food from all positions in the mazes whereas XCSTS without gradient-based updates fails to learn an optimal path and stays basically at random performance. Investigations in the evolved payoff landscape showed that XCS with gradient strongly tends to evolve an overgeneral representation that result in an inaccurate high-variance reward prediction. XCSTS with gradient, on the other hand, evolves the underlying payoff landscape accurately. Also with ten additional random attributes, which are set uniformly randomly at each step, XCSTS without gradient does not reach any better performance. XCSTS with gradient again reaches optimal performance although slightly delayed due to the small population size of $N = 3000$.

Increasing the population sizes, XCS is even able to learn an optimal policy if 30 ($N = 5000$), 60 (N=7500), or 90 (N=10,000) bits are added to the perception (Figure 9.3a,c). These bits are changing at random and are consequently highly disruptive in the learning process. Nonetheless, XCS is able to identify the relevant bits fast and ignore the irrelevant attributes. The monitored population sizes (Figure 9.3a,c) reach a very high value initially but then quickly degrade to a low level. Hereby, as expected, the minimum population size is not reached since mutation continuously introduces specializations in the irrelevant bits and

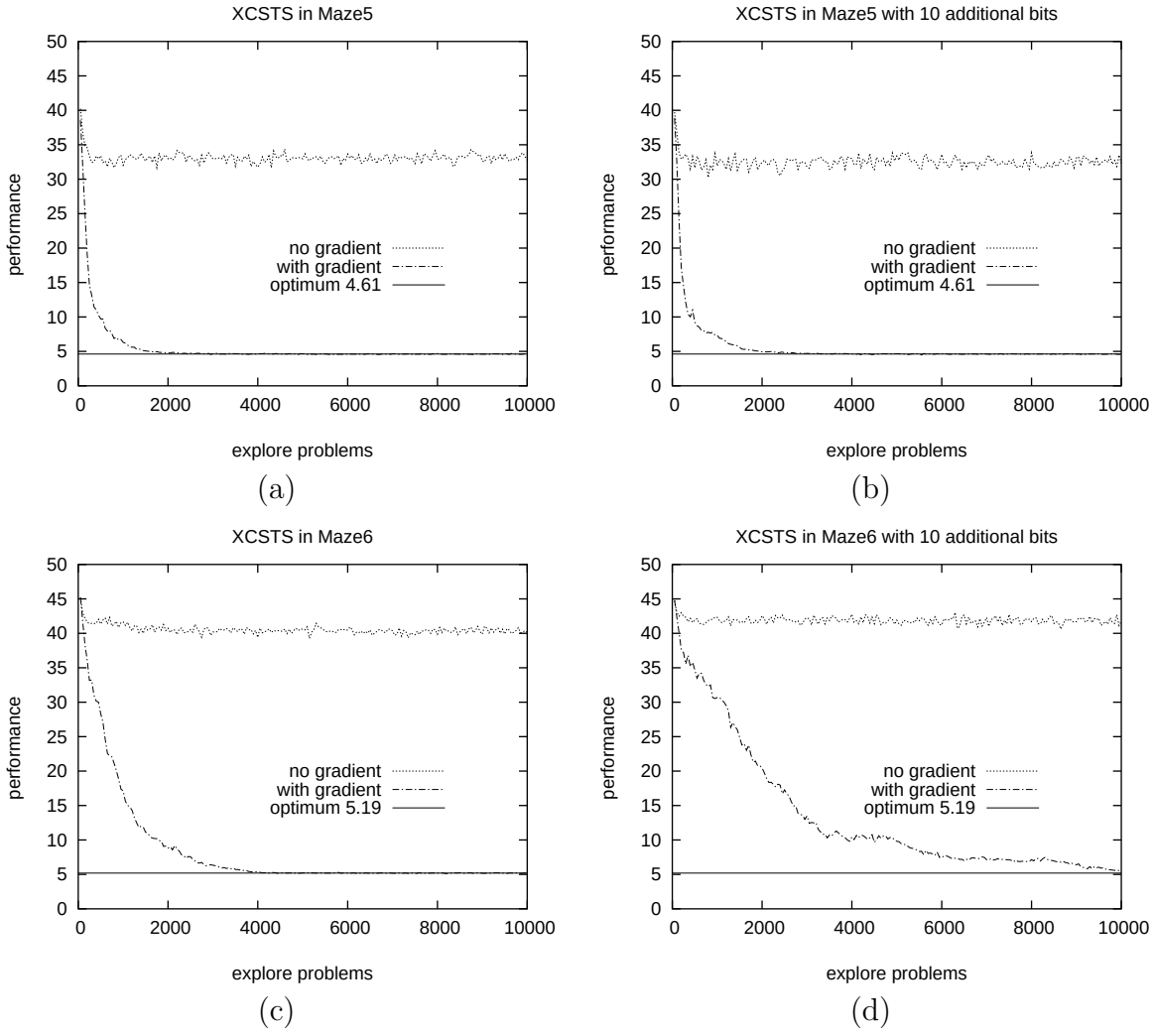


Figure 9.2: Performance comparisons in Maze5 and Maze6 confirm the superiority of the gradient approach.

thus additional classifiers in the population. Subsumption can decrease the population size only slowly since the reward propagation continuous to be noisy so that classifier accuracy ($\epsilon < \epsilon_0$) is hard to reach.

Similar observations can be made in Woods14 as shown in Figure 9.4. In Woods14 it seems particularly important to have a larger population size available. Due to the long chain of necessary back-propagations and the infrequent occurrence of distant states in the environment, problem sampling issues need to be reconsidered. In particular, due to the infrequent sampling of problem subspaces, niche support as well as reproductive opportunities can only be assured if the population is large enough. Due to the disruption when the population size is set too small, learning the optimal behavioral policy is delayed (Figure 9.4

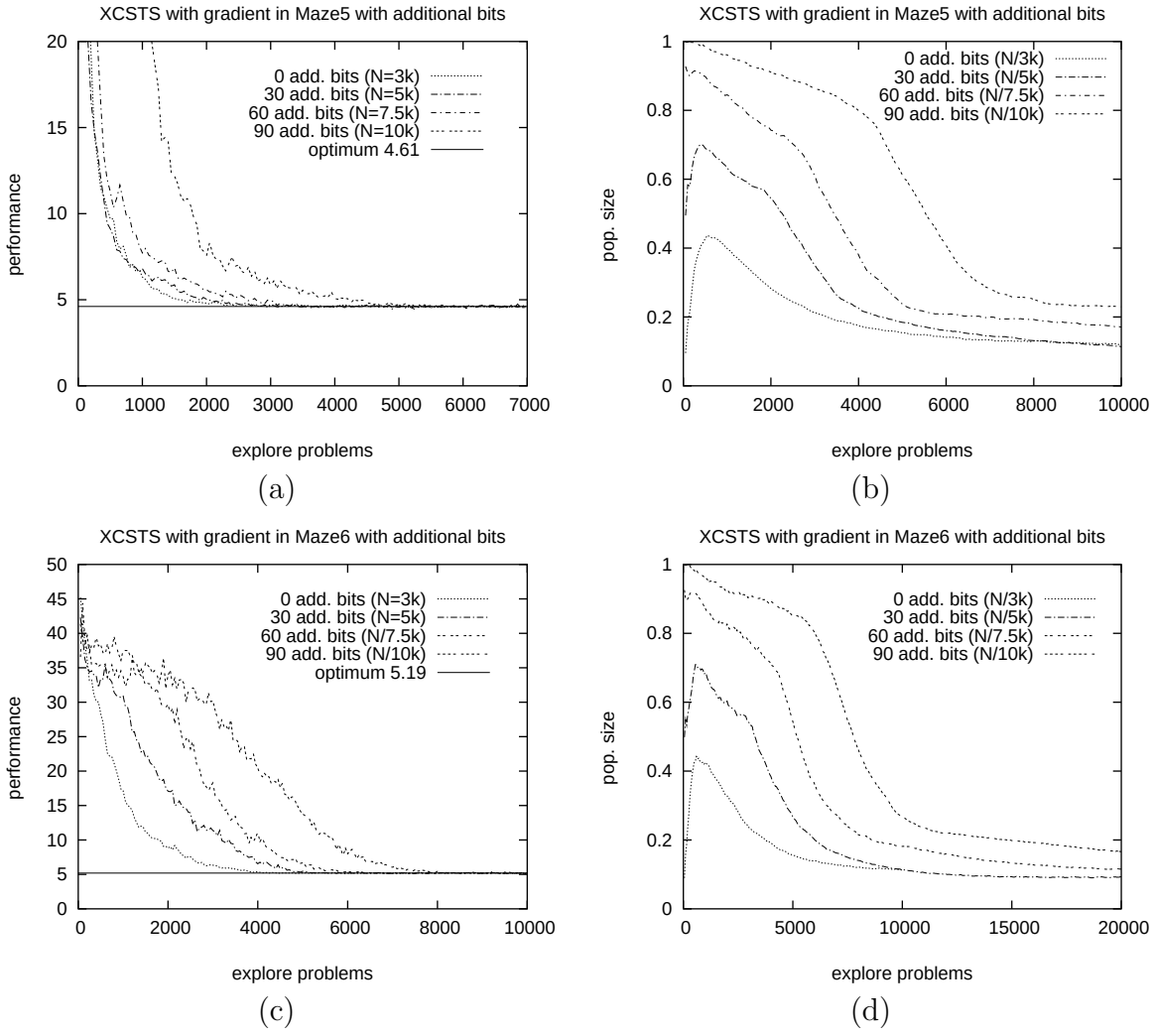


Figure 9.3: XCSTS with the gradient approach is able to learn an optimal policy even when adding 90 randomly changing bits. The population sizes show that although the additional random attributes result in an initial very large population size, population size drops off quickly indicating the evolving focus on relevant attributes.

a,b) in contrast to the sufficient size of $N = 5000$ (Figure 9.4 c).

This observation is also confirmed when analyzing the single Woods14 runs: Out of the fifty runs, four had not converged after 5000 exploration problems with a population size of $N = 3000$. Due to the skewed problem sampling, reproductive opportunities cannot be assured when the population size is set too low disrupting learning in some runs. The higher population size assures reproductive opportunities and thus assures fast learning in all runs. Thus, as expected by the facetwise LCS theory, extra care needs to be taken in multistep problems due to expectable highly skewed problem sampling.

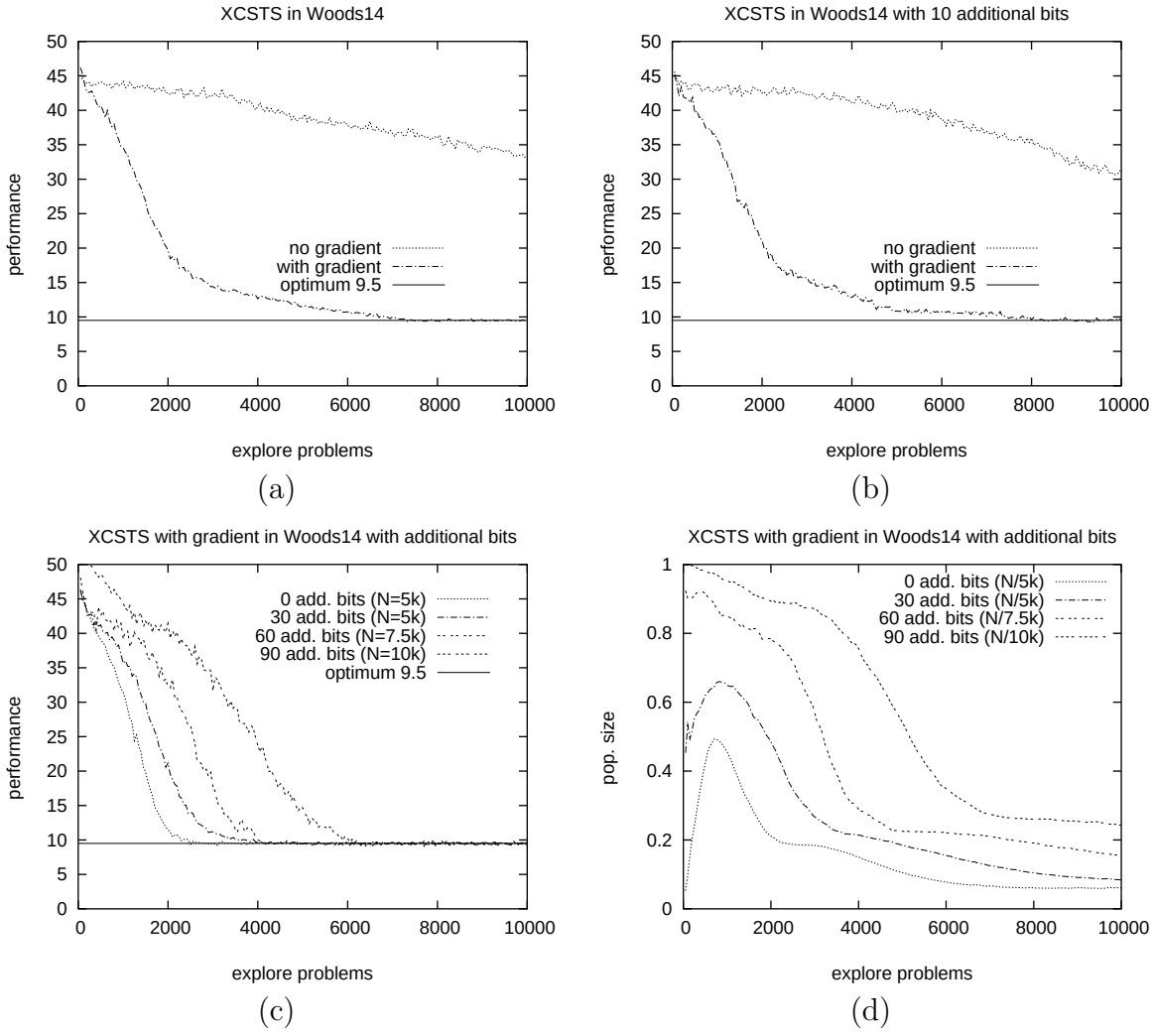


Figure 9.4: Also in Woods14, the gradient-based update approach is highly beneficial. Also a higher population size yields better performance (a vs. c). Additional random attributes show the resulting strong learning potential of the XCSTS system with gradient-based updates.

Besides the performance curves, we are also interested in the actual approximation of the underlying Q-value function. According to the theory above, XCS should learn an exact approximation of the Q-value function. Figure 9.5 shows the maximum Q-value estimate in each state of XCSTS (a,c) and XCSTS (b,d) with gradient in Maze5 (a,b) and Woods14 (c,d) including the standard deviation of the estimations over fifty runs after 10000 learning trials. Clearly, XCSTS with gradient is able to evolve an accurate representation of the Q-function. XCSTS without gradient, though, does not evolve accurate representations but stays overgeneral. In Woods14, the matter appears slightly better as also indicated by the

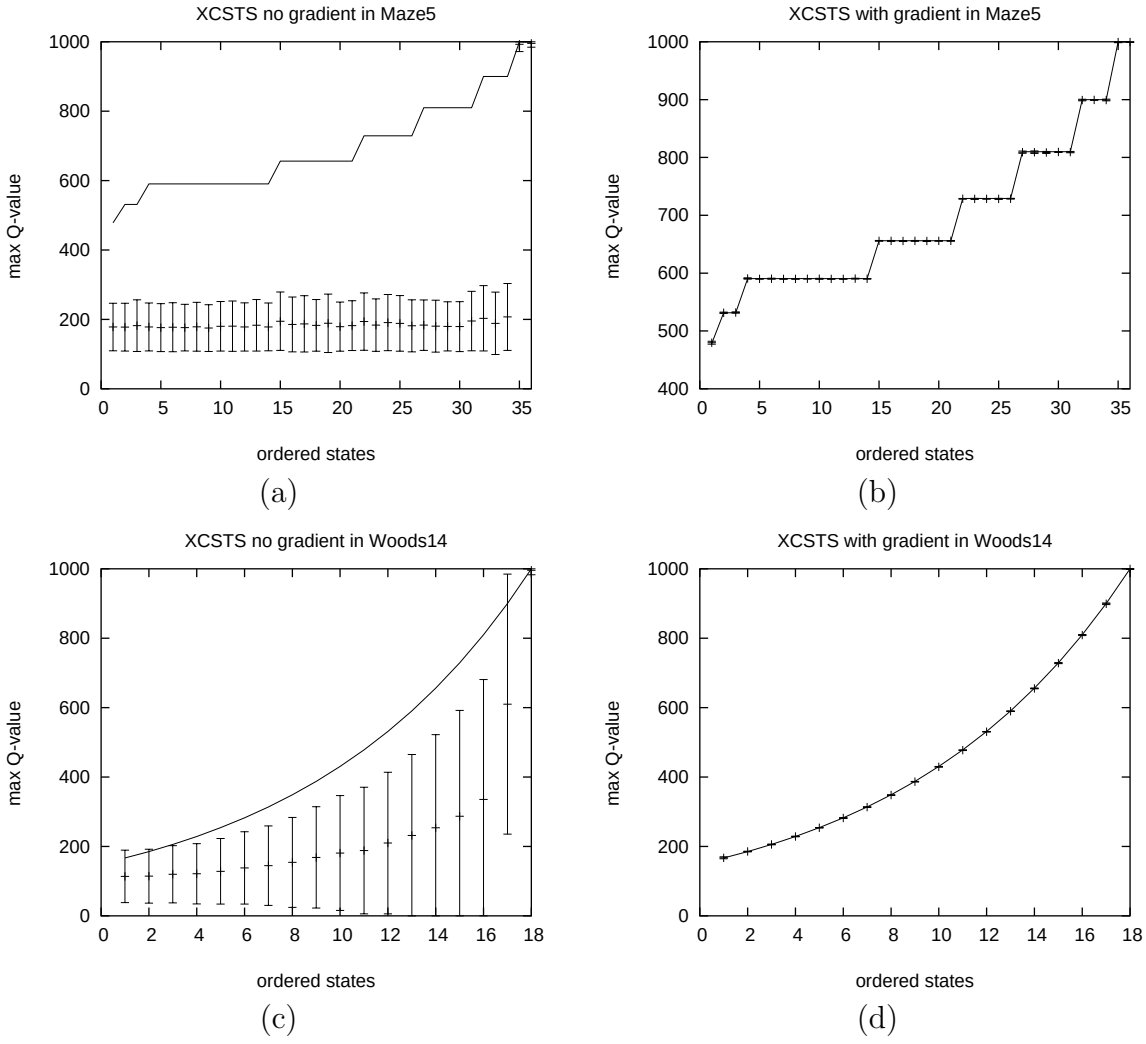


Figure 9.5: The actual Q-value estimates confirm that XCS with gradient reliably evolves accurate Q-value approximations.

performance curves in Figure 9.4b. However, the standard deviation is very high indicating high noise in the estimation values.

Finally, we investigated if XCSTS with gradient is also able to handle noisy problems. Consequently, we added noise to the actions of the system so that actions do not necessarily lead to the intended position but with a low probability, the actions leads to one of the neighboring positions. That is, with low probability an action to the north may lead to the position in the north-west or north-east. In our experiments, we set the probability of an action slip to 0.2. Results are shown in Figure 9.6 comparing performance in the case of a slippery surface to that in the deterministic problem. The results show that XCSTS with gradient is able to learn an accurate reward distribution in a noisy environment allowing it to

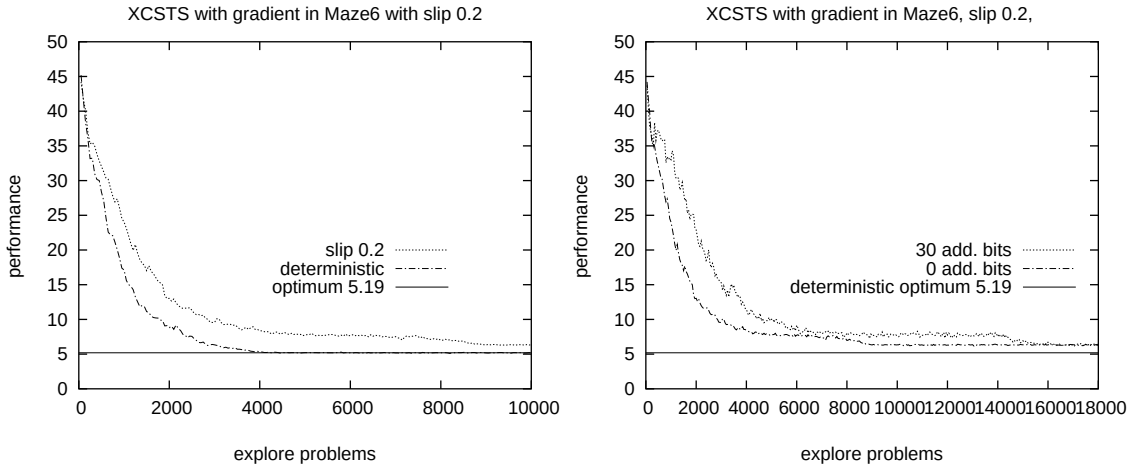


Figure 9.6: Adding non-determinism to the actions delays the evolution of an optimal performance. Nonetheless, even with thirty additional random bits (right-hand side) XCS converges to a stable, optimal policy.

evolve a stable, near-optimal performance. If additionally adding thirty randomly changing attributes, convergence is delayed further but after about 15000 learning trials also the last run of the twenty runs converges to the performance level of the one without additional random attributes.

9.3.2 Blocks-World Problems

Blocks worlds problems were investigated intensively over the last decades. Whitehead and Ballard (1991) studied temporal difference learning techniques in blocks worlds. The field of relational reinforcement learning studies logic-based relational representations. Kersting, Van Otterlo, and De Raedt (2004) show how to propagate reinforcement backward beginning from a formalized goal using logical inferences.

Our blocks world differs in that we code the blocks world perceptually instead of coding the explicit relations between the blocks. In fact, the relations need to be inferred from the online interaction with the blocks. This setup makes the problem more similar to an actual real-world scenario.

The blocks world is perceived from a global perspective observing the distribution of blocks over the available stacks in the problem. Actions manipulate the blocks by gripping or releasing a block on a specific stack. Figure 9.7 shows several exemplar states and the corresponding perceptions in a blocks world with s_{BW}=3 stacks and $b_{BW} = 4$ blocks. The perception consists of b_{BW} positions for each stack coding the presence of a block by 1 and

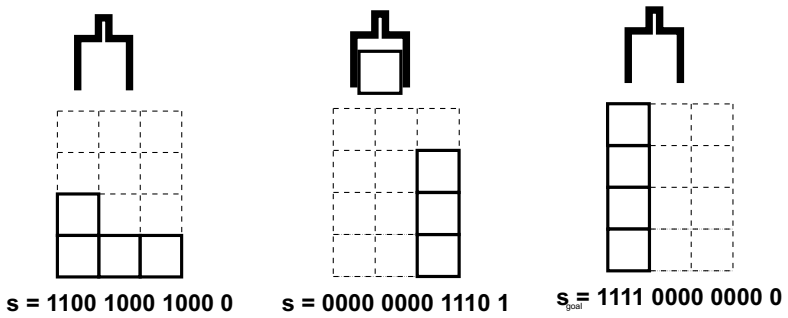


Figure 9.7: The three exemplar states in the blocks world with s_{BW}=3 stacks and $b_{BW} = 4$ blocks illustrate the blocks world problem. The goal is defined as transporting a certain number of blocks onto the first stack.

the absence by 0. An additional bit codes if the gripper currently holds a block. Actions are possible of gripping or releasing a block on a specific stack. Thus, there are $2s_{BW}$ actions available in this blocks world. The goal is defined by transporting a certain number of blocks onto the first stack. For example, in the shown blocks world in Figure 9.7, the goal is to move all four blocks onto the first stack (right-hand side figure). Similar to the maze scenario, XCS encounters a blocks world scenario receiving reward once the goal state is reached triggering the start of the next trial.

Results in blocks world scenario are depicted in Figure 9.8. The figure on the left confirms the superiority of the gradient-based update once more. The goal to put three blocks onto the first stack is still rather easy to accomplish and also the setting without gradient evolves an optimal policy, albeit slower. Due to the low required height of the target stack, reaching the goal by chance occurs frequently, which triggers the provision of external reinforcement frequently, which again provides a much stronger reinforcement signal. Increasing the required block height of the goal stack by one makes it harder to evolve an optimal policy for XCS without gradient. Reaching the goal stack becomes infrequent, which makes it hard for XCS to learn a proper reward distribution. With the gradient-based update technique, the propagation of reward becomes much more stable and XCS is still able to learn the optimal policy.

Note that prioritized exploration algorithms might improve performance such as the Dyna-Q+ algorithm (Sutton, 1991), which prioritizes updates according to the delay since the most recent update, or the prioritized sweeping approach (Moore & Atkeson, 1993), which prioritizes exploration as well as internal RL updates according to the amount of change caused by the most resent update. The point of the gradient-based update technique, though, is to make XCS learning more robust. Prioritized exploration could be incorporated

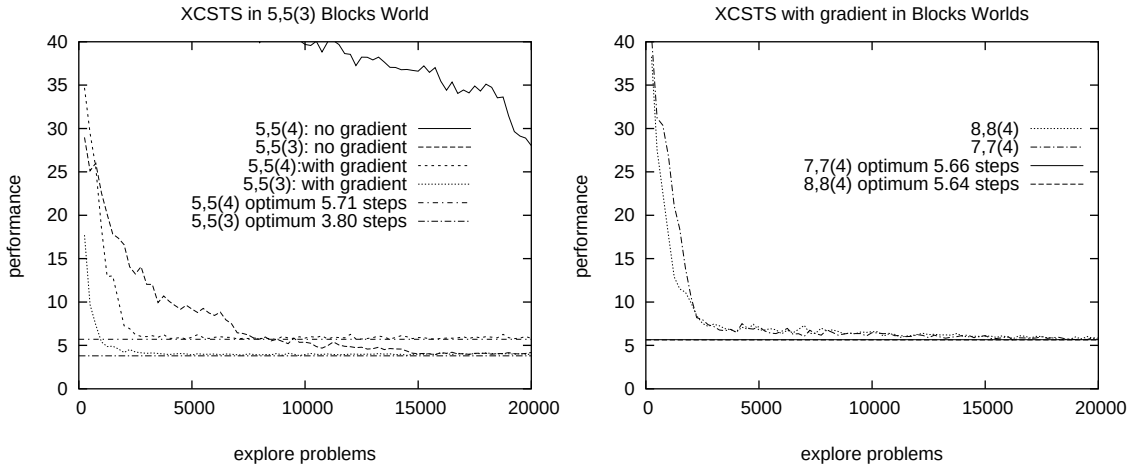


Figure 9.8: Also in the blocks world problem, the gradient update techniques speeds-up and stabilizes learning of an optimal performance. Population sizes were set to $N = 5,000$ for the $s_{BW} = 7$, $b_{BW} = 7$ and to $N = 6000$ for the $s_{BW} = 8.b_{BW} = 8$ problem.

additionally to make learning even faster.

Figure 9.8 (right-hand side) shows performance of XCS with gradient in larger blocks world problems. XCS can solve problems as large as the 8x8 blocks world problem that codes a state by $l = 65$ bits and comprises 9876 distinct states and 16 possible actions. Thus, XCS is able to filter the input bits for relevancies with respect to the current problem and goal at hand efficiently evolving a near-optimal behavioral policy. It appears that XCS does not exactly reach optimality, however, which is especially visible in the smaller problems. The explanation is the states above the goal state, that is, when more than the actual required blocks are situated on the goal stack initially. This case occurs very infrequently so that the sampling of this subproblem space is very infrequent. Thus, a stable representation of the exceptional cases cannot evolve as predicted by the reproductive-opportunity bound and our facetwise LCS theory.. Thus, if this scenario occurs, XCS is not able to act optimally, which accounts for the slight in-optimality in the overall average performance.

9.4 Summary and Conclusions

This chapter has shown that XCS with tournament selection and gradient descent is able to solve a variety of multistep problems reliably and optimally. XCS showed to be noise robust as well as very powerful in ignoring additional irrelevant randomly changing attributes. Hereby, the population size needs to grow linearly in the additional number of features as

predicted by our XCS theory.

Thus, we confirmed that the basic learning results from our facetwise theory approach carry over to multistep problems. However, multistep problems also showed to pose additional learning challenges as suggested in the facetwise LCS theory approach. Essentially, effective adaptive behavior needs to be ensured. Moreover, population sizes may need to be increased due to potentially skewed problem sampling. Most importantly, though, reward needs to be propagated and approximated most accurately.

The addition of the gradient-based reward-prediction update technique showed that the XCS learning architecture lies somewhat in between a tabular-based Q-learner and a neural-network based Q-learner. In tabular-based RL and in particular in Q-learning each Q-value is represented by *one* entry in the Q-table. In neural-based reinforcement learning each Q-value is approximated by the activity in the *whole* neural network. XCS lies in between because it estimates Q-values by a *subset* of classifiers. The subsets distinguish different reward levels which explains why the residual gradient addition appears not necessary in XCS: The residual gradient distinguishes different, subsequent reward levels. In XCS, this distinction is evolved by the GA component.

In essence, XCS approaches function approximation by detecting subspaces whose function values can be effectively approximated by the chosen approximation mechanism. In the simplest case, the approximation is a constant as in the work herein but other methods, such as linear approximation, are under investigation (Wilson, 2001a). XCS's learning mechanism evolves such a distributed representation interactively. While the genetic component evolves problem space partitions that allow an accurate value estimation in each of the evolved subspaces, the RL-based approximation method estimates the value in each partition and rates the relative quality of each partition in terms of the accuracy of the value prediction. The gradient approach enabled a more reliable reward back-propagation and thus more reliable learning of an optimal behavioral policy.

In conclusion, this chapter showed that XCS is a valuable alternative to neural network based function approximation techniques. Especially when only few problem features (that is, sensory inputs) are relevant for the task at hand, XCS's computational requirements increase only linearly in the number of irrelevant features hardly affecting learning speed. Thus, XCS serves well in RL-based problems with many irrelevant and redundant problem features detecting and propagating the features relevant for accurate predictions reliably.

Chapter 10

From Facetwise LCS Theory Towards Competent Cognitive Systems

The XCS classifier system is only one among many evolutionary rule-based learning systems. With the successful facetwise analysis approach carried through in XCS, it needs to be considered how the theory may carry over to (1) analyze other LCSs and related systems in the same way, (2) apply the knowledge to create new LCS systems, targeted to a specific problem at hand.

This chapter first explores the functioning of all LCS learning facets in turn, outlining the differences between operators such as strength-based vs. accuracy-based fitness or tournament selection vs. proportionate selection. We especially investigate for which problems and objectives which mechanism is most appropriate. Along the facetwise analysis approach we put forward alternatives in and highlight similarities with other LCS mechanisms.

After the LCS analysis from this problem-oriented perspective, we outline how our knowledge can be carried over into system design. In particular, we propose the integration of LCS-like search mechanisms in cognitive structures for structural growth and distributed relevancy identification. Additionally, we show that the systems may be integrated in multi-layered, hierarchical learning structures influencing solution growth interdependently using only RL mechanisms and evolutionary problem solution structuring.

10.1 General Facetwise LCS Analysis

Besides the gained advances in theory and understanding due to our facetwise LCS approach, the approach also serves well for system analysis. We saw that different learning mechanisms cause different learning biases in the XCS classifier system. Now, we relate these biases to other LCSs and other LCS mechanisms that may yield additional or alternative learning

biases in the light of our facetwise analysis approach.

10.1.1 Which Fitness for Which Solution?

Already in the introduction to simple LCSs we mentioned the importance of the correct fitness approach. Strength-based systems suffer from the problem of strong overgenerals (Kovacs, 2000). Although accuracy-based systems can be misled by unequally distributed variances in payoff, our experimental evaluations in Chapter 7 showed that XCS is very robust with respect to noise. Calculations in Kovacs (2003) showed that different noise values can result in strong overgeneral classifiers also in XCS. However, the resulting overgeneral classifier can be over-ruled only in very extreme noise distribution settings proving XCS's robustness.

Nonetheless, accuracy-based fitness approaches might have other drawbacks with respect to the task at hand. Essentially, XCS's mechanism is designed to learn a reward map over the complete problem space as accurate as possible. Thus, independent of the amount of reinforcement received, XCS strives to learn accurate reward predictions for all rewards. This is desirable when a complete reward map should be learned as necessary for example in RL problems, in which XCS learns an approximation of the Q-value function.

In other problem settings, reward might indicate the importance of a classifier representation rather than the correctness of applying that representation. In such a scenario, a different fitness approach seems more appropriate. In other words, if reward indicates the desired distribution of the problem representation, the learner should not learn an accurate representation of the distribution but rather it should represent the distribution itself in its classifiers. In such a scenario, more reproductive events and learning effort should occur in areas in which higher reinforcement is received.

Incidentally, the ZCS classifier system (Wilson, 1994) is essentially taking this road. ZCS is a strength-based system that applies fitness sharing to overcome the problem of strong overgenerals as shown in Bull and Hurst (2002). Additionally, ZCS applies the GA globally using proportionate selection based on fitness and deletion based on the inverse of fitness. Thus, ZCS reproduces classifiers with higher fitness evolving a population of classifiers that converges to identical fitness values. Given an average fitness vale of $\overline{f}$ in the population, a problem niche in which a reward of r is encountered will be populated by $r/\overline{f}$ classifiers on average. Thus, ZCS should work very well in problems in which reward actually indicates representational importance. The recent successful application in the 37-multiplexer function confirms the potential of the system (Bull, 2003).

Thus, when comparing accuracy-based fitness with shared, strength-based fitness, it is clear that neither is always appropriate. If the task is to learn a complete and accurate reward map representation (including zero rewards), then accuracy-based fitness seems to be the right choice. If the task is to learn the highest reward cases accurately and the lower reward cases progressively less accurately, then the ZCS-framework should be more effective.

10.1.2 Parameter Estimation

In most of this work we assumed that the parameter estimates accurately reflect the expected parameter values in the long run. We additionally showed that in very noisy problems a lower learning rate β is necessary to assure dependable reward and reward prediction error estimates. In Chapter 9 we additionally showed the relation of the XCS parameter estimates to function-approximation techniques in RL. All these relations suggest that β should be a rather small value ($\beta \leq 0.2$).

Somewhat surprisingly, Bull and Hurst (2002) showed that in the ZCS system larger β values might actually be advantageous. Both approaches are correct, though, and strongly depend on the dynamics in the population and the value the parameter intends to estimate.

In XCS, reward prediction and reward prediction error are estimates that are determined *independent* of the rest of the classifier population. They are designed to approximate the expected average reward prediction and reward prediction error value as accurately as possible. Thus, the lower the learning rate β the more accurate the final estimate, especially if the problem is noisy. The higher the learning rate β, though, the faster the initial estimate will be meaningful. This explains why it is very useful to apply the moyenne adaptive modifiée technique (Venturini, 1994) that applies initial large updates (effectively averaging the so-far encountered experiences) and then converges to the small β-based updates.

Fitness, on the other hand, reflects the relative accuracy of a classifier and thus does not apply the moyenne adaptive modifiée technique but approximates its fitness value controlled by learning rate β from the beginning. Since an appropriate relative fitness estimation relies on an accurate error estimation, initial sufficiently low fitness updates are mandatory to prevent disruptive effects due to potential fitness over-estimation. Once a classifier is sufficiently experienced, though, it might be interesting to experiment with larger learning rates for the fitness measure to be able to adapt more rapidly to the current relative accuracy.

The successful application of very high learning rates for β (up to one) in ZCS that improved system performance in several problems (Bull & Hurst, 2002) can only be explained by an advantageous rapid adaptation to current population dynamics. Since classifier fitness

in ZCS depends only on the shared reward in a problem, the reward share directly reflects the actual coverage of the niche in question by the current classifier population. The coverage is dependent on the niche importance, which should be reflected by the received reward, as discussed in Section 10.1. It still remains to be investigated, though, in which problems a high learning rate β in ZCS might actually cause misleading estimates or niche loss especially in problems in which overlapping problem subsolutions evolve.

Nonetheless, a similarly high learning rate may actually be useful for updating the action set size estimate in XCS. Since the estimate solely depends on the population dynamics, faster adjustments should lead to a more appropriate representation of the current population dynamics and should thus be advantageous. However, in overlapping problem domains additional niching care needs to be applied.

As an alternative to varying learning rate β, Goldberg's variance-sensitive bidding approach for LCSs may be used (Goldberg, 1990). It is suggested that classifiers with high estimation variance should add additional variance to their reward prediction or fitness to prevent overestimations due to young or inaccurate classifiers. In this way, promising young classifiers may still get a chance to be considered for reproduction making the evolutionary learning progress somewhat more noisy but stabilizing it once all classifiers converge to accurate values.

10.1.3 Generalization Mechanisms

Besides the search for a complete and accurate problem solution, it is important to evolve a maximally general solution. Following Occam's Razor, the evolutionary learning method needs to be biased towards favoring more general solutions. In XCS, we saw that there is an implicit generalization pressure, that is, the set pressure, which propagates *semantically more general* rules. There is also an explicit generality pressure, that is, subsumption pressure, which favors *syntactically more general* rules. Both pressures favor more general solutions as long as accuracy is maintained.

Semantic generality refers to rule generality with respect to rule applicability, that is, the more often a rule applies in a given data set, the more general the rule is. Syntactic generality refers to generality with respect to the syntax of the rule, that is, the larger the size of the subspace the rule covers, the more general it is. We now discuss several methods to evolve semantically general and syntactically general rules in an evolutionary learning system.

Semantic generalization can be realized, as done in XCS, by applying reproduction in

match sets or action sets and deletion in the whole population. However, this approach may be misleading since the resulting reproduction bias may cause an over-representation of unimportant problem subspaces as discussed above.

A similar approach for semantic generalization was proposed recently in the ZCS system (Bull, 2003). When taxing the generation of offspring in ZCS decreasing fitness, both parent and child have a very low reproduction probability (dependent on the amount of tax). The low probability of reproduction persists until the next time they take part in an action set, updating fitness and consequently restoring parts of the actual fitness dependent on the learning rate.

Interestingly, the consequential generalization pressure is very similar to the one in XCS. As in XCS, the time until the next reproductive event depends on the frequency of niche occurrence. On the other hand, the taxed fitness makes the offspring more likely to be deleted, which might cause a drawback in the ZCS system in some scenarios with lots of problem niches. In effect, it might actually be interesting to think about introducing two related fitness parameters to ZCS: one for reproduction and one for deletion. The reproductive fitness measure should be taxed upon reproduction whereas the fitness measure relevant for deletion should immediately reflect the actual fitness in the current set. Note that in XCS the initial fitness decrease in an offspring classifier has a somewhat similar effect preventing the premature reproduction of under-evaluated classifiers.

The taxation of offspring generation in ZCS as well as the set pressure in XCS cause an implicit semantic generalization pressure rather than an explicit pressure. However, an explicit pressure could be applied as well by installing an explicit measure of semantic generality. For example, the average time between rule activation could be recorded using again, for example, temporal difference learning techniques. The resulting semantic generality measure could be used to bias offspring reproduction or deletion. This approach is implemented in the ACS2 system (Butz, 2002), which keeps track of the average time until activation of a classifier and uses this measure to further bias deletion if accuracy is insignificantly different. In general, a two-stage tournament-based selection process can be implemented that first selects classifiers based on fitness but takes semantic generality into account if fitness does not differ significantly.

The same approach could be applied with respect to syntactic generality. Instead of monitoring the average time until application, it would be possible to consider syntactic generality in the conditions of competing classifiers.

We saw that XCS uses subsumption deletion to prevent the generation of unnecessary over-specialized offspring once an accurate, more general classifier is found. However, sub-

sumption is restricted to deterministic problem cases since it only applies once the classifier error estimate drops below $\varepsilon < \varepsilon_0$.

Another approach might bias the effect of mutation on specificity. Given that high accuracy is reached, mutation could mainly be changed to a generalizing mutation operator that may only cause generalizations in the offspring conditions. This approach was taken in ACS2 in conjunction with heuristic specialization in order to achieve a maximally general problem representation. However, the heuristic approach again relies on determinism in the environment and would need to be replaced with a more general mechanism.

Thus, generalizing mutation as well as subsumption rely on an appropriate setting of the error tolerance threshold ε_0. Since this threshold cannot be assumed to be available in general, only biased selection or deletion mechanisms appear applicable in the general case.

Regardless of the generalization method, though, it needs to be assured that the generality criterion does not compare totally unrelated classifiers. As mentioned before, different problem niches may require different syntactic generality levels. Moreover, during learning different problem niches may be at different developmental stages exhibiting potentially significantly different current generality. Thus, a generality competition should only be applied in subsets of similar classifiers as the naturally defined classifier subsets in an action set.

10.1.4 Which Selection For Solution Growth?

The applied fitness approach guides the selection mechanisms in reproduction and deletion. There are two fundamentally different types of selection: *population-wide selection* and *niche selection*.

As we have seen, XCS applies niche selection for reproduction but population-wide selection for deletion. Apart from the implicit generalization effect analyzed in Chapter 4 and the effect on niching analyzed in Chapter 5, the combination also results in a general search bias. Since GA reproduction is directly dependent on niche occurrence, search effort is dependent on the frequency of niche occurrence. As shown in the niche support analysis in Chapter 5, the number of representatives of a niche is directly correlated with the frequency of niche occurrence so that more frequently occurring niches are searched more excessively and are represented by more classifiers. Parameter θ_{GA} is able to partially balance this bias to ensure the maintenance of infrequently occurring niches. However, the balancing effect is limited and has the side-effect of delaying the overall learning speed. Thus, population-wide selection may be advantageous if niche occurrence frequency is not correlated with the importance of learning a niche.

Again, the decision which selection method to choose is certainly very problem dependent. First, the occurrence frequency of problem instances may not reflect learning importance. Thus, applying niche reproduction but population-wide deletion results in an inappropriately skewed niche support distribution in the problem representation. Second, even if the occurrence frequency reflects niche importance, different subspaces in the problem space may require different computational search effort to find an appropriate solution. Thus, even if importance is reflected by niche occurrence, additional search biases may need to be applied.

An additional problem is caused by the reliance on (potentially sparse) reinforcement feedback. Since this reliance results in potential high noise in the offspring parameter estimates, reproduction needs to be balanced with evaluation preventing niche loss as well as overproduction of under-evaluated, inaccurate classifiers. Restricting selection to the problem niche in which evaluation occurred naturally balances evaluation and reproduction. The XCS classifier system is doing just that.

Deletion usually serves additional purposes in the LCS realm. On the one hand, deletion should be biased to delete useless classifiers. On the other hand, deletion should be biased to delete classifiers in overpopulated niches. These two criteria are combined in the deletion criterion in XCS biasing deletion on the action set size estimate as well as on the fitness estimate.

Besides the selection biases, it is important to decide on which selection technique to apply. The two direct competitors are proportionate selection and tournament selection (although alternatives exist, see e.g. Goldberg, 1989). As discussed in Chapter 1, proportionate selection is more appropriate for balancing problem niches possibly maintaining a large number of equally fit classifiers. Tournament selection strives to convergence to the best individual and thus is best suited to propagate best solutions. In XCS, niche reproduction is designed to find the accurate, maximally general classifier for the current problem niche, represented by the current action set. Thus, tournament selection works best. On the other hand, deletion is designed to assure niche maintenance and an equal distribution of action set sizes so that proportionate selection is more suitable for deletion.

The ZCS system applies population-wide selection striving to maintain a larger number of equally important subsolutions via selection and deletion. The fitness sharing approach assures that equal fitness corresponds to equal importance. Thus, in ZCS proportionate selection is the appropriate choice for reproduction and deletion.

10.1.5 Niching for Solution Sustenance

Besides the problems of learning a problem solution, niching techniques need to assure solution sustenance. XCS's mechanism of choice is niche selection for reproduction in combination with population-wide selection for deletion. Since deletion is biased towards deleting larger niches and selection is based on niche frequency, problem subsolutions are sustained with a computational effort that is linear in the number of required subsolutions. However, we also saw that in overlapping problem solutions additional niching mechanisms might stabilize performance further.

ZCS niching is accomplished by fitness sharing in conjunction with proportionate selection. As shown in Harik (1994), niching based on sharing and proportionate selection assures niche sustenance in time exponential in the population size. Thus, it is very effective. However, also in this case overlapping problem solutions can interfere.

Thus, other techniques might be more appropriate, as the diverse crowding techniques. For example, in the mentioned restricted tournament replacement (Harik, 1994; Pelikan, 2002), niche maintenance and niche support balance are assured further by restricting classifier replacement to the close neighborhood of the offspring. Thus, instead of selecting classifiers for deletion at random, classifiers with structural similarities may yield better deletion choices leading to better subsolution maintenance.

In the most extreme case, niche-based deletion may be applied similar to niche reproduction restricting the set of deletion candidates to the niche in which a classifier was reproduced. The ACS2 system applies this method ensuring equal support of all problem niches (Butz, 2002).

With respect to overlapping subsolutions, however, deletion may still cause competition when an instance applies that is overlapping. In this case, two mechanisms are imaginable: (1) Apply two-stage deletion that biases towards deleting the classifier that is represented more often; or (2) apply restricted tournament replacement techniques inside the action set to replace a most similar classifier. Both techniques can serve as niche stabilizers and should be investigated in problems in which overlapping problem solutions are inevitable (like the carry problem, see Appendix C).

10.1.6 Effective Evolutionary Search

Besides the resulting search biases caused by the fitness approach, reproduction, and deletion, the actual search in the neighborhood of currently promising solutions is guided by mutation and crossover operators. As discussed before, mutation is designed to search in the

local, syntactic (genotypic) neighborhood of the current best solutions. Crossover, on the other hand, searches in the neighborhood defined by the two parental classifiers. Thus, while mutation in combination with selection searches for better solutions in the local neighborhood, simple crossover searches more globally combining the information of two classifiers in the generated offspring rules.

Both, mutation and crossover can be disruptive in that they may cause unwanted search biases. Mutation may cause an unwanted specialization effect as observed in our evolutionary pressure analysis (Chapter 4) potentially causing the population to grow unnecessarily. Large mutation rates may additionally disrupt the neighborhood search potentially resulting in completely random offspring generation. Dependent on the problem, both effects should be prevented. On the other hand, though, mutation is important to maintain diversity in the population and cause the suggested search in the local neighborhood. Thus, a balanced mutation rate is important to maintain diversity and assure effective neighborhood search without causing disruption.

Crossover may be disruptive as well as already expressed in Holland's original schema theory (Holland, 1975). Uniform crossover assumes attribute independence consequently only recombining attribute value frequencies. One- and two-point crossover assume that neighboring attributes in the coded problem instance may be correlated. Thus, in problems in which such a correlation is known, one- or two-point crossover should be applied. Additionally, dependent on if classifiers are selected for reproduction in problem niches or population-wide, crossover can be expected to cause less or more disruptions, respectively, since classifiers in a problem niche can be expected to be more similar. On the other hand, crossing over two highly correlated classifiers may hardly cause any innovative search bias.

Competent crossover operators, such as the BB-based offspring generation or the Bayes net-based offspring generation (applied to XCS in Chapter 6), can overcome both potential drawbacks of simple crossover operators (i) preventing distribution and (ii) propagating innovation by combining previously successful substructures (Goldberg, 2002). Results of the combination with the XCS classifier system showed that the generated dependency structure learned from the patterns of the current best classifiers effectively biases the neighborhood search replacing crossover and diminishing the importance of mutation. Interestingly, we used the *global* dependency knowledge to generate *local* offspring biasing local offspring generation towards the global neighborhood structure in the problem. Similar model-based offspring generation mechanisms can certainly be expected to work well in other LCS systems given a problem structure in which local neighborhood search and simple crossover operators are not sufficient to evolve a complete problem solution.

10.1.7 Conclusions

The purpose of this section was to give the reader further intuition about which mechanisms are responsible for which biases in LCSs. The facetwise approach enabled us to consider each learning aspect relevant for LCSs in separation discussing various LCS mechanisms that result in similar or alternative learning biases with respect to one facetwise theory aspect.

With the knowledge in this section in hand, it is hoped that the reader is ready to successfully design his own XCS or LCS system, tuned to the task at hand, adapted to the most suitable representation, exploiting expected problem properties. Accordingly, the next section outlines how these tools might be used to create a cognitive LCS as envisioned in Holland (1976) and Holland and Reitman (1978), terming the first LCS implementation the *cognitive system CS1*.

10.2 Towards LCS-based Cognitive Learning Structures

With the facetwise knowledge of LCS mechanisms in hand, this section considers the design of cognitive systems utilizing XCS and LCS-based learning mechanisms.

To consider a learning system *cognitive*, the system should satisfy certain criteria. First, learning should be done online interacting with an outside environment. Second, learning may not be supervised in the sense that an error signal should not be provided directly from the outside environment. Only internal gradients and reinforcement can serve as learning signals. Third, the evolving knowledge should be distributed in a network structure to make it robust against failures of small computational entities (neurons). Fourth, learning should start from scratch with potentially many learning predispositions but without any explicitly coded knowledge.

This section approaches the design of cognitive learning architectures in a facetwise approach. Since the proposition of a complete facetwise approach for cognition is beyond the scope of this thesis, we focus on several cognitive aspects, which we believe to be most important. For each aspect, we outline how LCS-like learning mechanisms may serve as a valuable algorithm to design the desired learning structure adjusted to the cognitive task at hand.

In particular, we focus on the following four cognitive aspects:

1. **Predictive representations**: In order to learn internally without any explicit teaching signal, internal predictions need to be learned that express the expected future.

2. **Incremental learning**: Online learning from scratch requires incremental learning growing the intended knowledge base.

3. **Hierarchical representations**: Our modular, hierarchical world composed of progressively more complex substructures needs to be reflected in the learning structure.

4. **Anticipatory behavior**: Anticipatory behavior serves as the control mechanism controlling attention, action decision, and action execution. Meanwhile, it serves as the learning signal using predictions, expectations and their violations, and intentions to guide and stabilize the learning progress.

The next sections discuss these issues in turn leading to the proposition of a multi-modular, hierarchical, anticipatory learning structure.

10.2.1 Predictive and Associative Representations

Predictive representations gained significant research interest over the recent years. The recent workshops on Predictive Representation of World Knowledge (Sutton & Singh, 2004) as well as the second workshop on Anticipatory Behavior in Adaptive Learning Systems (Butz, Sigaud, & Swarup, 2004) reflect only a small fraction of the actual present interest.

Reinforcement learning recently suggested the framework of *predictive state representations* (PSRs) (Littman, Sutton, & Singh, 2002; James & Singh, 2004). In PSRs, states are represented by the expectable predictions of the future in that state. Thus, states are not represented by themselves but rather by a collection of predictions about the future. Analyses showed that the representation is as powerful as POMDP models (hidden Markov models) and of size no larger than the corresponding POMDP model (Littman, Sutton, & Singh, 2002). Since POMDP-based model learning methods scale rather poorly, it is hoped that PSR-based learning methods may be learned faster and have better scalability properties (Singh, Littman, Jong, & Pardoe, 2003; James & Singh, 2004). However, at least the current implementations of PSRs appear hardly cognitive requiring an explicit distinction of states represented by a concatenation of predictions.

Observable operator models (OOMs) are an alternative approach to PSRs that currently appears to be slightly further developed indicating their online learning capability and scalability (Jaeger, 2000; Jaeger, 2004). Observable operators are operator activities that reflect

a certain event in the environment. Only most recently, Jaeger (2004) proposed an online learning algorithm for learning the required set of suitable events and for learning their respective probabilities. As are PSRs, OOMs are an alternative approach to other POMDP models where OOMs are at least as expressive as POMDP models.

Could we be able to actually learn predictive representations in a much more distributed, LCS-like learning fashion? First approaches in that direction, also referred to as *anticipatory learning classifier systems* (ALCSs) were proposed elsewhere (Stolzmann, 1998; Butz, Goldberg, & Stolzmann, 2002). The ACS2 system (Butz, 2002) showed to be able to learn in diverse problem domains also solving the multiplexer problem but additionally being able to handle predictions in diverse maze and blocks world scenarios. However, ACS2 depends on a deterministic environment and is designed to learn complete predictions of next states. Also other, similar learning systems such as YACS (Gérard & Sigaud, 2001b; Gérard & Sigaud, 2001a) depend on determinism.

The ALCS system MACS (Gérard & Sigaud, 2003) is modularized in that different predictive modules predict different perceptual (or sensory) features in the environment. This has the advantage of evolving separate modules for potentially independent problem features. Currently, though, the modules are learned completely independently so that knowledge exchange or knowledge reuse is impossible. Since behavior of problem features can be expected to be correlated, an algorithm needs to be designed that detects, propagates, and exchanges useful structure among the learning modules. In this way, distinct (predictive) modules may reuse lower level structures for higher level predictions. Thus, the learning structure needs to be biased towards detecting *expectable* problem features that should be accessible to all related learning modules.

The evolutionary-based learning mechanisms available in XCS do not rely on determinism but showed to also work well in many stochastic problem settings. They are also well-suited to ignore task-irrelevant attributes for which they require only linear additional computational effort. Thus, an XCS-like predictive mechanism may be created that predicts future features similar to other ALCSs.

The envisioned system has the advantage to be able to ignore irrelevant features as well as to handle noisy problem cases well. Thus, XCS may be used as a modular, predictive learning architecture in which the prediction of the next state is evaluated in terms of predictive accuracy. The exchange of knowledge may be realized by enabling crossover mechanisms over the boundaries of the different predictive modules. Figure 10.1 visualizes the proposed learning structure in which each XCS module learns to predict the behavior of one problem feature. The predictive modules exchange dependency structures by the means of simple

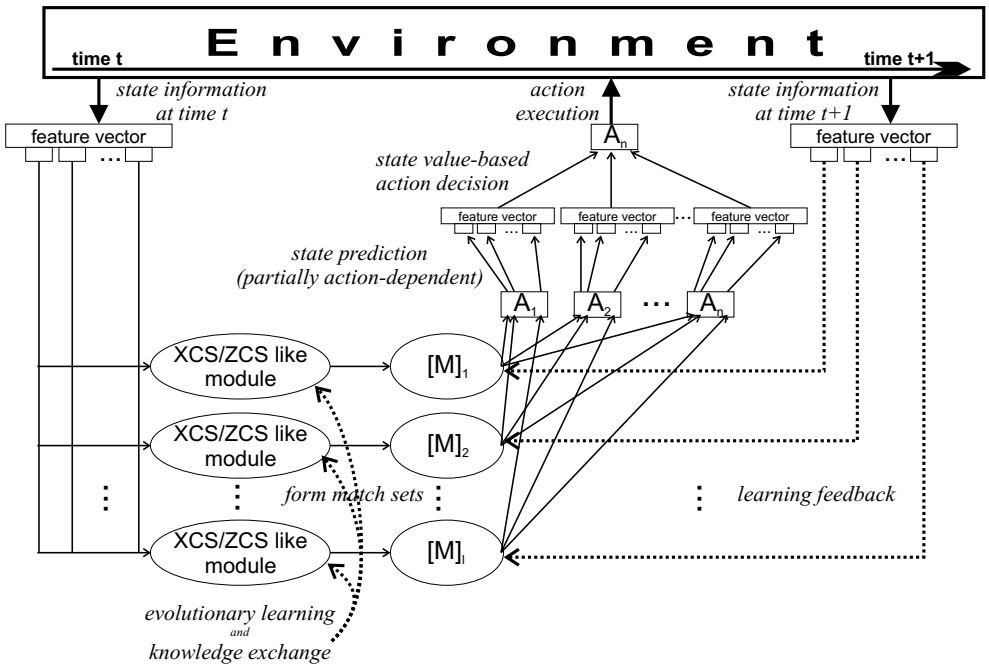


Figure 10.1: The proposed modular, predictive learning architecture is well-suited for an XCS-based learning implementation. Predictions are formed in parallel. Learning is based on the difference between the predicted and the encountered environmental state information. Knowledge exchange between modules is accomplished by the means of genetic recombination of BB structures.

crossover or the model-based dependency structures introduced in Chapter 6.

In close distance to predictive learning lies the realm of associative learning. The only difference to predictive learning is that in associative learning both patterns are available at the same time and not in succession. Nonetheless, associations may be learned by two interconnected predictive modules that learn to predict the one pattern from the other and vice versa. This method can easily be implemented with the proposed predictive extension of the XCS classifier system. The advantage in comparison to neural network structures is the additional adaptivity gained by the evolutionary component in XCS structuring connectivity and relevancy.

For predictions and associations even more importantly, the expected nature of the interaction should be investigated in more detail. That is, what can be expected to be associated? Are their structural commonalities that can be exploited? For example, when predicting change, higher activity can be expected to cause stronger change. However, when associating patterns, their representation may strongly differ between modules and thus should be investigated in further detail. Thus, an interactive pattern associator should be implemented

that is predisposed to learn certain structures in the problem. The top-down influence then decides which structures are most effective for predicting other dependencies and structures.

10.2.2 Incremental Learning

Predictions cannot be known exactly beforehand so that learning of the predictions can only occur online interacting with the body and the environment. Learning predictions must be predisposed or biased towards learning certain structures. Parts of this predisposition are already realized by the natural sensory and motor capabilities. Thus, predictions can and should be biased towards learning certain structures and connections but need to be grown-up, adjusted to the actual body-related "hardware".

The idea of the importance of incremental learning is not new. Several previous promising systems can be identified such as fast incremental learning in the ART networks (Grossberg, 1976a; Grossberg, 1976b) as well as initial LCS-like rule-based methods (Wilson, 1987b), to name a few early approaches.

The advantage of an incremental growth of a network structure, rather than fixed size and fixed connectivity from the beginning, is to direct computational needs to where they are really needed. Thus, growth is a critical time in development—as are the first few month in the life of an infant shaping its mind for the rest of its life. However, growth is limited and once a distribution is reached that accomplishes current activity and tasks satisfactory, it can be expected to become more and more fixed. Growth takes time and energy and thus fights for limited resources. The resulting resource distribution, manifested in terms of neural concentration and connectivity, predisposes the cognitive structure. In the next learning stage, the existing connectivity may be shaped and finally only small adjustments may be possible.

What are the most desired features of a useful growth algorithm? Growth should be interactive in that grown structure immediately influences the further structural growth. Growth should be modular consisting of different, interdependent growth centers that have distinct representational responsibilities. The learning design does not only realize the center's learning capability but also its connectivity and interactivity with other growth centers. These factors lead to a strong predisposition of the overall growing structure, which then learns in the interdependent centers that detect and propagate structure for proper prediction and consequent activity generation.

Growth needs to ensure covering, that is, the growing structure needs to assure that all potential inputs can be processed. Once the growth process is converging to a stable

structure, no input or occurring state should result in a random reaction but should rather be influenced by all related experiences previously encountered by the growing cognitive system.

The essential question is how to create a growing, distributed structure mutually shaped and controlled by actual activity as well as by the influence of the outside world partially resulting from own activity. LCSs provide a solid framework that assures the evolution of a distributed knowledge representation. As we have seen in Chapter 9, the XCS system is comparable to a neural network learning structure but actually forms connectivity and detects relevancy online. While a neural network assumes relevancy of all inputs by default, XCS detects relevancy on the fly using evolutionary techniques to propagate and distribute successful relevancies. Thus, the XCS system might serve very well as the actual growing mechanism evolving connectivity effectively shaping relevancy.

Again, the advantage of the reinforcement-dependent learning enables XCS to be flexible and to shape its connectivity while further growing—reproducing successful structures mimicking duplication, mutating structures to search in the local-neighborhoods, recombining structures to detect relations and interdependencies. Once the structure is grown, other, more weight-oriented mechanisms might condense, shape, and refine the grown structure.

10.2.3 Hierarchical Architectures

Recent insights in cognitive and neural processing mechanisms suggest that many brain areas are structured hierarchically. Different but related areas usually interact bidirectionally. For example, the well-studied auditive system of humans and birds is hierarchically structured (Feng & Ratnam, 2000). Lower-level areas respond to specific sounds, such as phonemes or syllables, and higher levels use the extracted features responding to larger chunks of auditive input, such as words or song parts.

Even more prominently, vision research has shown that there are many areas in the brain that are responsible for feature-extraction and basic visual stimulus processing. These mechanisms work in parallel and are hardly influenced by the bottleneck of cognitive attention (Pashler, 1998). However, it was also shown that the structure extraction mechanisms are strongly influenced by top-down processes such as attention related to object-properties, location properties, color properties, as well as predictive behavioral properties (Pashler, Johnston, & Ruthruff, 2001).

Despite this strong evidence from neuroscience, only few artificial neural systems have been developed that mimic such hierarchical, interactive structures.

Hierarchical clustering forms hierarchical problem representations. However, regardless if the taken approach is agglomerative, in which many clusters are progressively combined, or divisive, in which one cluster is progressively divided, the clusters are usually non-overlapping in each level of the hierarchy (see e.g. Duda, Hart, & Stork, 2001 and citations therein). Hierarchical self-organizing maps (Dittenbach, Merkl, & Rauber, 2000) are available that grow hierarchies on demand. However, also these hierarchies are non-overlapping. Both approaches do not consider to reuse certain useful clusters in a lower level by several clusters on the higher level.

In the RL literature, hierarchical RL has gained increasing interest over the recent years (Sutton, Precup, & Singh, 1999; Dietterich, 2000; Drummond, 2002). Sutton, Precup, and Singh (1999) propose the framework of *options*—actions extended in time—that can be used hierarchically. For example, in a problem with four interconnected rooms, an option might learn the way to the doorway. This option may then be combined with other options effectively learning to cross several rooms to reach a reward position much more effectively. Similarly, the hierarchy may be used to evolve a predictive representation of the environment. Barto and Mahadevan (2003) give an excellent overview of the current state-of-the-art in hierarchical reinforcement learning. While all papers emphasize the importance of automatically learning such hierarchies, most of the papers focus on how to apply options but not how to learn a hierarchy in the first place. Most recently, it was suggested to learn the hierarchy on the fly by detecting *transitional states* in the explored environment applying simple statistical methods monitoring the temporal state distribution online (Butz, Swarup, & Goldberg, 2004; Simsek & Barto, 2004).

Clearly, neural structures including the multi-layer perceptron are reusing the representations emerging in hidden layer units for classification or prediction on the higher level. Even more potential seems to lie in the recently emerging multi-layer kernel-based representations in which kernel-based feature extraction units evolve in the lower-layer, which are then combined on the higher layer for the task at hand.

One of the simplest of these networks might be the radial-basis function network (RBF), which evolves radial-basis kernels approximating the proximity of the actual data to each of the hidden units (see e.g. Hassoun, 1995 and citations therein). Other kernels are expected to be useful for other types of problems with other types of non-linearities, that is, other types of dependency-structures, or BBs, in the problem space. Shawe-Taylor and Cristianini (2004) give an excellent overview over which kernels may be suitable for which types of problems and expectable patterns.

The XCS framework is ready to serve as the module that detects relevant kernel structures

growing them problem dependently. The datamining analysis in Chapter 8 has shown that XCS can detect and evolve patterns in real-valued data problems. Thus, XCS is readily combinable with kernel-based feature extraction mechanisms in which the feature vector could consist of a set of kernels that evolve with the classifier population over time.

Another, more hierarchical approach is imaginable in which lower-level structures evolve interactively with higher level structures. In fact, lower-level kernels may be suitable to be reused in many higher-level classification rules, function approximation neurons, or predictor units. Hierarchical structures are imaginable in which the upper levels reward the lower levels for providing useful structure activity.

Since XCS is designed to learns structural patterns online solely dependent on reward-based feedback, it appears to be a very suitable learning mechanism for each of the levels. Figure 10.2 shows how such an hierarchical XCS structure could look like. Activity is propagated bottom-up. Each layer is predisposed by the top-down activity to process bottom-up activity (attention). Thus, subsequent activity is predisposed by the inner state of the system but ultimately determined by the bottom-up input. Note that the architecture certainly does not need to be strictly bottom-up, top-down biased. Essentially, it is imaginable that the bottom-up influence comes from other activity centers. It might not necessarily stem from sensory activity.

Upper layers provide feedback in terms of reinforcement rewarding lower layers for their activating activity. The lower layers compete for the feedback evolving a complete problem distribution clustered in a way suitable for higher level predictions, classifications or whatever the task might be. Additional layer-owned feedback may be provided to ensure a base-rate activity in each layer.

The advantage of the solely reinforcement-based learning is that feedback might come from any other layer so that the shaping of one layer may be coordinated by the needs of whatever layer is interested in the evolving information. This enables the designer to strongly bias the learning structure by defining neighborhoods of layers that influence each other to a degree controlled by the neighborhood structure.

We recently experimented with a related structure that searches for hierarchical language patterns over time (Butz, 2004b). The proposed cognitive sequence learning architecture (COSEL) showed to be able to detect most frequent characters, syllables, and words in a large text document growing a hierarchical problem representation from scratch. While COSEL is still in its infancy, growing hierarchical architectures in which the top-down influence shapes the evolution of the bottom-up growth, stabilizing and guiding it, might be one of the key elements to learn stable, distributed, hierarchical cognitive learning architectures.

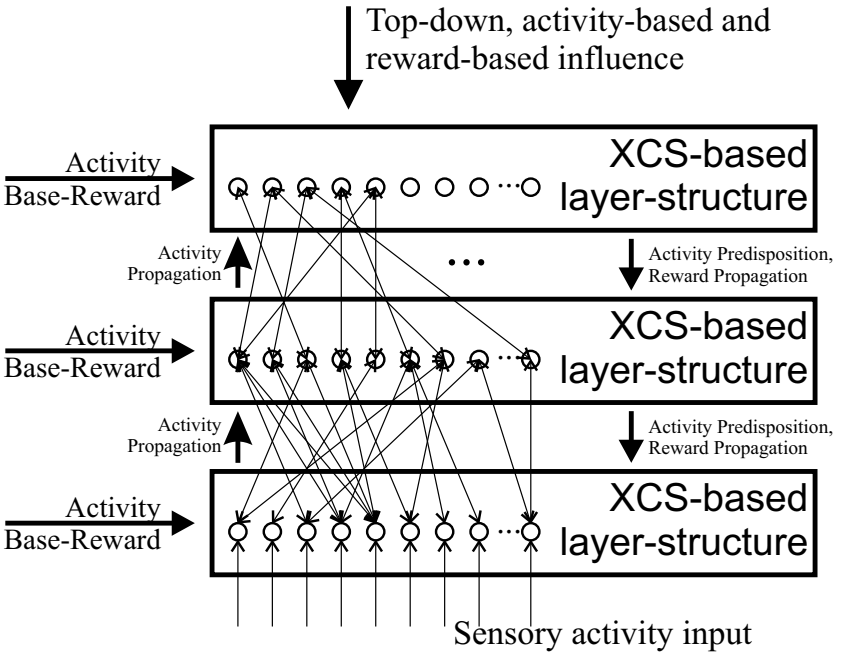


Figure 10.2: The proposed hierarchical XCS-based learning architecture propagates neural activity bottom-up and predisposes the propagation top-down. Learning is purely reward-based and depends on the appropriateness of the activity predispositions and the interdependent activity structure.

10.2.4 Anticipatory Behavior

Over the last decades, psychology realized that most goal-directed behavior is actually not stimulus-response driven but rather influenced by the expectations about the future (Tolman, 1932; Hoffmann, 1993; Hoffmann, 2003; Hommel, Müsseler, Aschersleben, & Prinz, 2001). It was shown that anticipations influence behavioral selection, initialization, and execution (Kunde, Koch, & Hoffmann, 2004). Moreover, anticipation guides attentional processes (Pashler, Johnston, & Ruthruff, 2001). Finally, anticipation is responsible for higher level cognition potentially leading to self-controlled consciousness in terms of a predictive, attention-based control system (Taylor, 2002).

While a more detailed review of anticipatory behavior including its manifestations in different research disciplines as well as its potential for the development of competent cognitive architectures is out of the scope of this thesis (see for example Butz, Sigaud, & Gérard, 2003; Butz, 2004a for more detailed overviews), it needs to be noted that predictions, associations, incrementally growing structures, and hierarchies are some of the fundamental building blocks of anticipatory behavior.

Prediction is that part of an anticipation that predicts future changes, states, and

progress. Anticipatory behavior then is the part that decides on how the predictions actually influence the cognitive system. As mentioned before, associations are similar to predictions and similarly serve as structures that enable anticipatory behavior.

Hierarchical structures enable anticipations on multiple levels of abstraction in time and space. Starting from very basic sensory inputs on the lowest level, a hierarchical representation needs to develop in which higher levels represent progressively larger and more abstract representations of the environment. Each level may serve as a predictor for the lower level and as an information propagator to the higher level. In that way, each level predicts the next lower level input given its current activity resulting in an anticipatory behavioral influence on the lower level. In the mean time, the activity will be propagated to the next higher level influencing the resulting next prediction from that level.

The hierarchical abstraction can be accomplished in space and time. In space, the abstraction leads to progressively larger and more abstract objects or entities in the environment. In time, on the other hand, the hierarchy leads to larger sequences in time. In relation to speech, for example, the hierarchy would abstract from phonemes to syllables to words to whole sentences. In relation to face recognition, the hierarchy would abstract from edges, corners, colors, shades, and other basic visual features to eyes, nose, mouth, ears, and so forth to relations between them to overall relations between those relations. Each higher-level abstract representation allows the prediction of lower level relations—as the imagination of a face facilitates the detection of that face or the expectation of a train of thought facilitates word recognition.

Thus, the hierarchies enable anticipatory processes in time and space having high potential for explaining attentional phenomena but also the mentioned influences on action decision, initiation, and execution. Moreover, learning is influenced by this top-down influence since only activity can cause learning and the top-down controlled predisposition of activity consequently predisposes learning.

It remains to be shown how these hierarchical levels may be interconnected and how they might be grown guided by its own activity and the environmental influences. Nonetheless, the XCS framework provides a potentially valuable tool to model levels in the hierarchy. Since the system is only dependent on reinforcement feedback, higher levels may reward lower-level activity that propagates activity successfully. Classifiers in a layer thus serve as activity propagation entities to the next higher level, but even more importantly serve as the predictors of lower level activity. Accurate predictors will gain more reward and thus establish themselves in the hierarchical level. Anticipations manifest themselves by higher level classifier activity that predisposes classifier activity on the lower level.

10.2.5 Conclusions

We discussed four fundamental building blocks of cognitive learning structures including predictive and associative representations, incremental learning, hierarchical learning, and anticipatory behavior. The former three types basically enable the forth type which in turn potentially further influences and stabilizes the development of the former three. Thus, the four discussed aspects of cognition are highly interactive passing reward messages that cause connective attraction and repulsion, growth and death.

We showed that the XCS framework is a suitable learning system to face the design challenges of such a highly interactive, cognitive learning system. Certainly, the successful design of such an architecture is not straight forward and a single reward feedback may not be enough to develop, propagate, and grow substructures successfully. The XCS framework, nonetheless, appears to have the potential to serve as the tool that develops the proposed hierarchically layered learning structures interactively—adaptive to continuous input and feedback and robust to noisy and partially misleading learning signals.

Chapter 11

Summary and Conclusions

This thesis has investigated rule-based evolutionary online learning systems, often referred to as learning classifier systems (LCSs). We proposed and pursued a facetwise problem analysis approach that showed that the XCS classifier system is an online learning and generalizing reinforcement learning (RL) system that evolves a complete, maximally accurate and maximally general problem solution quickly and reliably. Our scalability analysis showed that XCS can PAC-learn restricted *k-DNF* functions of maximal order of difficulty k_m. However, XCS is an RL system that also showed to be able to solve large multistep RL problems optimally ignoring additional, irrelevant attributes effectively.

In the following we first provide a detailed summary of the thesis findings and achievements. Next, we provide conclusions focusing on learned lessons for system analysis and design and on future LCS research directions.

11.1 Summary

This thesis has addressed the problem domains of optimization problems, classification problems, and RL problems. While usually one global solution is searched in an optimization problem, a classification problem can be represented by a distributed problem solution in which different subsolutions are responsible for different problem subspaces. Since RL problems only provide, potentially delayed, reinforcement feedback, additionally, appropriate reward estimation and propagation is necessary. All problems require a search for problem solution structure. Often, problems can be expected to contain lower-level structure, referred to as building-blocks (BBs), that guide to an overall problem solution. RL techniques, such as the well-known Q-learning, are available for appropriate structure evaluation by the means to reward estimation and distribution. Genetic algorithms (GAs) serve as BB

identification and recombination mechanisms that inductively evolve, or grow, an overall solution structure.

LCSs are able to solve RL problems and classification problems combining RL and GA techniques. Hereby, RL techniques serve as the critique component that estimates rule fitness and propagates reward estimates. GAs generate new rules evolving and propagating rule structure. Overall, LCSs evolve distributed problem solutions represented by a population of rules in which subsets of rules are responsible for different problem subspaces.

Our facetwise LCS theory approach suggests that the following four major aspects need to be satisfied to assure the successful application of LCSs:

1. Evolutionary pressures need to apply appropriately.
2. Solution growth and sustenance need to be enabled and ensured.
3. Solution search needs to be effective.
4. Multistep problems pose additional challenges concerning reward propagation and environmental exploration.

In particular, evolutionary pressures need to ensure that classifier parameter estimations guide towards better solutions and ultimately to the optimal solution disabling overgenerals. Interactively, generalization pressure needs to apply to favor most compact solutions but may not overrule the fitness pressure. Parameter estimates need to be approximated as fast and accurate as possible but also need to be adjusted appropriately to the population dynamics where necessary. Additionally, fitness overestimations in young and inexperienced classifiers need to be avoided.

To ensure solution growth, *evaluation time* needs to be available to initialize the evolutionary process. Secondly, the *supply* of minimal order classifier representatives needs to enable the identification of better classifiers. Additionally, time for *identification and reproduction* needs to assure the accurate identification of better classifiers as well as the growth of these better classifiers. Once growth is assured, *solution sustenance* requires the implementation of effective niching techniques such as fitness sharing, niche reproduction, and restricted replacement.

To ensure the fastest search for the problem solution, search operators of mutation and recombination need to result in *effective neighborhood search* with respect to current best solutions. Hereby, *structural differences* in solution complexity in different *problem subspaces* need to be detected and properly incorporated into the search operators.

In *multistep* RL problems, distortions in *niche frequencies*, proper *behavioral policies*, and *accurate reward propagation* need to be considered, additionally. Due to the interdependence of RL-based rule evaluation and GA-based rule evolution, reward needs to be propagated most accurately. The implemented behavioral policy may partially prevent niche frequency distortions.

With the facetwise approach in hand, we then introduced and analyzed the XCS classifier system. XCS is an accuracy-based LCS that is designed to reliably evolve a complete, maximally accurate, and maximally general problem solution. The RL component in XCS utilizes adapted Q-learning mechanisms to accurately estimate rule reward prediction, prediction error, and fitness. The GA component is a steady-state niche GA that reproduces, mutates and recombines classifiers in problem niches (action sets) and deletes classifiers from the whole population.

Our evolutionary pressure analysis showed that XCS applies an *implicit semantic generalization* pressure combining niche-based reproduction with population-wide deletion. The derived *specificity equation* quantifies this pressure taking the effect of simple mutation into account. *Fitness pressure* is the main drive towards maximal accuracy. *Tournament selection* with tournament sizes proportional to the current niche size assures reliable fitness pressure with a constant, minimal strength that depends on the tournament size proportion. Finally, *subsumption deletion* provides further *syntactic generalization* pressure in deterministic problems.

The analysis of solution growth and sustenance revealed that XCS is a machine learning competitive learning system whose learning complexity scales polynomially in solution complexity, problem length, learning reliability, and solution accuracy. The order of the polynomial depends on the minimal order problem complexity k_m, which denotes the minimal number of specified attributes necessary to decrease class distribution entropy.

Also in classification problems, lower order BB structure may need to be identified, propagated, and recombined effectively. In XCS, BBs are subsets of specified attributes that increase accuracy only as a whole. This is the case in hierarchically structured classification problems in which substructures are evaluated independently and then combined in an overall solution evaluation mechanism. To make the evolutionary search more effective in such BB-hard classification problems, statistical structure analysis methods such as Bayesian networks can replace recombination. However, in contrast to the incorporation of such mechanisms in GAs, in LCSs local structural subsolution differences need to be accounted for so that statistical models need to be adjusted to local problem properties resulting in a niche-biased statistical offspring generation.

The subsequent analysis in binary classification problems confirmed the theoretic analysis. We showed that XCS performance is robust to noisy reward feedback and noisy problem instances. The investigated problems included large multiplexer problems, combined multiplexer problems that partially require appropriate structure identification and propagation, problems that require an overlapping problem solution representation, and problems in which different problem subspaces require different subsolution complexities.

Also the datamining analysis confirmed XCS's competence. Comparing XCS's performance in forty-two datasets with ten other machine learning techniques yielded competitive performance scores. In fact, no machine learning method was found that consistently outperformed XCS but XCS did not consistently outperform any other learning method, either (except majority). The results confirm the expectable performance differences due to the different learning biases in each machine learner and the different structural properties of each dataset. However, it needs to be kept in mind that XCS learns a problem solution online receiving only reinforcement feedback whereas all other learners learn offline in batch mode analyzing the whole training data with respect to the available supervised feedback. Thus, the learning mechanism in XCS appears much more general searching online for relevant problem structure.

This hypothesis was then confirmed in the RL section, in which we applied XCS to diverse maze and blocks world problems. Realizing the connection to neural-based RL techniques, we introduced gradient-based updates to stabilize reward estimation and propagation. Hereby, we noticed that XCS combines the neural-based reward estimation approach with a somewhat tabular-based structure identification mechanism. The GA component searches for solution subspaces that yield similar reward values whereas the reward component evaluates the evolved subspaces. This is the reason why residual gradient approaches appear unnecessary in the current XCS system. As a whole, the performance analysis showed that XCS is able to solve a variety of multistep RL problems being able to focus on relevant problem features. Additionally, irrelevant, randomly changing problem features require only low-order (often linear) polynomial computational effort, dependent on the solution complexity and the size of the problem space.

Finally, we showed how the facetwise approach may be carried over to analyze other LCSs and design more advanced LCS architectures. We reviewed all major LCS learning mechanisms and their importance and impact on learning performance. With such an LCS mechanisms toolbox in hand, we then proposed the design of modular, hierarchical, interactive learning architectures in which each learning module may be realized by an LCS-based learning mechanism. Since cognitive systems require reinforcement-based predictive online

learning mechanisms, LCSs may serve as an important entity for the successful design of competent, interactive, and adaptive cognitive learning systems.

11.2 Conclusions

As shown in this thesis, a facetwise analysis approach enables the successful analysis of highly interactive learning systems such as the XCS classifier system. We were not only able to qualitatively understand the system but we were also able to quantify learning behavior and computational learning requirements. Moreover, the facetwise approach enabled modular system design and system improvement as shown with the introduction of tournament selection, statistics-based search techniques, and gradient-based update techniques. Thus, the facetwise approach enables flexibility and robustness in system analysis as well as in system design.

11.2.1 Expansion of Facetwise System Analysis and Design Approach

How can a facetwise approach be applied to other systems and other problems? First, it is necessary to understand underlying problem structure, available feedback, as well as the intended solution structure. For example, if the solution structure can be expected to be linearly dependent on problem input, linear problem solvers such as the Naive Bayes approach for classification can be expected to be most effective. On the other hand, if subspace-wise constant solutions are expected, an approach that partitions subspaces appropriately—such as the investigated XCS system—can be expected to be most effective.

In the case of a piecewise solution approximation, two issues need to be considered: (1) How may the space be partitioned most effectively? (2) What is the expected solution structure in each of the partitions? For example, if the partitioning is expected to be linear in the problem features, linear decision-theoretic partitioning will be most appropriate as done in the incremental model tree learner introduced in Potts (2004). Similarly, if subsolutions can be expected to be linear transformations, linear solution transformations and learners, such as linear regression, can be expected to be most effective.

The understanding of problem structure (concept space structure) and expected problem solution structure (hypothesis space structure) leads to the design of an appropriate problem solver. Learning biases need to be targeted to search the expected structure. For example, the XCS system is targeted to detect and propagate substructures that increase

prediction accuracy. XCS propagates subsolutions relevant for different problem subspaces. The subspaces are defined by rule conditions, which in their current form are hypercubes in the problem space.

With the knowledge of addressed problem structure, solution structure, solution representation, and biased learner at hand, it is then possible to determine the competence of the learner with respect to its computational requirements and scalability. In essence, it is necessary to identify how much computational effort is necessary to identify different types of problem subspaces and solution structures. In XCS, the computational effort was derived by population sizing equations as well as expected learning time. Hereby, learning initialization, solution growth, and solution sustenance had to be addressed.

Expected structural similarities and thus subsolution structure exchange can be expected to be helpful in the evolution of a complete problem solution. For example, a multi-layer neural network strongly assumes structural similarities in different problem subspaces reusing hidden layer representation for the derivation of problem solutions. A tabular approach, such as tabular Q-learning, assumes no structural similarities developing independent solution representations for each problem instance (such as each possible state-action pair in an MDP). An XCS-based approach exchanges structural information to identify similarly structured problem subspaces relevant for different problem solutions. The enhanced offspring generation biased towards processing globally relevant dependencies is consequently helpful in problems in which similar solution structures are present in different problem subspaces (represented by the investigated hierarchical, BB-hard problems).

11.2.2 LCS-based Future Research Directions

Four future research directions can be identified. (1) The analysis and design of other LCSs using a similar facetwise approach. (2) The design of advanced LCSs that may evolve more complex problem partitions as well as predictions. (3) The identification of further problem difficulties and their solution. (4) The application of LCS-based learning mechanisms to more complex, interactive problems.

The previous chapter already outlined how other LCS learning mechanisms may be analyzed in a similar facetwise fashion. Especially the ZCS system appears to function very similar to the XCS system so that a similar analysis approach should yield similar qualitative and quantitative insights. Also, the improvement of related LCS systems appears possible exchanging and evaluating different mechanisms for, for example, semantic and syntactic generalization, niching, or recombinatory search. Similarly, the design of completely new

LCS systems is possible combining the discussed learning mechanisms along the lines of the facetwise design decomposition.

Besides the design of similar LCSs, LCSs may be extended to handle more complex problem spaces, problem space partitions, and problem solution representations. As mentioned earlier, the linear separators in condition parts may be replaced by kernel-based condition structures such as polynomial kernels or radial-basis functions (Shawe-Taylor & Cristianini, 2004). Which kernels are available results in different space partition biases so that kernels should be chosen wisely with respect to the problem at hand. Similarly, more complex prediction structures may be chosen such as the linear predictions used in Wilson (2001a) or the action-dependent linear predictions introduced in Wilson (2004). Other forms of predictions are possible neither restricted to an one-dimensional prediction space nor to linear predictions.

Although we identified BB-hard problems for classification, we did not identify any deceptive problems in that accuracy may actually mislead the evolutionary search towards unnecessary specialization or further generalization. In fact, given a certain classification entropy in a certain problem subspace $\mathcal{S}$, any partition of $\mathcal{S}$ is assured to yield one subspace in which entropy does not increase (that is, information loss is not possible). However, entropy increase is possible in one of the subspaces so that fitness guidance may not be sufficiently strong to explore the latter subspace. Fitness sharing and niche reproduction somewhat alleviate this problem but further investigations are necessary to fully understand this issue.

Nonetheless, our analysis and the experimental evaluations have confirmed that XCS is a widely applicable learning system that detects, propagates, and grows distributed problem solutions accurately predicting reinforcement. Thus, XCS and LCS mechanisms in general appear ready to be applied in more interactive, modular, or hierarchical learning structures. As proposed in John Holland's original publication of the cognitive system CS1 (Holland, 1976; Holland, 1977), LCSs may serve as the main process that grows highly interactive, cognitive learning structures solely guided by reward propagation between interactive LCS-based learning modules. Applying our facetwise approach to the design of these learning structures, we seem to be one step closer to accomplish the creation of the envisioned competent cognitive learning systems.

Appendix A

Notation and Parameters

Problem Notation

Problem Notation

Concept Learning Problem

$\mathcal{S}$	problem space
$s \in \mathcal{S}$	problem instance
$\mathcal{S} = \{0,1\}^l$	binary problem space of length l
l	problem length l (that is, number of features in a problem instance)
$\mathcal{A}$	problem classes
$a \in \mathcal{A}$	current class
$\mathcal{A} = \{0,1\}$	binary (two-class) problem
n	number of problem classes

Reinforcement Learning Problem

$\mathcal{S}$	set of possible sensory inputs
$s \in \mathcal{S}$	current sensory input
l	number of sensory features
$\mathcal{A}$	available actions
$a \in \mathcal{A}$	current action
n	number of possible actions
$\mathcal{R}$	reward feedback
$r \in \mathcal{R}$	current reward

Problem Difficulty

k_m	minimal number of attributes necessary to decrease entropy of class distribution
k_d	order of problem difficulty—the target concept space is exponential in k_d

Simple Learning Classifier System

LCS Parameters

N	maximal population size
ϵ	ϵ-greedy strategy parameter
γ	reward discount factor
β	learning rate
$P_\#$	don't care probability
μ	probability of mutating a condition attribute (or the action)
χ	probability of applying the chosen crossover operator

Classifier Parameters

C	condition part—in binary domains, $C \in \{0, 1, \#\}^l$
A	action part $A \in \mathcal{A}$
R	reward prediction

Other Notations

$[P]$	classifier population
$[M]$	match set
$[A]$	action set
$\sigma(cl)$	specificity of condition part of classifier cl; in binary domains specificity equals to the number of specified attributes over l
$\sigma[X]$	average specificity in classifier set X

XCS Classifier System

LCS Parameters

N	maximal population size
$P_\#$	don't care probability
P_I, ε_I, F_I	default parameter initialization values set to 500, 500, and .01, respectively
ϵ	ϵ-greedy strategy parameter
γ	reward discount factor
β	learning rate
$\alpha, \varepsilon_0, \nu$	accuracy determination parameters
θ_{GA}	threshold that controls GA invocation
τ	the tournament size (proportion of current action set size)
μ	probability of mutating a condition attribute (or the action)
χ	probability of applying the chosen crossover operator
θ_{del}	threshold that requires minimal experience for fitness influence during deletion
δ	fraction of mean fitness below which deletion probability is further decreased
θ_{sub}	threshold that requires minimal experience for subsumption

Classifier Parameters

C	condition part—in binary domains, $C \in \{0, 1, \#\}^l$
A	action part $A \in \mathcal{A}$
R	reward prediction
ε	mean absolute reward prediction error
F	fitness (in macro classifiers)
as	the mean action set size the classifier is part of
ts	the time the classifier was part of an action set in which the GA was applied
exp	the number of evaluation steps the classifier underwent so far
num	the numerosity, that is, the number of micro-classifiers represented by this (macro-) classifier

Other Notations

$[P]$	classifier population
$[M]$	match set
$[A]$	action set
$\sigma(cl)$	specificity of condition part of classifier cl; in binary domains, specificity equals to the number of specified attributes over l
$\sigma[X]$	average specificity in classifier set X
κ	the current accuracy of a classifier
κ'	the current set-relative accuracy of a classifier
ρ	the (combined) reward received by the current action set
$P(\mathcal{A})$	prediction array estimating the value of each possible action
$P(A)$	the predicted value of action A

Appendix B

Algorithmic Description

This section gives a concise algorithmic description of the XCS classifier system. A similar description can be found in Butz and Wilson (2002). It differs in the offspring classifier initialization technique as well as the subsumption technique.

The algorithmic description uses the dot notation to refer to classifier parameters. Subfunctions are indicated by pure capital letters. We use indentation to indicate the length of a sub-clause such as the effect of an `if` statement or a `for` loop. For reasons of readability, we refer to classifier parameters by denoting the parameter with the index of the classifier.

Overall Learning Iteration Cycle

At the beginning of a run, XCS's parameters may be initialized, iteration time needs to be reset, and problem may be loaded / initialized. XCS's population $[P]$ may be either left empty or may be initialized with the maximal number of classifiers N, generating each classifier with a random condition and action and initial parameters. The two methods usually differ only slightly in their effect on performance. Moreover, classifier generation due to covering assures that the problem space is immediately covered with respect to the problem space distribution resulting in an additional advantage. After XCS and problem initializations, the main learning loop is called.

XCS:
```
 1 initialize problem env
 3 initialize XCS
 4 RUN EXPERIMENT
```

In the main loop *RUN EXPERIMENT*, the current situation or problem instance is

first sensed (received as input). Second, the match set is formed from all classifiers that match the situation. Third, the prediction array $P(A)$ is formed based on the classifiers in the match set. Based on $P(A)$, one action is chosen and the action set $[A]$ is formed. Next, the winning action is executed. Then the previous action set $[A]_{-1}$ (if this is a multistep problem and there is a previous action set) is modified using the previous reward ρ_{-1} and the largest action prediction in the prediction array. Moreover, the GA may be applied to $[A]_{-1}$. If a problem ends on the current time-step (single-step problem or last step of a multistep problem), $[A]$ is modified according to the current reward ρ and the GA may be applied to $[A]$. The main loop is executed as long as the termination criterion is not met. A termination criterion is, e.g., a certain number of trials or a persistent 100% performance level. In this thesis we usually stick to a fixed number of trials.

RUN EXPERIMENT():

```
 1 while(termination criteria are not met)
 2    s ← env: get situation
 3    GENERATE MATCH SET [M] out of [P] using s
 4    GENERATE PREDICTION ARRAY P(A) out of [M]
 5    a ← SELECT ACTION according to P(A)
 6    GENERATE ACTION SET [A] out of [M] according to a
 7    env: execute action a
 8    r ← env: get reward
 9    if([A]_-1 is not empty)
10       ρ ← r_-1 + γ * max P(A)
11       UPDATE SET [A]_-1 using P possibly deleting in [P]
12       RUN GA in [A]_-1 considering s_-1 inserting in [P]
13    if(env: eop)
14       ρ ← r
15       UPDATE SET [A] using P possibly deleting in [P]
16       RUN GA in [A] considering s inserting in [P]
17       empty [A]_-1
18    else
19       [A]_-1 ← [A]
20       r_-1 ← r
21       s_-1 ← s
```

Sub-procedures

The main loop specifies many sub-procedures essential for learning in XCS. Some of the procedures are more or less trivial while others are complex and themselves call other sub-procedures. This section describes all procedures specified in the main loop. It covers all relevant processes and describes them algorithmically.

Formation of the Match Set

The *GENERATE MATCH SET* procedure receives the current population $[P]$ and the current problem instance s as input. While matching is rather trivial, covering may take place. Covering is called when the number of different actions represented by matching classifiers is less than the parameter θ_{mna} (this is usually set to the number of possible classifications n in a problem). Thus, *GENERATE MATCH SET* first looks for the classifiers in $[P]$ that match s and next, checks if covering is required. Note that a classifier generated by covering can be directly added to the population since it differs from all current classifiers.

GENERATE MATCH SET($[P]$, σ):
```
 1 initialize empty set [M]
 2 for each classifier cl in [P]
 3    if(DOES MATCH classifier cl in situation s)
 4       add classifier cl to set [M]
 5 while(the number of different actions in [M] < θ_mna)
 6    GENERATE COVERING CLASSIFIER cl_c considering [M] and covering s
 7    add classifier cl_c to set [P]
 8    DELETE FROM POPULATION [P]
 9    add classifier cl_c to set [M]
10 return [M]
```

In the following paragraphs describe the sub-procedures included in the *GENERATE MATCH SET* algorithm.

Classifier Matching The matching procedure is commonly used in LCSs. A 'don't care'-symbol # in C matches any symbol in the corresponding position of s. A 'care', or non-# symbol, only matches with the exact same symbol at that position. This description focuses on binary problems and thus a ternary alphabet for XCS's conditions. Note that classifier conditions may be expressed more efficiently specifying the care-positions only. In this way, the matching process can be sped-up significantly since only the specified

attributes are relevant. Only if all comparisons hold, the classifier matches s and the procedure returns *true*.

DOES MATCH(cl, s):

```
1 for each attribute x in cl.C
2    if(x ≠ # and x ≠ the corresponding attribute in s)
3       return false
4 return true
```

Covering Covering occurs if there are less than θ_{mna} actions are represented by classifiers in $[M]$. If covering is triggered, a classifier is created whose condition matches s generating don't care symbols in the condition part with probability $P_{\#}$. The classifier action is chosen randomly from among those not present in $[M]$.

GENERATE COVERING CLASSIFIER($[M]$, σ):

```
 1 initialize classifier cl
 2 initialize condition cl.C with the length of s
 3 for each attribute x in cl.C
 4    if(RandomNumber[0,1[ < P_#)
 5       x ← #
 6    else
 7       x ← the corresponding attribute in s
 8 cl.A ← random action not present in [M]
 9 cl.P ← p_I
10 cl.ε ← ε_I
11 cl.F ← f_I
12 cl.exp ← 1
13 cl.ts ← actual time t
14 cl.as ← 1
15 cl.num ← 1
```

The Prediction Array

After the generation of the match set, the prediction array $P(A)$ provides reward estimates of all possible actions. The *GENERATE PREDICTION ARRAY* procedure considers each classifier in $[M]$ and adds its prediction multiplied by its fitness to the prediction value total for that action. The total for each action is then divided by the sum of the fitness values for that action to yield the system prediction.

GENERATE PREDICTION ARRAY([M]):

```
1 initialize prediction array P(A) to all null
2 initialize fitness sum array FSA to all 0.0
3 for each classifier cl in [M]
4    P(cl.A) ← P(cl.A) + cl.P * cl.F
5    FSA(cl.A) ← FSA(cl.A) + cl.F
6 for each possible action A
7    if(FSA(A) is not zero)
8       P(A) ← P(A) / FSA(A)
9 return P(A)
```

Choosing an Action

XCS is not dependent on one particular action-selection method, and any of a great variety can be employed. For example, actions may be selected randomly, independent of the system predictions, or the selection may be based on those predictions—using, e.g., roulette-wheel selection or simply picking the action with the highest system prediction. However, note that a non-random action selection algorithm biases niche occurrence and may thus require additional population size adjustments.

In our *SELECT ACTION* procedure we illustrate a combination of *pure exploration*—choosing the action randomly—and *pure exploitation*—choosing the best one. This action selection method is known as *ϵ-greedy* selection in the reinforcement learning literature (Sutton & Barto, 1998). Due to the mentioned distribution effect, XCS learns more stable if the GA is applied in exploration steps only.

SELECT ACTION(PA):

```
1 if(RandomNumber[0,1[ > ε)
2    //Do pure exploitation here
3    return the best action in P(A)
4 else
5    //Do pure exploration here
6    return a randomly chosen action
```

Formation of the Action Set

After the match set $[M]$ is formed and an action is chosen for execution, the *GENERATE ACTION SET* procedure forms the action set out of the match set. It includes all classifiers

that propose the chosen action for execution.

GENERATE ACTION SET([M], a):

```
1 initialize empty set [A]
2 for each classifier cl in [M]
3    if(cl.A = a)
4       add classifier cl to set [A]
```

Updating Classifier Parameters

The reinforcement portion of the update procedure follows the pattern of Q-learning (Sutton & Barto, 1998). Classifier predictions are updated using the immediate reward and the discounted maximum payoff anticipated on the next time-step. Note that in single-step problems, the prediction is updated using only the direct reward ρ.

Each time a classifier enters the set $[A]$, its parameters are modified in the order: exp, ε, p, f, and as. Hereby, the update of the action set size estimate is independent from the other updates and consequently can be executed at any point in time. While the updates of exp, p, ε, and as are straightforward, the update of f is more complex and requires more computational steps. Thus, we refer to another sub-procedure. Finally, if the program is using action set subsumption, the procedure calls the *DO ACTION SET SUBSUMPTION* procedure.

UPDATE SET([A], P, [P]):

```
 1 for each classifier cl in [A]
 2    cl.exp++
 3    //update prediction error cl.ε
 4    if(cl.exp < 1/β)
 5       cl.ε ← cl.ε + (|P - cl.P| - cl.ε) / cl.exp
 6    else
 7       cl.ε ← cl.ε + β * (|P - cl.P| - cl.ε)
 8    //update prediction cl.P
 9    if(cl.exp < 1/β)
10       cl.P ← cl.P + (P - cl.P) / cl.exp
11    else
12       cl.P ← cl.P + β * (P - cl.P)
13    //update action set size estimate cl.as
14    if(cl.exp < 1/β)
15       cl.as ← cl.as + (∑_{c∈[A]} c.num - cl.as) / cl.exp
16    else
17       cl.as ← cl.as + β * (∑_{c∈[A]} c.num - cl.as)
18 UPDATE FITNESS in set [A]
19 if(doActionSetSubsumption)
20    DO ACTION SET SUBSUMPTION in [A] updating [P]
```

Fitness Update The fitness of a classifier in XCS is based on the relative *accuracy* of its reward predictions. The *UPDATE FITNESS* procedure first calculates the classifier's accuracy κ using the classifier's prediction error ε. Then the classifier's fitness is updated using the *normalized accuracy* computed in lines 7-9.

UPDATE FITNESS([A]):

```
 1 accuracySum ← 0
 2 initialize accuracy vector κ
 3 for each classifier cl in [A]
 4    if(cl.ε < ε_0)
 5       κ(cl) ← 1
 6    else
 7       κ(cl) ← α * (cl.ε / ε_0)^{-ν}
 8    accuracySum ← accuracySum + κ(cl) * cl.num
 9 for each classifier cl in [A]
10    cl.F ← cl.F + β( κ(cl) * cl.num / accuracySum - cl.F)
```

The Genetic Algorithm in XCS

The final sub-procedure in the main loop, *RUN GA*, is also the most complex. First of all, the action set is checked to see if the GA should be applied at all with respect to the θ_{GA} threshold. Next, two classifiers (i.e. the parents) are selected and reproduced. After that, the resulting offspring are possibly crossed and mutated. If the offspring are crossed, their prediction values are set to the average of the parents' values. Finally, the offspring are inserted in the population, followed by corresponding deletions. If GA subsumption is being used, each offspring is first tested to see if it is subsumed by any classifier in the current action set; if so, that offspring is not inserted in the population, and the subsuming parent's numerosity is increased.

RUN GA([A], s, [P]):

```
 1 if(actual time t - Σ_{cl∈[A]} cl.ts · cl.num /  Σ_{cl∈[A]} cl.num > θ_GA)
 2    for each classifier cl in [A]
 3       cl.ts ← actual time t
 4    parent_1 ← SELECT OFFSPRING in [A]
 5    parent_2 ← SELECT OFFSPRING in [A]
 6    child_1 ← copy classifier parent_1
 7    child_2 ← copy classifier parent_2
 8    child_1.num ← child_2.num ← 1
 9    child_1.exp ← child_2.exp ← 0
10    if(RandomNumber [0,1[ < χ)
11       APPLY CROSSOVER on child_1 and child_2
12       child_1.P ← child_2.P ← (parent_1.P + parent_2.P)/2
13       child1.ε ← child_2.ε ← (parent_1.ε + parent_2.ε)/2
14       child_1.F ← child_2.F ← 0.1 * (parent_1.F + parent_2.F)/2
18    for both children child
19       APPLY MUTATION on child according to s
20       if(doGASubsumption)
21          if(DOES SUBSUME parent_1, child)
22             parent_1.num++
23          else if(DOES SUBSUME parent_2, child)
24             parent_2.num++
25          else
26             INSERT child IN POPULATION
27       else
28          INSERT child IN POPULATION
29       DELETE FROM POPULATION [P]
```

Roulette-Wheel Selection The Roulette-Wheel Selection chooses a classifier for reproduction proportional to the fitness of the classifiers in set $[A]$. First, the sum of all the fitness values in the set $[A]$ is computed. Next, the roulette-wheel is spun. Finally, the classifier is chosen according to the roulette-wheel result.

```
SELECT OFFSPRING([A]):
 1 fitnessSum ← 0
 2 for each classifier cl in [A]
 3     fitnessSum ← fitnessSum + cl.F
 4 choicePoint ← RandomNumber [0,1[ * fitnessSum
 5 fitnessSum ← 0
 6 for each classifier cl in [A]
 7      fitnessSum ← fitnessSum + cl.F
 8      if(fitnessSum > choicePoint)
 9          return cl
```

Tournament Selection As shown in Chapter 4, tournament selection with tournament sizes proportional to the action set size proved to improve XCS's performance in all problem settings. The following algorithm describes the selection approach using tournament selection with an approximate tournament size of a fraction τ of the action set size.

```
SELECT OFFSPRING([A]):
 1 clb ← null
 2 while(clb = null)
 3     maxf ← 0
 4     for each classifier cl in [A]
 5        if(cl.F/cl.num > maxf)
 6           for each mikro-classifier cl'
 7              if(RandomNumber [0,1) < τ)
 8                 clb ← cl
 9                 maxf ← cl.F/cl.num
10                 break
11 return clb
```

Crossover The crossover procedure is similar to the standard crossover procedure in GAs. In the *APPLY CROSSOVER* procedure we show uniform crossover. The action part is not affected by crossover.

APPLY CROSSOVER(cl_1, cl_2):

```
1 for(i ← 0to Length of cl1.C)
2   if(RandomNumber [0,1) < 0.5)
3     swap cl1.C[i] and cl2.C[i]
```

Mutation While crossover does not affect the action, mutation takes place in both the condition and the action. A mutation in the condition flips the attribute to one of the other possibilities showing free mutation. Since in XCS most of the time is spent in processes that operate on the whole population such as matching and deletion, the algorithms used for mutation as well as crossover only slightly affect efficiency.

APPLY MUTATION(cl, σ):

```
 1 for(i ← 0to Length of cl.C)
 2    if(RandomNumber [0,1[ < μ)
 3       if(RandomNumber [0,1[ < 0.5)
 4          if(cl.C[i] = #)
 5             cl.C[i] ←0
 6          else
 7             cl.C[i] ← #
 8       else
 9          if(cl.C[i] = #)
10             cl.C[i] ←1
11          else
12             if(cl.C[i] = 0)
13                cl.C[i] ←1
14             else
15                cl.C[i] ←0
16 if(RandomNumber [0,1[ < μ)
17    cl.A ← a randomly chosen other possible action
```

Insertion in and Deletion from the Population

This section covers processes that handle the insertion and deletion of classifiers in the current population [P].

The *INSERT IN POPULATION* procedure searches for an identical classifier. If one exists, the latter's numerosity is incremented; if not, the new classifier is added to the population.

INSERT IN POPULATION(cl, [P]):

```
1 for all c in [P]
2    if(c is equal to cl in condition and action)
3       c.num++
4       return
5 add cl to set [P]
```

Roulette-Wheel Deletion The deletion procedure realizes two ideas at the same time: (1) It assures an approximately equal number of classifiers in each action set, or environmental 'niche'; (2) It removes low-fitness individuals from the population.

The deletion procedure chooses individuals (for deletion) by roulette-wheel selection. Deletion is only triggered if the current population size is larger than N. If deletion is triggered, proportionate selection is applied based on the deletion vote. If the chosen classifier is a macro-classifier, its numerosity is decreased by one. Otherwise, the classifier is removed from the population.

DELETE FROM POPULATION([P]):

```
1 if(∑_{c∈[P]} c.num < N)
2    return
3 voteSum ← 0
4 for each classifier c in [P]
5    voteSum ← voteSum + DELETION VOTE of c in [P]
6 choicePoint ← RandomNumber [0,1) * voteSum
7 voteSum ← 0
8 for each classifier c in [P]
9    voteSum ← voteSum + DELETION VOTE of c
10   if(voteSum > choicePoint)
11      if(c.num > 1)
12         c.num--
13      else
14         remove classifier c from set [P]
```

The Deletion Vote As mentioned above, the deletion vote realizes niching as well as removal of the lowest fitness classifiers. The deletion vote of each classifier is based on the action set size estimate as. Moreover, if the classifier has sufficient experience and its fitness is significantly lower than the average fitness in the population, the vote is increased in inverse proportion to the fitness. In this calculation, since we are

deleting one micro-classifier at a time, we need to use as fitness the (macro)classifier's fitness divided by its numerosity. The following *DELETION VOTE* procedure realizes all this.

DELETION VOTE(cl, [P]):
```
1 vote ← cl.as * cl.num
2 averageFitnessInPopulation ← Σ_{c∈[P]} c.F / Σ_{c∈[P]} c.num
3 if(cl.exp > θ_del and cl.F / cl.num < δ * averageFitnessInPopulation)
4     vote ← vote * averageFitnessInPopulation / ( cl.F / cl.num)
5 return vote
```

Subsumption

Two subsumption procedures were introduced into XCS in Wilson (1998). The first, 'GA subsumption', checks an offspring classifier to see if its condition is logically subsumed by the condition of an accurate and sufficiently experienced action set member. If so, the offspring is not added to the population, but the subsumer's numerosity is incremented. If GA subsumption is applied, it occurs within the procedure *RUN GA*. It is detailed within that procedure, and is not called as a sub-procedure (though it could be). However, the sub-procedure *DOES SUBSUME* is called, and is described in this section.

If 'action set subsumption' is enabled the action set is searched for the most general classifier that is both accurate and sufficiently experienced. Then all other classifiers in the set are tested against the general one to see if it subsumes them. Any classifiers that are subsumed are eliminated from the population.

DO ACTION SET SUBSUMPTION([A], [P]):
```
 1 initialize cl
 2 for each classifier c in [A]
 3    if(c COULD SUBSUME)
 4       if(cl is empty or c IS MORE GENERAL than cl)
 5          cl ← c
 6 if(cl is not empty)
 7    for each classifier c in [A]
 8       if(cl IS MORE GENERAL than c)
 9          cl.num ← cl.num + c.num
10          remove classifier c from set [A]
11          remove classifier c from set [P]
```

Subsumption of a classifier For a classifier to subsume another classifier, it must first be sufficiently accurate and sufficiently experienced. This is tested by the *COULD SUBSUME* function. Then, if a classifier could be a subsumer, it must be tested to see if it has the same action and is really more general than the classifier that is to be subsumed. This is the case if the set of situations matched by the condition of the potentially subsumed classifier form a proper subset of the situations matched by the potential subsumer. The *IS MORE GENERAL* tests this. The *DOES SUBSUME* procedure combines all the requirements.

DOES SUBSUME(cls, clt):

```
1 if(cls.A = clt.A)
2    if(cls COULD SUBSUME)
3       if(cls IS MORE GENERAL than clt)
4          return true
5 return false
```

COULD SUBSUME(cl):

```
1 if(cl.exp > θ_sub)
2    if(cl.ε < ε_0)
3       return true
4 return false
```

IS MORE GENERAL(clg, cls):

```
1 if (the number of # in clg.C ≤ the number of # in cls.C)
2    return false
1 for(i ← 0to Length of clg.C)
5    if(clg.C[i] ≠ # and clg.C[i] ≠ cls.C[i])
6       return false
9 return true
```

Appendix C

Boolean Function Problems

A short definition of all used binary classification (concept learning) problems is provided with augmenting examples. We also provide a short explanation of the characteristics of each problem. Some of the problems are augmented with a special reward scheme.

Multiplexer

Multiplexer problems have been shown to be challenging with respect to other machine learning methods. De Jong and Spears (1991) show the superiority of LCSs in comparison to other machine learning approaches such as C4.5 in the multiplexer problem. XCS was applied to multiplexer problems beginning with its first publication (Wilson, 1995).

The multiplexer function is defined for binary strings of length $k + 2^k$. The output of the multiplexer function is determined by one of the bit 2^k value bits. The location is determined by the k address bits. For example, in the six multiplexer $f(\texttt{100010}) = 1$, $f(\texttt{000111}) = 0$, or $f(\texttt{110101}) = 1$. Any multiplexer can also be written in disjunctive normal form (DNF) in which there are 2^k conjunctions of length $k + 1$. The DNF of the 6-multiplexer is

$$6MP(x_1, x_2, x_3, x_4, x_5, x_6) = \neg x_1 \neg x_2 x_3 \vee \neg x_1 x_2 x_4 \vee x_1 \neg x_2 x_5 \vee x_1 x_2 x_6 \qquad \text{(C.1)}$$

Using reward as feedback, a correct classification results in a payoff of 1000 while an incorrect classification results in a payoff of 0.

In terms of fitness guidance, the multiplexer problem provides slight fitness guidance although not directly towards specifying the k position bits. Specifying any of the 2^k remaining bits gives the classifier a bias towards class zero or one. Thus, classifiers with more ones or zeros specified in the 2^k remaining bits have a lower error estimate on average. Specifying position bits (initially by chance) restricts the remaining relevant bits. These properties

result in the fitness guidance in the multiplexer problem. The multiplexer problem needs $|[O]| = 2^{k+2}$ accurate, maximally general classifiers in order to represent the whole problem accurately. With respect to problem difficulty, the multiplexer problem has a minimal order $k_m = 1$ and problem difficulty $k_d = k + 1$. The initial fitness guidance is actually somewhat misleading in that first the specialization of the value bits increases accuracy.

Interestingly, although there exists a complete, accurate, and maximally general non-overlapping problem solution, there are potential overlapping rules with equal generality. For example, classifier `0#00##`→`0` is accurate and maximally general but it overlaps with classifiers `000###`→`0` and `01#0##`→`0`. Due to the space distribution, the latter two classifiers are expected to erase the former. However, due to continuous search, the former can be expected to continuously reappear. This phenomenon is investigated in Chapter 5 in the niche support section with respect to the influence of overlapping classifiers.

Layered Multiplexer

A layered reward scheme for the multiplexer problem was introduced by Wilson (1995) intending to show that XCS is able to handle more than two reward levels (particularly relevant for multistep problems, in which reward is discounted and propagated). Later (as shown in Chapter 7), it was shown that the layered reward scheme is actually easier for XCS since the used scheme provides stronger fitness guidance (Butz, Kovacs, Lanzi, & Wilson, 2004).

Reward of a specific instance-classification case is determined by

$$R(S, A) = (\text{value of } k \text{ position bits } + \text{ return value }) \cdot 100 + \text{ correctness } \cdot 300 \qquad \text{(C.2)}$$

This function assures that the more position bits are specified, the less different the resulting reward values can be. Thus, XCS successively learns to have all position bits specified beginning with the left-most one. Similar to the 'normal' multiplexer problem above, the layered multiplexer needs $|[O]| = 2^{k+2}$ accurate, maximally general classifiers in order to represent the whole problem accurately.

Also with respect to problem difficulty, the layered multiplexer is equal to the normal multiplexer. However, due to the layered reward scheme fitness guidance immediately propagates position attributes and overlapping classifiers are not as prominent as in the normal multiplexer problem.

xy-Biased Multiplexer

The xy-biased multiplexer was originally designed to investigate fitness guidance. The problem is designed to exhibit more direct fitness guidance.

The problem is composed of x biased multiplexer problems. Roughly, a biased multiplexer is a Boolean multiplexer whose output is biased towards outcome zero or one. We can distinguish a *zero-biased multiplexer* in which the zero output is more likely to be correct and a *one-biased multiplexer* in which the one output is more likely to be correct. A biased multiplexer, is defined over $l = y + (2^y - 1)$ bits; as in the Boolean multiplexer the first y bits represent an address which indexes the remaining $2^y - 1$ bits. In a biased multiplexer, it is not possible to address one of the configurations: y address bits would address 2^y bits, but in this case only $2^y - 1$ bits are available. The missing configuration results in the bias. A *one-biased multiplexer* always returns one when the y address bits are all 1, i.e., the output one is correct for one configuration more than would be in the case in a Boolean multiplexer with y address bits. For instance, in a *one-biased multiplexer* with two address bits the condition `11###` always corresponds to the output `1`. A *zero-biased multiplexer* always returns zero when the y address bits are all 0s.

A set of *biased multiplexers* can be used to build an xy-biased multiplexer. The problem uses the x reference bits to refer to one of the 2^x biased multiplexers. y refers to the size of each one of the biased multiplexers involved. The first half of the 2^x biased multiplexers are zero-biased and the second half are one-biased. Overall, a problem instance is of length $l = x + 2^x \cdot (y + 2^y - 1)$ bits.

As an example, let us consider the *11-biased multiplexer* (*11bMP*). The first bit of the multiplexer, x_0, refers to one of the two biased multiplexers, which consist of two bits; the first biased multiplexer is *zero biased*, the second biased multiplexer is *one biased*. The *11-biased multiplexer* for input 00111 outputs 0 because the first zero refers to the (first) zero biased multiplexer and the second zero determines output zero; input 01011 has output 0 as well; input 10010 has output 1 since the one-biased multiplexer is now referenced; input 10000 would be output 0. The *11-biased multiplexer* can be written in DNF as:

$$11 - bMP(x_1, x_2, x_3, x_4, x_5) = \neg x_1 x_2 x_3 \vee x_1 x_4 \vee x_1 \neg x_4 x_5 \tag{C.3}$$

Again, using reward as feedback, a reward of 1000 indicates a correct classification and a reward of 0 an incorrect one.

To represent a y-biased multiplexer accurately, XCS needs to evolve $2 \cdot 2^{y+1} - 1$ classifiers. Thus, the complete, accurate, and maximally general problem representation in the xy-biased

multiplexer requires $|[O]| = 2^{x+y+2} - 2^{x+1}$ classifiers. Due to the bias in the outcome, XCS can be expected to detect the x reference bits quickly. Once the relevancy of one biased-multiplexer is specified, first the address bits and then the value bits are expected to be correctly specified.

Hidden Parity Function

Kovacs and Kerber (2001) triggered the notion of a hidden parity function to show the dependence of the problem difficulty on the necessary number of accurate, maximally general classifiers $[O]$ in XCS. Hidden parity functions are the extreme case of a BB of size k and can be compared with the XOR function. In the hidden parity function only k bits are relevant in a string of length l. The number of ones modulo two determines the class of the input string. For example, in a $k = 3$, $l = 6$ hidden parity problem (in which the first k bits are assigned to be the relevant ones) string `110000` as well as `011000` would be in class zero while string `010000` as well as string `111000` would be class one. In DNF, the hidden parity problem with $k = 3$ and $l = 6$ can be written as:

$$HP3(x_1, x_2, x_3, x_4, x_5, x_6) = x_1x_2x_3 \vee \neg x_1\neg x_2x_3 \vee \neg x_1x_2\neg x_3 \vee x_1\neg x_2\neg x_3 \tag{C.4}$$

The reward scheme is again `1000` for a correct classification and zero otherwise. The complete problem solution of XCS is of size 2^{k+1}.

The hidden parity problem is the most difficult problem with respect to problem difficulty since the minimal order of difficulty is equal to the overlap problem difficulty ($k_m = k_d = k$). Thus, XCS needs to have classifiers with all k relevant attributes specified or needs to generate them by chance due to mutation as described in Chapter 5. The complete problem solution requires the specialization of all k attributes and thus the optimal population has size $|[O]| = 2^{k+1}$.

Count Ones

The count ones problem is similar to the hidden parity function in that only k positions in a string of length l are relevant. However, it is very much different in that the schema with minimal order that provides fitness guidance is of order one ($k_m = 1$). Any specialization of only ones or only zeros in the k relevant attributes makes a classifier more accurate.

The class in the count ones problem is defined by the number of ones in the k relevant bits. If this number is greater than half k, the class is one and otherwise the class is zero. For

example, considering a count ones problem of length $l = 5$ with $k = 3$ relevant attributes, for example, problem instances `11100` as well as `01111` belong to class one whereas `01011` or `00011` belong to class zero. In DNF, the problem can be written as:

$$CO3(x_1, x_2, x_3, x_4, x_5) = x_1x_2 \vee x_1x_3 \vee x_2x_3 \tag{C.5}$$

We again use the 1000/0 reward scheme in this problem. Due to the counting approach, even a specialization of only one attribute of the k relevant attributes biases the probability of correctness and consequently decreases the prediction error estimate. Thus, this problem provides a very strong fitness guidance and thus $k_m = 1$. The order of difficulty equals $\lceil \frac{k}{2} \rceil$ since more than half of the features need to be one to determine class one. The number of accurate, maximally general classifiers is $[O] = 4\binom{k}{\lceil \frac{k}{2} \rceil}$. Note that the optimal set of accurate, maximally general classifier is strongly overlapping.

Layered Count Ones

The layered count ones problem is identical to the count ones problem except for the reward scheme used. The received reward does now depend on the number of ones in the string. That is, a reward of $1000 \cdot \sum_{i=1}^{k} value[i]$ is provided if the classification was correct and 1000 minus that reward otherwise. This scheme causes any specialization in the k position bits to result in a lower deviation of reward outcomes and thus in a lower reward error estimate. Thus, as the count ones problem, the layered count ones problem provides a very strong fitness guidance and the evolutionary process can be expected to stay rather independent of the string length. However, note that there are more classes to distinguish than in the layered count ones problem. Since any change in one of the relevant bits changes the resulting reward, all k bits need to be specified in any classifier that specifies the input string accurately. Thus, the number of accurate, maximally general classifiers $|[O]| = 2^{k+1}$. Unlike in the count ones problem, the accurate, maximally general classifiers do not overlap.

Carry Problem

The carry problem adds the the above spectrum of problems the additional problem of differently sized problem niches. That is, different solution subspaces require different numbers of specialized attributes.

In the carry problem essentially two binary numbers of length k are added together. If the addition yields a carry (that is, if the addition results in a number greater than

expressible with the number of bits used), then the class of the problem instance is one and zero otherwise. For example, the 3,3-carry problem adds two binary numbers of length three so that problem instance `001011` yields value `100` (class zero) or `101010` yields value `111` (class zero). One the other hand, `100100` yields value `1000` and thus class one as does `111001`. The carry problem can be written in DNF-form as follows:

$$3,3-Carry(x_1,x_2,x_3,x_4,x_5,x_6) = \\ x_1x_4 \vee x_1x_2x_5 \vee x_2x_4x_5 \vee x_1x_2x_3x_6 \vee x_1x_3x_5x_6 \vee x_2x_3x_4x_6 \vee x_3x_4x_5x_6 \quad \text{(C.6)}$$

Essentially, in all carry problems the lower order accurate subsolution is of order two (specifying either both left bits as zero or as one) and the maximal order accurate subsolution is of order $k+1$ specifying that a carry happens due to the propagation of the carry starting right at the back of the number (see the latter conjunctions in the DNF-form). In effect, the size of the optimal problem solution is $\sum_{i=1}^{k} 2^{i-1} = 2^k - 1$ to represent class one and $2^k + \sum_{i=1}^{k-1} 2^{i-1} = 2^k + 2^{k-1} - 1$ to represent class zero. Thus, the optimal solution representation in XCS comprises $|[O]| = 2(2^k - 1 + 2^k + 2^{k-1} - 1) = 52^k - 4$ classifiers.

Similar to the count ones problems, the carry problem provides helpful fitness guidance in that the specialization of only ones or only zeros increases accuracy (decreasing entropy in the class distribution). Thus, the minimal order of difficulty $k_m = 1$ and the order of difficulty $k_d = k + 1$. Due to the overlapping nature and the unequally sized niches in the problem, XCS requires larger population sizes to ensure niche support as evaluated in Chapter 5.

Hierarchically Composed Problems

We introduced general hierarchal problems in Chapter 6 showing that they require effective BB processing in XCS. The hierarchical problems are designed in a two-level hierarchy in which the lower-level is evaluated by one set of Boolean functions and the output of the lower-level is then fed as input to the higher level.

For example, we can combine parity problems on the lower level with a multiplexer problem on the higher level. The evaluation takes a problem instance and evaluates chunks of bits using the lower level function. The result of the lower level function results in a shorter bit string which is then evaluated using the higher level function. For example, consider the hierarchical 2-parity, 3-multiplexer problem. The problem is six bits long. On the lower level, blocks of two bits are evaluated by the parity function, that is, if there are an even

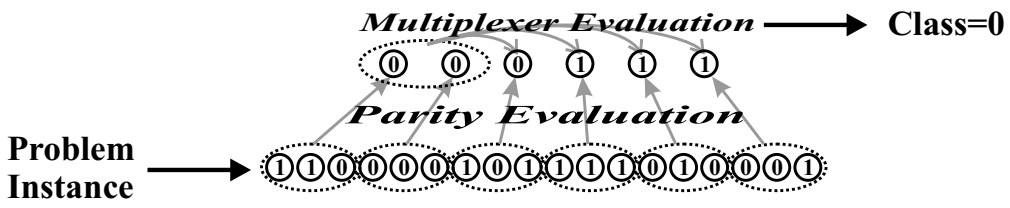


Figure C.1: Illustration of an exemplar hierarchical 3-parity, 6-multiplexer problem evaluation

number of ones, the result will be zero and one otherwise. The result is a string of three bits that is then evaluated by the 3-multiplexer function. The result is the class of the problem instance. The hierarchical 2-parity, 3-multiplexer problem can be written in DNF as follows:

$$\texttt{2-PA,3-MP}(x_1, x_2, x_3, x_4, x_5, x_6) = x_1x_2\neg x_3x_4 \vee x_1x_2x_3\neg x_4 \vee \neg x_1\neg x_2\neg x_3x_4 \vee$$
$$\neg x_1\neg x_2x_3\neg x_4 \vee \neg x_1x_2\neg x_5x_6 \vee \neg x_1x_2x_5\neg x_6 \vee x_1\neg x_2\neg x_5x_6 \vee x_1\neg x_2x_5\neg x_6 \quad \text{(C.7)}$$

Figure C.1 shows the structure of the hierarchical 3-parity-6-multiplexer problem. The figure shows the tight coding of the problem: All 3-parity chunks are coded in one block. Certainly, the property of tight coding may not be given in a natural problem and thus our learning algorithm should not rely on this property.

With respect to problem difficulty, generally the previously determined sizes carry over only that each problem feature is now represented by a lower-level BB of features defined by the lower level functions. In the k-parity-k'-multiplexer combination, effectively the optimal population is of size $|[O]| = 2(2^{k(k'+1)})$. The minimal order of difficulty $k_m = k$ and the problem order of difficulty $k_d = k(k'+1)$.

Bibliography

Aha, D. W., Kibler, D., & Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning*, *6*, 37–66.

Bacardit, J., & Butz, M. V. (2004). *Data mining in learning classifier systems: Comparing XCS with GAssist* (IlliGAL report 2004030). University of Illinois at Urbana-Champaign: Illinois Genetic Algorithms Laboratory.

Bäck, T., & Schwefel, H.-P. (1995). Evolution strategies I: Variants and their computational implementation. In Winter, G., Périaux, J., Gal'an, M., & Cuesta, P. (Eds.), *Genetic algorithms in engineering and computer science* (Chapter 7, pp. 111–126). Chichester: John Wiley & Sons.

Baird, L. C. (1995). Residual algorithms: Reinforcement learning with function approximation. *Proceedings of the Twelfth International Conference on Machine Learning*, 30–37.

Baird, L. C. (1999). *Reinforcement learning through gradient descent*. Doctoral dissertation, School of Computer Science. Carnegie Mellon University, Pittsburgh, PA 15213.

Baker, J. (1985). Adaptive selection methods for genetic algorithms. In Grefenstette, J. J. (Ed.), *Proceedings of the international conference on genetic algorithms and their applications* (pp. 14–21). Hillsdale, NJ: Lawrence Erlbaum Associates.

Barto, A. G., & Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, *13*, 341–379.

Bellman, R. (1957). *Dynamic programming*. Princeton, NY: Princeton University Press.

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

Bernadó-Mansilla, E., & Garrell-Guiu, J. M. (2003). Accuracy-based learning classifier systems: Models, analysis, and applications to classification tasks. *Evolutionary Computation*, *11*, 209–238.

Blake, C., Keogh, E., & Merz, C. (1998). UCI repository of machine learning databases. (http://www.ics.uci.edu/mlearn/MLRepository.html).

Booker, L. (1982). *Improving behavior as a adaptation to the task environment*. Doctoral dissertation, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor.

Booker, L. B. (1993). Recombination distributions for genetic algorithms. *Foundations of Genetic Algorithms*, *2*, 29–44.

Booker, L. B., Goldberg, D. E., & Holland, J. H. (1989). Classifier systems and genetic algorithms. *Artificial Intelligence*, *40*, 235–282.

Bridges, C. L., & Goldberg, D. E. (1987). An analysis of reproduction and crossover in a binary-coded genetic algorithm. *Proceedings of the Second International Conference on Genetic Algorithms*, 9–13.

Bull, L. (2003). *Investigating fitness sharing in a simple payoff-based learning classifier system* (Technical Report UWELCSG03-009). Bristol, UK: University of Western England, Learning Classifier System Group.

Bull, L., & Hurst, J. (2002). ZCS redux. *Evolutionary Computation*, *10*(2), 185–205.

Buntine, W. L. (1991). Theory refinement of Bayesian networks. *Proceedings of the Uncertainty in Artificial Intelligence (UAI-91)*, 52–60.

Butz, M. V. (2002). *Anticipatory learning classifier systems*. Boston, MA: Kluwer Academic Publishers.

Butz, M. V. (2004a). Anticipation for learning, cognition, and education. *On the Horizon*. in press.

Butz, M. V. (2004b). *COSEL: A cognitive sequence learning architecture* (IlliGAL report 2004021). University of Illinois at Urbana-Champaign: Illinois Genetic Algorithms Laboratory.

Butz, M. V., & Goldberg, D. E. (2003). Bounding the population size in XCS to ensure reproductive opportunities. *Proceedings of the Fifth Genetic and Evolutionary Computation Conference (GECCO-2003)*, 1844–1856.

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

Butz, M. V., Goldberg, D. E., & Lanzi, P. L. (2004b). Gradient-based learning updates improve XCS performance in multistep problems. *Proceedings of the Sixth Genetic and Evolutionary Computation Conference (GECCO-2004): Part II*, 751–762.

Butz, M. V., Goldberg, D. E., & Lanzi, P. L. (2004c). Gradient Descent Methods in Learning Classifier Systems: Improving XCS Performance in Multistep Problems. *IEEE Transactions on Evolutionary Computation*. in press.

Butz, M. V., Goldberg, D. E., & Stolzmann, W. (2002). The anticipatory classifier system and genetic generalization. *Natural Computing*, *1*, 427–467.

Butz, M. V., Goldberg, D. E., & Tharakunnel, K. (2003). Analysis and improvement of fitness exploitation in XCS: Bounding models, tournament selection, and bilateral accuracy. *Evolutionary Computation*, *11*, 239–277.

Butz, M. V., Kovacs, T., Lanzi, P. L., & Wilson, S. W. (2001). How XCS evolves accurate classifiers. *Proceedings of the Third Genetic and Evolutionary Computation Conference (GECCO-2001)*, 927–934.

Butz, M. V., Kovacs, T., Lanzi, P. L., & Wilson, S. W. (2004). Toward a theory of generalization and learning in XCS. *IEEE Transactions on Evolutionary Computation*, *8*, 28– 46.

Butz, M. V., & Pelikan, M. (2001). Analyzing the evolutionary pressures in XCS. *Proceedings of the Third Genetic and Evolutionary Computation Conference (GECCO-2001)*, 935–942.

Butz, M. V., Sastry, K., & Goldberg, D. E. (2003). Tournament selection in XCS. *Proceedings of the Fifth Genetic and Evolutionary Computation Conference (GECCO-2003)*, 1857–1869.

Butz, M. V., Sastry, K., & Goldberg, D. E. (2004). Strong, stable, and reliable fitness pressure in XCS due to tournament selection. *Genetic Programming and Evolvable Machines*, *6*, 53–77.

Butz, M. V., Sigaud, O., & Gérard, P. (2003). Internal models and anticipations in adaptive learning systems. In Butz, M. V., Sigaud, O., & Gérard, P. (Eds.), *Anticipatory Behavior in Adaptive Learning Systems: Foundations, Theories, and Systems* (pp. 86–109).

Butz, M. V., Sigaud, O., & Swarup, S. (Eds.) (2004). *Anticipatory behavior in adaptive learning systems, ABiALS 2004 workshop proceedings.*

Butz, M. V., Swarup, S., & Goldberg, D. E. (2004). *Effective online detection of task-independent landmarks* (IlliGAL report 2004002). University of Illinois at Urbana-Champaign: Illinois Genetic Algorithms Laboratory.

Butz, M. V., & Wilson, S. W. (2002). An algorithmic description of XCS. *Soft Computing*, *6*, 144–153.

Chickering, D. M., Heckerman, D., & Meek, C. (1997). *A Bayesian approach to learning Bayesian networks with local structure* (Technical Report MSR-TR-97-07). Redmond, WA: Microsoft Research.

Cooper, G. F., & Herskovits, E. H. (1992). A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, *9*, 309–347.

Darwin, C. (1968 (orig. 1859)). *The origin of species by means of natural selection.* Penguin Books.

De Jong, K. A. (1975). *An analysis of the behavior of a class of genetic adaptive systems.* Doctoral dissertation, University of Michigan, Ann Arbor. University Microfilms No. 76-9381.

De Jong, K. A., & Spears, W. M. (1991). Learning concept classification rules using genetic algorithms. *IJCAI-91 Proceedings of the Twelfth International Conference on Artificial Intelligence*, 651–656.

Dietterich, T. G. (1997). Machine learning research: Four current directions. *AI Magazine*, *18*(4), 97–136.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, *13*, 227–303.

Dittenbach, M., Merkl, D., & Rauber, A. (2000). The growing hierarchical self-organizing map. *Proceedings of the International Joint Conference on Neural Networks*, *VI*, 15–19.

Drummond, C. (2002). Accelerating reinforcement learning by composing solutions of automatically identified subtasks. *Journal of Artificial Intelligence Research*, *16*, 59–104.

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern classification* (2 ed.). New York, NY: John Wiley & Sons.

Feng, A. S., & Ratnam, R. (2000). Neural basis of hearing in real-world situations. *Annual Review of Psychology*, *51*, 699–725.

Frank, E., & Witten, I. H. (1998). Generating accurate rule sets without global optimization. *Proceedings of the Fifteenth International Conference on Machine Learning*, 144–151.

Friedman, N., & Goldszmidt, M. (1999). Learning Bayesian networks with local structure. In Jordan, M. I. (Ed.), *Graphical Models* (pp. 421–459). Cambridge, MA: MIT Press.

Gelb, A., Kasper, J. F., Nash, R. A., Price, C. F., & Sutherland, A. A. (Eds.) (1974). *Applied optimal estimation*. Cambridge, MA: MIT Press.

Gérard, P., & Sigaud, O. (2001a). Adding a generalization mechanism to YACS. *Proceedings of the Third Genetic and Evolutionary Computation Conference (GECCO-2001)*, 951–957.

Gérard, P., & Sigaud, O. (2001b). YACS: Combining dynamic programming with generalization in classifier systems. In Lanzi, P. L., Stolzmann, W., & Wilson, S. W. (Eds.), *Advances in learning classifier systems: Third international workshop, IWLCS 2000 (LNAI 1996)* (pp. 52–69). Berlin Heidelberg: Springer-Verlag.

Gérard, P., & Sigaud, O. (2003). Designing efficient exploration with MACS: Modules and function approximation. *Proceedings of the Fifth Genetic and Evolutionary Computation Conference (GECCO-2003)*, 1882–1893.

Goldberg, D., & Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. *Proceedings of the Second International Conference on Genetic Algorithms*, 41–49.

Goldberg, D. E. (1983). Computer-aided gas pipeline operation using genetic algorithms and rule learning. *Dissertation Abstracts International*, *44*(10), 3174B. Doctoral dissertation, University of Michigan.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*. Reading, MA: Addison-Wesley.

Goldberg, D. E. (1990). Probability matching, the magnitude of reinforcement, and classifier system bidding. *Machine Learning*, *5*(4), 407–425.

Goldberg, D. E. (1991). Genetic algorithms as a computational theory of conceptual design. In Rzevski, G., & Adey, R. A. (Eds.), *Applications of Artificial Intelligence in*

Engineering VI (pp. 3–16). Boston and New York: Computational Mechanics Publications and Elsevier Applied Science.

Goldberg, D. E. (1999). The race, the hurdle and the sweet spot: Lessons from genetic algorithms for the automation of innovation and creativity. In Bentley, P. (Ed.), *Evolutionary design by computers* (pp. 105–118). San Francisco, CA: Morgan Kaufmann.

Goldberg, D. E. (2002). *The design of innovation: Lessons from and for competent genetic algorithms*. Boston, MA: Kluwer Academic Publishers.

Goldberg, D. E., & Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. *Foundations of Genetic Algorithms*, 69–93.

Goldberg, D. E., Deb, K., & Clark, J. H. (1992). Genetic algorithms, noise, and the sizing of populations. *Complex Systems*, *6*, 333–362.

Goldberg, D. E., Deb, K., & Thierens, D. (1993). Toward a better understanding of mixing in genetic algorithms. *Journal of the Society of Instrument and Control Engineers*, *32*(1), 10–16.

Goldberg, D. E., & Sastry, K. (2001). A practical schema theorem for genetic algorithm design and tuning. *Proceedings of the Third Genetic and Evolutionary Computation Conference (GECCO-2001)*, 328–335.

Grossberg, S. (1976a). Adaptive pattern classification and universal recoding, I: Parallel development and coding of neural feature detectors. *Biological Cybernetics*, *23*, 121–134.

Grossberg, S. (1976b). Adaptive pattern classification and universal recoding, II: Feedback, expectation, olfaction, and illusions. *Biological Cybernetics*, *23*, 197–202.

Harik, G. (1994). *Finding multiple solutions in problems of bounded difficulty* (IlliGAL report 94002). University of Illinois at Urbana-Champaign: Illinois Genetic Algorithms Laboratory.

Harik, G. (1999). *Linkage learning via probabilistic modeling in the ECGA* (IlliGAL report 99010). University of Illinois at Urbana-Champaign: Illinois Genetic Algorithms Laboratory.

Hassoun, M. H. (1995). *Fundamentals of artificial neural networks*. Cambridge, MA: MIT Press.

Heckerman, D., Geiger, D., & Chickering, D. M. (1994). *Learning Bayesian networks: The combination of knowledge and statistical data* (Technical Report MSR-TR-94-09).

Redmond, WA: Microsoft Research.

Henrion, M. (1988). Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In Lemmer, J. F., & Kanal, L. N. (Eds.), *Uncertainty in Artificial Intelligence* (pp. 149–163). Amsterdam, London, New York: Elsevier.

Hoffmann, J. (1993). *Vorhersage und Erkenntnis: Die Funktion von Antizipationen in der menschlichen Verhaltenssteuerung und Wahrnehmung. [Anticipation and cognition: The function of anticipations in human behavioral control and perception.]*. Göttingen, Germany: Hogrefe.

Hoffmann, J. (2003). Anticipatory behavioral control. In Butz, M. V., Sigaud, O., & Gérard, P. (Eds.), *Anticipatory Behavior in Adaptive Learning Systems: Foundations, Theories, and Systems* (pp. 44–65). Berlin Heidelberg: Springer-Verlag.

Holland, J. H. (1971). Processing and processors for schemata. In Jacks, E. L. (Ed.), *Associative Information Techniques* (pp. 127–146). New York: American Elsevier.

Holland, J. H. (1975). *Adaptation in natural and artificial systems.* Ann Arbor, MI: University of Michigan Press. second edition, 1992.

Holland, J. H. (1976). Adaptation. In Rosen, R., & Snell, F. (Eds.), *Progress in theoretical biology*, Volume 4 (pp. 263–293). New York: Academic Press.

Holland, J. H. (1977). *A cognitive system with powers of generalization and adaptation.* Unpublished manuscript.

Holland, J. H., & Reitman, J. S. (1978). Cognitive systems based on adaptive algorithms. In Waterman, D. A., & Hayes-Roth, F. (Eds.), *Pattern directed inference systems* (pp. 313–329). New York: Academic Press.

Holte, R. C. (1993). Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, *11*, 63–90.

Hommel, B., Müsseler, J., Aschersleben, G., & Prinz, W. (2001). The theory of event coding (TEC): A framework for perception and action planning. *Behavioral and Brain Sciences*, *24*, 849–878.

Horn, J. (1993). Finite Markov chain analysis of genetic algorithms with niching. *Proceedings of the Fifth International Conference on Genetic Algorithms*, 110–117.

Horn, J., Goldberg, D. E., & Deb, K. (1994). Implicit niching in a learning classifier system: Nature's way. *Evolutionary Computation*, *2*(1), 37–66.

Howard, R. A., & Matheson, J. E. (1981). Influence diagrams. In Howard, R. A., & Matheson, J. E. (Eds.), *Readings on the principles and applications of decision analysis*, Volume II (pp. 721–762). Menlo Park, CA: Strategic Decisions Group.

Jaeger, H. (2000). Observable operator models for discrete stochastic time series. *Neural Computation*, *12*, 1371–1398.

Jaeger, H. (2004). Online learning algorithms for observable operator models.

James, M. R., & Singh, S. (2004). Learning and discovery of predictive state representations in dynamical systems with reset. *Proceedings of the Twenty-First International Conference on Machine Learning (ICML-2004)*, 417–424.

John, G. H., & Langley, P. (1995). Estimating continuous distributions in Bayesian classifiers. *11th Conference on Uncertainty in Artificial Intelligence*, 338–345.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, *4*, 237–285.

Kersting, K., Van Otterlo, M., & De Raedt, L. (2004). Bellman goes relational. *Proceedings of the Twenty-First International Conference on Machine Learning (ICML-2004)*, 465–472.

Kleinrock, L. (1975). *Queueing systems: Theory*. New York: John Wiley & Sons.

Kovacs, T. (1996). *Evolving optimal populations with XCS classifier systems*. Master's thesis, School of Computer Science, University of Birmingham, Birmingham, U.K.

Kovacs, T. (1997). XCS classifier system reliably evolves accurate, complete, and minimal representations for boolean functions. In Roy, Chawdhry, & Pant (Eds.), *Soft computing in engineering design and manufacturing* (pp. 59–68). Springer-Verlag, London.

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

Kovacs, T. (2000). Strength or Accuracy? Fitness calculation in learning classifier systems. In Lanzi, P. L., Stolzmann, W., & Wilson, S. W. (Eds.), *Learning classifier systems: From foundations to applications (LNAI 1813)* (pp. 143–160). Berlin Heidelberg: Springer-Verlag.

Kovacs, T. (2001). Towards a theory of strong overgeneral classifiers. *Foundations of Genetic Algorithms 6*, 165–184.

Kovacs, T. (2003). *Strength or accuracy: Credit assignment in learning classifier systems*. Berlin Heidelberg: Springer-Verlag.

Kovacs, T., & Kerber, M. (2001). What makes a problem hard for XCS? In Lanzi, P. L., Stolzmann, W., & Wilson, S. W. (Eds.), *Advances in learning classifier systems: Third international workshop, IWLCS 2000 (LNAI 1996)* (pp. 80–99). Berlin Heidelberg: Springer-Verlag.

Kovacs, T., & Yang, C. (2004). An initial study of a model-based XCS. http://www.psychologie.uni-wuerzburg.de/IWLCS/.

Kunde, W., Koch, I., & Hoffmann, J. (2004). Anticipated action effects affect the selection, initiation, and execution of actions. *The Quarterly Journal of Experimental Psychology. Section A: Human Experimental Psychology*, *57*, 87–106.

Lanzi, P. L. (1997). A study of the generalization capabilities of XCS. *Proceedings of the Seventh International Conference on Genetic Algorithm*, 418–425.

Lanzi, P. L. (1999a). An analysis of generalization in the XCS classifier system. *Evolutionary Computation*, *7*(2), 125–149.

Lanzi, P. L. (1999b). Extending the Representation of Classifier Conditions Part II: From Messy Coding to S-Expressions. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, 345–352.

Lanzi, P. L. (1999c). An extension to the XCS classifier system for stochastic environments. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, 353–360.

Lanzi, P. L. (2000). Adaptive agents with reinforcement learning and internal memory. *From Animals to Animats 6: Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior*, 333–342.

Lanzi, P. L. (2002). Learning classifier systems from a reinforcement learning perspective. *Soft Computing: A Fusion of Foundations, Methodologies and Applications*, *6*, 162–170.

Lanzi, P. L., & Wilson, S. W. (2000). Toward optimal classifier system performance in non-Markov environments. *Evolutionary Computation*, *8*(4), 393–418.

Larrañaga, P. (2002). A review on estimation of distribution algorithms. In Larrañaga, P., & Lozano, J. A. (Eds.), *Estimation of Distribution Algorithms* (Chapter 3, pp. 57–100). Boston, MA: Kluwer Academic Publishers.

Littman, M. L., Sutton, R. S., & Singh, S. (2002). Predictive representations of state. *Advances in Neural Information Processing Systems*, *14*, 1555–1561.

Llorà, X., & Goldberg, D. E. (2002). *Minimal achievable error in the LED problem* (IlliGAL report 2002015). University of Illinois at Urbana-Champaign: Illinois Genetic Algorithms Laboratory.

Llorà, X., & Goldberg, D. E. (2003). Bounding the effect of noise in multiobjective learning classifier systems. *Evolutionary Computation*, *11*, 279–298.

Llorà, X., Goldberg, D. E., Traus, I., & Bernadó, E. (2003). Accuracy, Parsimony, and Generality in Evolutionary Learning Systems via Multiobjective Selection. In Lanzi, P. L., Stolzmann, W., & Wilson, S. W. (Eds.), *Learning classifier system: Fifth international workshop, IWLCS 2002 (LNAI 2661)* (pp. 118–142). Berlin Heidelberg: Springer-Verlag.

Lobo, F., & Harik, G. (1999). *Extended compact genetic algorithm in C++* (IlliGAL report 99016). University of Illinois at Urbana-Champaign: Illinois Genetic Algorithms Laboratory.

Mahfoud, S. W. (1992). Crowding and preselection revisited. *Parallel Problem Solving from Nature*, 27–36.

Mahfoud, S. W. (1995). *Niching methods for genetic algorithms*. Doctoral dissertation, University of Illinois at Urbana-Champaign.

Mitchell, M., Forrest, S., & Holland, J. H. (1991). The royal road for genetic algorithms: Fitness landscapes and GA performance. In Varela, F. V., & Bourgine, P. (Eds.), *Toward a Practice of Autonomous Systems: First European Conference on Artificial Life* (pp. 245–254). Cambridge, MA: MIT Press.

Mitchell, T. M. (1997). *Machine learning*. Boston, MA: McGraw-Hill.

Moore, A. W., & Atkeson, C. (1993). Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, *13*, 103–130.

Neal, R. M. (1993). *Probabilistic inference using Markov chain Monte Carlo methods* (Technical Report CRG-TR-93-1). Dept. of Computer Science, University of Toronto.

Pashler, H., Johnston, J. C., & Ruthruff, E. (2001). Attention and performance. *Annual Review of Psychology*, *52*, 629–651.

Pashler, H. E. (1998). *The psychology of attention*. Cambridge, MA: MIT Press.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. San Mateo, CA: Morgan Kaufmann.

Pelikan, M. (2001). Bayesian optimization algorithm, decision graphs, and Occam's razor. *Proceedings of the Third Genetic and Evolutionary Computation Conference (GECCO-2001)*, 511–518.

Pelikan, M. (2002). *Bayesian optimization algorithm: From single level to hierarchy*. Doctoral dissertation, University of Illinois at Urbana-Champaign, Urbana, IL. Also IlliGAL Report No. 2002023.

Pelikan, M., Goldberg, D. E., & Cantu-Paz, E. (1999). BOA: The Bayesian optimization algorithm. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, 525–532.

Pelikan, M., Goldberg, D. E., & Lobo, F. (2002). A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, *21*(1), 5–20.

Platt, J. (1998). Fast training of support vector machines using sequential minimal optimization. In Schlkopf, B., Burges, C., & Smola, A. (Eds.), *Advances in Kernel Methods – Support Vector Learning* (pp. 42–65). Cambridge, MA: MIT Press.

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

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Francisco, CA: Morgan Kaufmann.

Rechenberg, I. (1973). *Evolutionsstrategie Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart-Bad Cannstatt: Friedrich Frommann Verlag.

Sastry, K., & Goldberg, D. E. (2000). *On extended compact genetic algorithm* (IlliGAL report 2000026). University of Illinois at Urbana-Champaign: Illinois Genetic Algorithms Laboratory.

Schwarz, G. (1978). Estimating the dimension of a model. *The Annals of Statistics*, *6*, 461–464.

Servedio, R. A. (2001). *Efficient algorithms in computational learning theory*. Doctoral dissertation, Harvard University, Cambridge, MA.

Shawe-Taylor, J., & Cristianini, N. (2004). *Kernel methods for pattern analysis*. Cambridge, UK: Cambridge University Press.

Simon, H. A. (1969). *Sciences of the artificial*. Cambridge, MA: MIT Press.

Simsek, O., & Barto, A. G. (2004). Using relative novelty to identify useful temporal abstractions in reinforcement learning. *Proceedings of the Twenty-First International Conference on Machine Learning (ICML-2004)*, 751–758.

Singh, S., Littman, M. L., Jong, N. K., & Pardoe, D. (2003). Learning predictive state representations. *Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003)*, 712–719.

Stolzmann, W. (1998). Anticipatory classifier systems. *Genetic Programming 1998: Proceedings of the Third Annual Conference*, 658–664.

Stone, C., & Bull, L. (2003). For real! XCS with continuous-values inputs. *Evolutionary Computation*, *11*, 299–336.

Sutton, R., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*, 181–211.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. *Proceedings of the Seventh International Conference on Machine Learning*, 216–224.

Sutton, R. S. (1991). Reinforcement learning architectures for animats. *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, 288–296.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

Sutton, R. S., & Singh, S. (Eds.) (2004). *Proceedings for the ICML'04 workshop on predictive representations of world knowledge*.

Taylor, J. G. (2002). From matter to mind. *Journal of Consciousness Studies*, *9*, 3–22.

Thierens, D., & Goldberg, D. E. (1993). Mixing in genetic algorithms. *Proceedings of the Fifth International Conference on Genetic Algorithms*, 38–45.

Thierens, D., Goldberg, D. E., & Pereira, A. G. (1998). Domino convergence, drift, and the temporal-salience structure of problems. In *Proceedings of the 1998 IEEE World Congress on Computational Intelligence* (pp. 535–540). New York, NY: IEEE Press.

Tolman, E. C. (1932). *Purposive behavior in animals and men*. New York: Appleton.

Valiant, L. (1984). A theory of the learnable. *Communications of the ACM*, *27*, 1134–1142.

Venturini, G. (1994). Adaptation in dynamic environments through a minimal probability of exploration. *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, 371–381.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. Doctoral dissertation, King's College, Cambridge, UK.

Whitehead, S. D., & Ballard, D. H. (1991). Learning to perceive and act by trial and error. *Machine Learning*, *7*(1), 45–83.

Wilson, S. W. (1985). Knowledge growth in an artificial animal. *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, 16–23.

Wilson, S. W. (1987a). Classifier systems and the animat problem. *Machine Learning*, *2*, 199–228.

Wilson, S. W. (1987b). The genetic algorithm and simulated evolution. In Langton, C. G. (Ed.), *Artificial Life*, Volume VI (pp. 157–166). Addison-Wesley.

Wilson, S. W. (1991). The animat path to AI. *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, 15–21.

Wilson, S. W. (1994). ZCS: A zeroth level classifier system. *Evolutionary Computation*, *2*, 1–18.

Wilson, S. W. (1995). Classifier fitness based on accuracy. *Evolutionary Computation*, *3*(2), 149–175.

Wilson, S. W. (1998). Generalization in the XCS classifier system. *Genetic Programming 1998: Proceedings of the Third Annual Conference*, 665–674.

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

Wilson, S. W. (2001a). Function approximation with a classifier system. *Proceedings of the Third Genetic and Evolutionary Computation Conference (GECCO-2001)*, 974–981.

Wilson, S. W. (2001b). Mining oblique data with XCS. In Lanzi, P. L., Stolzmann, W., & Wilson, S. W. (Eds.), *Advances in learning classifier systems: Third international workshop, IWLCS 2000 (LNAI 1996)* (pp. 158–174). Berlin Heidelberg: Springer-Verlag.

Wilson, S. W. (2004). Classifier systems for continuous payoff environments. *Proceedings of the Sixth Genetic and Evolutionary Computation Conference (GECCO-2004): Part II*, 824–835.

Wilson, S. W., & Goldberg, D. E. (1989). A critical review of classifier systems. *Proceedings of the Third International Conference on Genetic Algorithms*, 244–255.

Witten, I. H., & Frank, E. (2000). *Data mining. Practical machine learning tools and techniques with java implementations*. San Francisco, CA: Morgan Kaufmann.

Vita

Martin Volker Butz was born in Würzburg, Germany on August 4, 1975. He graduated high-school from the Gymnasium in Bad Königshofen in July, 1995. During high-school he spent the 1992/1993 school year in Cape Town, South Africa as an exchange student. Butz then commenced his undergraduate studies in computer science with a minor in psychology at the Bayerische Julius-Maximilians Universität Würzburg in fall 1995. During his studies he spent one year at the Illinois Genetic Algorithms Laboratory (IlliGAL) as a visiting scholar. He graduated from the Bayerische Julius-Maximilians Universität Würzburg with honors in August, 2001. Since then, he has been a research assistant at the department of cognitive psychology at the Bayerische Julius-Maximilians Universität Würzburg. In January, 2002, Butz joined the University of Illinois at Urbana-Champaign for his graduate studies. Following the completion of his Ph.D. degree, Butz will engage in postdoctoral research focusing on machine learning and cognitive science.